Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет
(НПИ) имени М. И. Платова

МАТЕМАТИЧЕСКИЕ И АЛГОРИТМИЧЕСКИЕ ОСНОВЫ ПРИКЛАДНОЙ КРИПТОГРАФИИ

Учебно-методическое пособие

Новочеркасск
ЮРГПУ (НПИ)
2021

УДК 004.056.5 (075.8)
ББК 32.811.4я73
С 72

Рецензент – канд. техн. наук, доцент Синецкий Р.М.

Спиридонова И.А., Кущий Д.Н.

С 72 Математические и алгоритмические основы прикладной криптографии : учебно-методическое пособие / И.А. Спиридонова, Д.Н. Кущий; Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2021. – 66 с.

В учебно-методическом пособии содержатся теоретические сведения и практические примеры, необходимые для обучения студентов основам компьютерной информационной безопасности, защиты информации от несанкционированного доступа и обеспечения конфиденциальности информационного обмена в информационно-вычислительных системах. Изложены математические основы криптографии, проиллюстрированные соответствующими числовыми примерами. Представлен учебный материал, посвященный основам построения, принципам реализации и классическим примерам криптографических систем и криптографических протоколов, в том числе созданию и использованию электронной цифровой подписи.

УДК 004.056.5 (075.8)
ББК 32.811.4я73

ВВЕДЕНИЕ

Компьютерные информационные технологии внедряются во все сферы деятельности человеческого общества. Все чаще информация рассматривается как товар, который можно приобрести, продать, обменять и т.п. Зачастую, стоимость информации во много раз превосходит стоимость компьютерной системы, в которой она хранится и обрабатывается. Поскольку информация может быть очень ценной или особо важной, ее владельцу необходимо принимать во внимание разнообразные злонамеренные действия со стороны злоумышленников по отношению к компьютерным системам, хранящим, обрабатывающим или передающим такую информацию. Например, нарушитель может попытаться выдать себя за другого пользователя системы, подслушать канал связи, перехватить и изменить информацию, которой обмениваются пользователи системы, попытаться расширить свои полномочия, чтобы получить доступ к информации, к которой ему предоставлен только частичный доступ, попытаться разрушить систему.

По мере развития и усложнения методов, средств и форм автоматизации процессов обработки информации повышается зависимость общества от степени безопасности используемых им информационных технологий. Интенсивное развитие открытых компьютерных сетей привлекает все большее внимание к ним пользователей. Большинство компаний и организаций подключают свои внутренние локальные сети к сети Internet, чтобы воспользоваться ее ресурсами и преимуществами. Однако при подключении к открытой сети Internet возникают серьезные проблемы с обеспечением информационной безопасности подключаемой к ней локальной или корпоративной сети. Сеть Internet предоставляет злоумышленникам множество возможностей для вторжения во внутренние

сети компаний и организаций с целью хищения, искажения или разрушения важной и конфиденциальной информации. В связи с этим, в настоящее время все большую актуальность приобретает [1–5] необходимость в защите информации от несанкционированного доступа, умышленного изменения, кражи, уничтожения и других преступных действий.

В данном учебно-методическом пособии представлено изложение материала по математическим и алгоритмическим основам прикладной криптографии с примерами реализации ключевых алгоритмов математических преобразований криптографии и базовых криптографических систем.

В первой главе представлено краткое изложение теоретического материала по базовым математическим основам прикладной криптографии с примерами *пошаговой* реализации алгоритмов математических преобразований криптографии. Подробно рассмотрены наиболее *практически значимые* алгоритмы преобразований модулярной арифметики: «Расширенный алгоритм Евклида» и «Метод повторного возведения в квадрат».

Во второй главе рассмотрены криптографические системы и криптографические протоколы, их особенности и области применения. Подробно рассмотрены алгоритмы и примеры применения *базовых элементов криптографических систем*.

В качестве примеров при рассмотрении алгоритмических основ криптографии подробно представлены реализации криптографической системы с протоколом Шамира и криптографической системы *RSA*, а также создание и использование электронной цифровой подписи (ЭЦП) по схеме Эль-Гамаля.

1. МАТЕМАТИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ

1.1. Наибольший общий делитель двух целых чисел. Классический алгоритм Евклида

Одним из широко и постоянно используемых условий выбора числовых параметров в криптографических системах является взаимная простота чисел.

Иногда практическое выполнение этого условия требует предварительной проверки, но, как правило, именно это условие выполняется выбором чисел из таблиц простых чисел, широко представленных в сети Интернет. Тем не менее для дальнейшего понимания сущности и особенностей *расширенного* алгоритма Евклида, широко и постоянно использующегося в криптографических системах, предварительное освоение *классического* алгоритма Евклида является немаловажным.

Определение. Наибольший общий делитель двух данных чисел a и b (обозначается $\text{НОД}(a,b)$) есть единственное целое положительное число, которое делит a и b, и делится при этом на любой их общий делитель.

Определение. Два целых числа a и b называются взаимно простыми, если $\text{НОД}(a,b)=1$, т.е. если они не имеют общих делителей больших 0.

Задание. Найти наибольший общий делитель $\text{НОД}(a,b)$ двух чисел $a \in Z$ (a – целое число), $b \in N$ (b – число натурального ряда, т.е. целое *положительное* число) методом Евклида.

Алгоритм и расчетные формулы метода Евклида:

$$r_{j-2} = d_j r_{j-1} + r_j,\ d_j = \left[\frac{r_{j-2}}{r_{j-1}} \right],\ j = 0,1,2\ldots,$$

где $r_{-2} = a,\ r_{-1} = b$.

Решением задачи является последний *ненулевой* остаток $r_j > 0$.

Выполнение алгоритма метода Евклида представлено на конкретном примере в табл. 1.1.

Пример 1.1.1. Найти $\text{НОД}(1547, 560)$.

Таблица 1.1 – Выполнение алгоритма Евклида для примера

Шаг	Кратность	Разложение	Остаток
$j=1$	$d_1 = \left[\frac{a}{b}\right] = \left[\frac{1547}{560}\right] = 2$	$a = d_1 b + r_0$: $1547 = 2 \cdot 560 + 427$	$r_0 = 427$
$j=2$	$d_2 = \left[\frac{b}{r_0}\right] = \left[\frac{560}{427}\right] = 1$	$b = d_2 r_0 + r_1$: $560 = 1 \cdot 427 + 133$	$r_1 = 133$
$j=3$	$d_3 = \left[\frac{r_0}{r_1}\right] = \left[\frac{427}{133}\right] = 3$	$r_0 = d_3 r_1 + r_2$: $427 = 3 \cdot 133 + 28$	$r_2 = 28$
$j=4$	$d_4 = \left[\frac{r_1}{r_2}\right] = \left[\frac{133}{28}\right] = 4$	$r_1 = d_4 r_2 + r_3$: $133 = 4 \cdot 28 + 21$	$r_3 = 21$
$j=5$	$d_5 = \left[\frac{r_2}{r_3}\right] = \left[\frac{28}{21}\right] = 1$	$r_2 = d_5 r_3 + r_4$: $28 = 1 \cdot 21 + 7$	$r_4 = 7$
$j=6$	$d_6 = \left[\frac{r_3}{r_4}\right] = \left[\frac{21}{7}\right] = 3$	$r_3 = d_6 r_4 + r_5$: $21 = 3 \cdot 7 + 0$	$r_5 = 0$

Получено: $r_5 = 0$, следовательно, выполнение алгоритма окончено. Решением является предшествующий (ненулевой) остаток $r_4 = 7$:

$$\text{НОД}(1547, 560) = 7.$$

1.2. Модулярная арифметика. Вычеты в кольце

Основной особенностью математических алгоритмов в криптографии является работа с конечным множеством целых чисел, представляющих собой номера позиций символов алфавита для шифрования сообщений.

Количество символов алфавита определяет основание кольца, в котором для любого большого числа определяется вычет, т.е. номер символа, не превышающий основание. Полностью «пройденное кольцо» (возможно и «пройденное» при этом несколько раз) при этом просто отбрасывается и на конечный результат никак не влияет. В кольце по модулю p вычетами могут быть целые числа из множества $\{0,...,p-1\}$. Отрицательное число, полученное при преобразованиях модулярной арифметики означает, что позиция нужного нам символа отсчитывается не от начала кольца, а от его конца и в конце преобразований может быть всегда переведено в положительный вычет при помощи простого действия прибавления основания кольца, т.е. самого числа p.

Пример 1.2.1. Определить вычет числа 3125 в кольце по модулю 7.

$$3125(\bmod 7)=\left(\left[\frac{3125}{7}\right]\cdot 7+3\right)\bmod 7=(446\cdot 7+3)\bmod 7\equiv 3.$$

В данном примере результатами определения вычета могут быть только числа $\{0,...,6\}$.

Пример 1.2.2. Рассмотрим пример перевода отрицательного числа в положительный вычет (используем результат примера 1.2.1):

$$-3125(\bmod 7)=-(3125(\bmod 7))=-3(\bmod 7)\equiv 7-3=4.$$

1.3. «Метод цепочек» в модулярной арифметике

Определение вычетов по $(\bmod m)$ может быть проведено с использованием метода цепочек модулярной арифметики:

$$(a\pm b)\bmod m=((a\bmod m)\pm(b\bmod m))\bmod m;$$

$$(a*b)\bmod m=((a\bmod m)*(b\bmod m))\bmod m;$$

$$(a*(b\pm c))\bmod m=(((a*b)\bmod m)\pm((a*c)\bmod m))\bmod m.$$

Пример 1.3.1. Пусть требуется представить в форме цепочки $a^8 (\mathrm{mod}\, m)$. Степень четная, более того $8 = 2^3$.

Цепочка преобразований

$$(((a^2 (\mathrm{mod}\, m)^2 \bmod m)^2 \bmod m.$$

Пример 1.3.2. Найти методом цепочек $a^n (\mathrm{mod}\, m) = 15^{63} (\mathrm{mod}\, 7)$.

Решение: $a = 15 = 3 \cdot 5;\ n = 63 = 3^2 \cdot 7.$

$15^{63} (\mathrm{mod}\, 7) = ((3^{63} \bmod 7) \cdot (5^{63} \bmod 7)) \bmod 7 = (A \cdot B) \bmod 7;$

$A = 3^{63} \bmod 7 = 3^{3 \cdot 3 \cdot 7} \bmod 7 = ((3^3 \bmod 7)^3 \bmod 7)^7 \bmod 7;$

$3^3 \bmod 7 = 27 \bmod 7 = (3 \cdot 7 + 6) \bmod 7 = 6;$

$6^3 \bmod 7 = 216 \bmod 7 = (30 \cdot 7 + 6) \bmod 7 = 6;$

$6^7 \bmod 7 = 6^{2 \cdot 2 + 3} \bmod 7 = ((6^{2 \cdot 2} \bmod 7) \cdot (6^3 \bmod 7)) \bmod 7;$

$$6^{2 \cdot 2} \bmod 7 = (6^2 \bmod 7)^2 \bmod 7;$$
$$6^2 \bmod 7 = 36 \bmod 7 = (5 \cdot 7 + 1) \bmod 7 = 1;$$

$1^2 \bmod 7 = 1;\ 6^7 \bmod 7 = (1 \cdot 6) \bmod 7 = 6;\ \ A = 6.$

Аналогичными действиями для $5^{63} \bmod 7$ получаем $B = 6.$

$15^{63} (\mathrm{mod}\, 7) = A \cdot B \bmod 7 = (6 \cdot 6) \bmod 7 = 36 \bmod 7 = (5 \cdot 7 + 1) \bmod 7 = 1.$

1.4. Функция Эйлера и теорема Эйлера

Функция Эйлера $\varphi(n)$ – арифметическая функция, равная количеству натуральных чисел, меньших числа *n* и взаимно простых с ним. При этом полагают по определению, что число 1 взаимно просто со всеми натуральными числами.

Функция Эйлера:

$$\varphi(n) = \left|\left\{0 \le b < n \middle| \text{НОД}(b,n) = 1\right\}\right|$$

обладает следующими ключевыми ***свойствами***:

1) $\varphi(1) = 1;$

2) для любого простого p: $\varphi(p) = p - 1;$

3) для любого простого p в степени α: $\varphi\left(p^{\alpha}\right) = p^{\alpha} - p^{\alpha-1};$

4) $\forall m, n \in N \mid \text{НОД}(m,n) = 1 : \varphi(m \cdot n) = \varphi(m) \cdot \varphi(n).$

Пример 1.4.1. Найти функцию Эйлера от простого числа

$$\varphi(31) = 31 - 1 = 30.$$

Пример 1.4.2. Найти функцию Эйлера от составного числа

$$\varphi(6000) = \varphi\left(5^3 \cdot 2^4 \cdot 3\right) = \varphi\left(5^3\right) \cdot \varphi\left(2^4\right) \cdot \varphi(3) = 100 \cdot 8 \cdot 2 = 1600.$$

Для решения поставленной задачи сперва проводим факторизацию числа (разложение на сомножители из простых чисел в соответствующих степенях):

$$6000 = 5^3 \cdot 2^4 \cdot 3.$$

Вычисляем функции Эйлера от сомножителей числа:

$$\varphi\left(5^3\right) = \left(5^3 - 5^2\right) = 125 - 25 = 100;$$

$$\varphi\left(2^4\right) = \varphi\left(2^4\right) = \left(2^4 - 2^3\right) = 16 - 8 = 8;$$

$$\varphi(3) = (3 - 1) = 2.$$

Затем перемножаем полученные функции Эйлера сомножителей.

Теорема Эйлера:

Для любых двух взаимнопростых чисел натурального ряда

$$\varphi(n)=\left|\left\{1\le b<n\,\middle|\,\text{НОД}(b,n)=1\right\}\right|$$

выполняется

$$a^{\varphi(m)}\equiv 1(\mathrm{mod}\,m). \tag{1.1}$$

Следствие из теоремы Эйлера:

Если числа $\text{НОД}(a,m)=1$ и n' – наименьший неотрицательный вычет n по модулю $\varphi(m)$, т.е.

$$n'=n\left(\mathrm{mod}\,\varphi(m)\right), \tag{1.2}$$

то

$$a^{\varphi(m)-1}\equiv a^{-1}(\mathrm{mod}\,m). \tag{1.3}$$

Комментарий. В данном пособии теоремы и следствия из них приводятся в своих формулировках для их дальнейшего использования в практических целях. Доказательства подробно рассмотрены в учебных пособиях [3, 4].

Пример 1.4.3. Возведение в степень по модулю с использованием функции Эйлера. Пусть требуется вычислить

$$5^{17}\,\mathrm{mod}\,7\,.$$

Используем формулы (1.2) и (1.3) из следствия теоремы Эйлера.

Решение.

$a=5; n=17; m=7$. При этом 7 – простое число.

Находим

$\varphi(m)=\varphi(7)=7-1=6;$

$n'=17\,\mathrm{mod}\,6=(2\cdot 6+5)\,\mathrm{mod}\,6=5;$

$5^{17}\,(\mathrm{mod}\,7)=5^{5}\,(\mathrm{mod}\,7)=3125(\mathrm{mod}\,7)=(446\cdot 7+3)\,\mathrm{mod}\,7=3.$

1.5. Обратное число по модулю

Определение. Число является обратным к числу a по модулю m (в кольце по модулю m), если выполнено условие

$$a \cdot a^{-1} (\mathrm{mod}\, m) \equiv 1. \qquad (1.4)$$

Теорема. Для каждого целого a существует единственное целое число b, такое что произведение ab равно 1 в кольце по модулю m, тогда и только тогда, когда a и m – взаимно просты:

$$\forall a \in Z : \exists! b \in Z : a \cdot b \equiv 1 (\mathrm{mod}\, m) \Leftrightarrow \text{НОД}(a, m) = 1.$$

При этом

$$b \equiv a^{-1} (\mathrm{mod}\, m).$$

Пример 1.5.1. Найти «обратное по модулю» число «по определению». Найти

$$x = 9^{-1} \left(\mathrm{mod}\, 5\right).$$

Решение. Учитываем правило модулярной арифметики «отбрасывания полностью пройденного кольца» и возвращаемся к правилам классической арифметики (как при работе с любыми периодическими функциями):

$$9x = 1 \left(\mathrm{mod}\, 5\right);$$

$$9x = 1 + km,$$

где k – целые числа. Ограничимся числами k как числами натурального ряда.

Получаем уравнение:

$$9 \cdot x - k \cdot 5 = 1;$$

$$x = \left(1 + k \cdot 5\right) : 9.$$

Производим последовательный перебор возможных кратностей k (количества раз полностью пройденного кольца m) как чисел последовательности натуральных чисел $\{1, 2, 3, 4, ...\}$ до тех пор, пока не

получим x как *целое* число. В данном примере при $k=7$ получено целое число $x=4$.

Итак, получено:

$9^{-1}\left(\mathrm{mod}\,5\right)=4.$

Проверка:

$9\cdot 4-7\cdot 5=36-35=1.$

Обратное число в кольце по модулю может быть найдено также с использованием формулы с функцией Эйлера, полученной из формулы (1.1) теоремы Эйлера:

$$a^{\varphi(m)}\equiv 1(\mathrm{mod}\,m).$$

Домножим левую и правую части этого уравнения на

$$a^{-1}(\mathrm{mod}\,m).$$

Получаем правило нахождения числа, обратного по модулю (в кольце), *с использованием функции Эйлера:*

$$a^{\varphi(m)-1}\equiv a^{-1}(\mathrm{mod}\,m). \qquad (1.5)$$

Пример 1.5.2. Найти «обратное по модулю» число с использованием функции Эйлера. Рассмотрим то же задание, что и в примере 1.5.1. Найти:

$$x=9^{-1}\left(\mathrm{mod}\,5\right).$$

Решение: $a=9, m=5.$ $\mathrm{НОД}\left(9,5\right)=1$, $m=5$ – простое число.

Находим $\varphi\left(m\right)=\varphi\left(5\right)=5-1=4;$

$x=\left(9^{4-1}\right)(\mathrm{mod}\,5)=729\left(\mathrm{mod}\,5\right)=\left(145\cdot 5+4\right)\mathrm{mod}\,5=4$, т.е. $b=x=4.$

Получен тот же результат без перебора неподходящих вариантов k. Если в дальнейшем потребуется k как коэффициент диофантова уравнения, его можно будет определить из уравнения $a\cdot b-k\cdot m=1.$

1.6. Расширенный алгоритм Евклида

Для определения обратных чисел в кольце по модулю при программной реализации в прикладной криптографии используются не аналитические преобразования п. 1.5, а специальный алгоритм, называемый «расширенный алгоритм Евклида», построенный на основе классического алгоритма Евклида (п. 1.1) и теоретических основ модулярной арифметики п. 1.5, позволяющий наряду с наибольшим общим делителем чисел A и B находить последовательность целых чисел x и y, удовлетворяющие равенству

$$Ax + By = \text{НОД}\left(A, B\right),$$

называемому *диофантово* уравнение.

Для взаимнопростых чисел это уравнение принимает вид:

$$Ax + By = 1, \tag{1.6}$$

полностью соответствующий правилам построения арифметического правила вычисления обратного числа по модулю в переводе к правилам классической, а не модулярной арифметики, что было рассмотрено на примере 1.5.1.

От обычного алгоритма Евклида (п. 1.1) расширенный алгоритм отличается тем, что наряду с последовательностью остатков r_i вычисляются ещё две вспомогательные последовательности x_i и y_i.

Расширенный алгоритм Евклида заключается в изложенных далее действиях.

На первом шаге положим, что

$r_{-1} = A,\ x_{-1} = 1,\ y_{-1} = 0;$

$r_0 = B,\ x_0 = 0,\ y_0 = 1.$

Далее будем выполнять следующую последовательность действий, до тех пор пока $r_i > 0$:

$d_i = [r_{i-2} / r_{i-1}];\ r_i = r_{i-2} - d_i r_{i-1};\ x_i = x_{i-2} - d_i x_{i-1};\ y_i = y_{i-2} - d_i y_{i-1};$
$i = i + 1.$

Значения x_k и y_k, при которых

$$r_k = \text{НОД}(A, B),$$

и будут искомыми в силу следующего *утверждения*.

Утверждение. При всех i, где $-1 \le i \le k$, выполнимо равенство:

$$x_i A + y_i B = r_i.$$

Внимание! При решении $Ax + By = 1$ (для взаимнопростых чисел) достаточно выполнить алгоритм до получения $r_k = 1$. При этом x_k будет «обратным по модулю *B* для числа *A*»:

$$x \equiv A^{-1} (\text{mod}\, B).$$

Пример 1.6.1. Найдем $x = 9^{-1} (\text{mod}\, 5)$ с использованием расширенного алгоритма Евклида, Важно: $\text{НОД}(9,5) = 1$, т.е. решаем уравнение $Ax + By = 1.$

Реализация алгоритма приведена в табл. 1.2.

Таблица 1.2 – Реализация алгоритма для $x = 9^{-1} (\text{mod}\, 5)$

Шаг		Начальные условия	Остаток
$j = -1$		$r_{-1} = A = 9;$ $x_{-1} = 1,$ $y_{-1} = 0$	$r_{-1} = A = 9;$
$j = 0$		$r_0 = B = 5;$ $x_0 = 0,$ $y_0 = 1$	$r_0 = B = 5;$

Продолжение табл. 1.2

Шаг	Кратность	Разложение	Остаток
$j=1$	$d_1=\left[\frac{r_{-1}}{r_0}\right]=\left[\frac{A}{B}\right]=\left[\frac{9}{5}\right]=1$	$r_1=r_{-1}-d_1\cdot r_0=9-1\cdot 5=4;$ $x_1=x_{-1}-d_1x_0=1-1\cdot 0=1;$ $y_1=y_{-1}-d_1y_0=0-1\cdot 1=-1;$	$r_1=4$
$j=2$	$d_2=\left[\frac{r_0}{r_1}\right]=\left[\frac{5}{4}\right]=1$	$r_2=r_0-d_2\cdot r_1=5-1\cdot 4=1;$ $x_2=x_0-d_2\cdot x_1=0-1\cdot 1=-1;$ $y_2=y_0-d_2\cdot y_1=1-1\cdot(-1)=2;$	$r_2=1$ Решение $x_2=-1$

Проверка 1: По диофантову уравнению получаем:

$Ax+By=1:\ 9\cdot(-1)+5\cdot 2=1$ – верно.

Следует особо отметить, что при численной реализации расширенного алгоритма Евклида в рассмотренном примере искомый результат получен как отрицательное число

$$x_2<0.$$

Для его дальнейшего применения в алгоритмах криптографии потребуется привести его к представлению *положительного* вычета:

$$9^{-1}\,(\mathrm{mod}\,5)=x_2\,(\mathrm{mod}\,5)=x_2+5=-1+5=4.$$

Для *положительного* вычета помимо проверки результата при помощи диофантова уравнения может быть проведена проверка обратного числа по определению (1.4).

Проверка 2:

$$A\cdot A^{-1}\bmod(B)=1:\ (9\cdot 4)(\mathrm{mod}\,5)=36(\mathrm{mod}\,5)=(7\cdot 5+1)(\mathrm{mod}\,5)=1.$$

Верно.

Комментарий о форме записи. Допустима запись

$$(9\cdot 4)=36=(7\cdot 5+1)\equiv 1(\mathrm{mod}\,5),$$

но при записи длинных вычислений «вручную» проще сохранять запись модуля, его основания и обычное равенство в промежуточных шагах, чтобы не возникло разночтений.

При использовании расширенного алгоритма Евклида в прикладной криптографии последовательность чисел начального приближения определяется самой задачей, а не пользователем. Это означает, что возможна как ситуация $A > B$, так и $A < B$. Рассмотрим пример, соответствующий случаю $A < B$.

Пример 1.6.2. Найдем $x = 5^{-1} (\mathrm{mod} 9)$ с использованием расширенного алгоритма Евклида, Важно: $\mathrm{НОД}(9,5) = 1$, т.е. решаем уравнение $Ax + By = 1.$

Реализация алгоритма приведена в табл. 1.3.

Таблица 1.3 – Реализация алгоритма для $x = 5^{-1} (\mathrm{mod} 9)$

Шаг		Начальные условия	Остаток
$j = -1$		$r_{-1} = A = 5;$ $x_{-1} = 1,\ y_{-1} = 0$	$r_{-1} = A = 5;$
$j = 0$		$r_0 = B = 9;$ $x_0 = 0,\ y_0 = 1$	$r_0 = B = 9;$
Шаг	Кратность	Разложение	Остаток
$j = 1$	$d_1 = \left[\frac{r_{-1}}{r_0}\right] = \left[\frac{A}{B}\right] = \left[\frac{5}{9}\right] = 0$	$r_1 = r_{-1} - d_1 \cdot r_0 = 5 - 0 \cdot 9 = 5;$ $x_1 = x_{-1} - d_1 x_0 = 1 - 0 \cdot 0 = 1;$ $y_1 = y_{-1} - d_1 y_0 = 0 - 0 \cdot 1 = 0;$	$r_1 = 5$
$j = 2$	$d_2 = \left[\frac{r_0}{r_1}\right] = \left[\frac{9}{5}\right] = 1$	$r_2 = r_0 - d_2 \cdot r_1 = 9 - 1 \cdot 5 = 4;$ $x_2 = x_0 - d_2 x_1 = 0 - 1 \cdot 1 = -1;$ $y_2 = y_0 - d_2 y_1 = 1 - 1 \cdot 0 = 1;$	$r_2 = 4$

Продолжение табл. 1.3

Шаг	Кратность	Разложение	Остаток
$j=3$	$d_3=\left[\frac{r_1}{r_2}\right]=\left[\frac{5}{4}\right]=1$	$r_3=r_1-d_3\cdot r_2=5-1\cdot 4=1;$ $x_3=x_1-d_3\cdot x_2=1-1\cdot(-1)=2;$ $y_3=y_1-d_3\cdot y_2=0-1\cdot 1=-1;$	$r_3=1$ Решение: $x_3=2$

Проверка 1:

$Ax+By=1:\ 5\cdot 2+9\cdot(-1)=1.$ Верно.

В данном случае получили

$x_3=2>0.$

Это и есть нужный нам положительный вычет.

Проверка 2:

$A\cdot A^{-1}\bmod(B)=1:\ (5\cdot 2)\bmod 9=10(\bmod 9)=(1\cdot 9+1)(\bmod 9)=1.$

Верно.

1.7. Примеры индивидуальных заданий для обращения числа

Примеры вариантов индивидуальных заданий для решения задачи обращения числа в кольце по модулю приведены в табл. 1.4.

Таблица 1.4 – Варианты заданий для обращения числа по модулю

№	Задание	№	Задание	№	Задание
1	$7^{-1}(\bmod 9)$	7	$7^{-1}(\bmod 17)$	13	$9^{-1}(\bmod 23)$
2	$19^{-1}(\bmod 5)$	8	$5^{-1}(\bmod 9)$	14	$7^{-1}(\bmod 23)$
3	$17^{-1}(\bmod 5)$	9	$19^{-1}(\bmod 7)$	15	$7^{-1}(\bmod 13)$
4	$5^{-1}(\bmod 27)$	10	$17^{-1}(\bmod 9)$	16	$7^{-1}(\bmod 5)$
5	$5^{-1}(\bmod 14)$	11	$17^{-1}(\bmod 7)$	17	$5^{-1}(\bmod 7)$

Продолжение табл. 1.4

№	Задание	№	Задание	№	Задание
6	$79^{-1}(\mathrm{mod}17)$	12	$17^{-1}(\mathrm{mod}79)$	18	$7^{-1}(\mathrm{mod}11)$
19	$19^{-1}(\mathrm{mod}17)$	23	$19^{-1}(\mathrm{mod}11)$	27	$19^{-1}(\mathrm{mod}23)$
20	$7^{-1}(\mathrm{mod}31)$	24	$17^{-1}(\mathrm{mod}31)$	28	$17^{-1}(\mathrm{mod}23)$
21	$9^{-1}(\mathrm{mod}11)$	25	$23^{-1}(\mathrm{mod}11)$	29	$31^{-1}(\mathrm{mod}11)$
22	$31^{-1}(\mathrm{mod}7)$	26	$31^{-1}(\mathrm{mod}5)$	30	$31^{-1}(\mathrm{mod}9)$

1.8. Метод повторного возведения в квадрат

В прикладной криптографии (при использовании двоичного кода представления целых чисел) особое значение имеет возможность построения на основе метода цепочек (п. 1.3) алгоритма повторного возведения в квадрат, который постоянно используется при программной реализации криптографических систем.

Обозначим промежуточный результат вычисления a. В конце работы алгоритма a примет значение наименьшего неотрицательного вычета $b^n(\mathrm{mod}\,m)$. Пусть

$$n = n_0 \cdot 2^0 + n_1 \cdot 2^1 + n_2 \cdot 2^2 + \ldots + n_{k-1} \cdot 2^{k-1},$$

где n_j при $j = 0, 1, 2, \ldots, k-1$ – цифры двоичной записи числа n. Каждое n_j равно либо 1, либо 0. Принимаем начальное значение $a = 1$. Первые шаги алгоритма метода повторного возведения в квадрат представлены в табл. 1.5.

Таблица 1.5 – Первые шаги алгоритма повторного возведения в квадрат

j	b_j	a
$j=0$	$b_0 = b(\mathrm{mod}\, m)$	при $n_0 = 1 \Rightarrow a = b_0$; при $n_0 = 0 \Rightarrow a = 1$
$j=1$	$b_1 = \left(b^2\right) \mathrm{mod}\, m$	при $n_1 = 1 \Rightarrow a = \left(a \cdot b_1\right) \mathrm{mod}\, m$
$j=2$	$b_2 = \left(b_1^2\right) \mathrm{mod}\, m$	при $n_2 = 1 \Rightarrow a = \left(a \cdot b_2\right) \mathrm{mod}\, m$
$j=3$	$b_3 = \left(b_2^2\right) \mathrm{mod}\, m$	при $n_3 = 1 \Rightarrow a = \left(a \cdot b_3\right) \mathrm{mod}\, m$

Алгоритм продолжается для всех $j = 0, 1, 2, ..., k-1$.

При $n_j = 0$ достигнутое значение a не меняется. На j-ом шаге, получим $b_j = b^{2^j} (\mathrm{mod}\, m)$. Если $n_j = 1$, то есть – когда 2^j входит в двоичное представление числа n, используем b_j как множитель для вычисления нового значения a и не делаем этого при $n_j = 0$. После выполнения шагов по всем j, получим искомое $a = b^n (\mathrm{mod}\, m)$.

Пример 1.8.1. Найти $5^{17} \mathrm{mod}\, 7$ методом повторного возведения в квадрат.

Получение бинарного («двоичного») представления числа 17 показано в табл. 1.6. Строки таблицы заполняются справа – налево.

Таблица 1.6 – Построение двоичного представления числа 17

Четное число без остатка					16
Делим на 2	1	2	4	8	17
Остаток (бинарное представление числа)	1	0	0	0	1
Позиция (разряд)	4	3	2	1	0

Проверка: $n = 1 \cdot 2^0 + 1 \cdot 2^4 = 1 + 16 = 17.$

Процедура «повторного возведения в квадрат» для $5^{17} \bmod 7$ представлена в табл. 1.7.

Таблица 1.7 – «Повторное возведение в квадрат» для $5^{17} \bmod 7$

j	n_j	b_j	a_j
0	1	$b_0 = 5(\bmod 7) = 5$	$a = b_0 = 5$
1	0	$b_1 = \left(5^2\right) \bmod 7 = 25(\bmod 7) = $ $= (3 \cdot 7 + 4) \bmod 7 = 4$	$a = a = 5$
2	0	$b_2 = \left(4^2\right) \bmod 7 = 16(\bmod 7) = $ $= (2 \cdot 7 + 2) \bmod 7 = 16(\bmod 7) = 2$	$a = a = 5$
3	0	$b_3 = \left(2^2\right) \bmod 7 = 4(\bmod 7) = 4$	$a = a = 5$
4	1	$b_4 = \left(4^2\right) \bmod 7 = 16(\bmod 7) = $ $= (2 \cdot 7 + 2) \bmod 7 = 2$	$a = (a \cdot b_4) \bmod m = $ $(5 \cdot 2) \bmod 7 = $ $= 10(\bmod 7) = (7 + 3) \bmod 7 = 3$

Ответ: $5^{17}(\bmod 7) = 3$. Результат совпадает с ответом примера 1.4.3, полученным аналитическим методом для той же постановки задачи.

1.9. Примеры индивидуальных заданий для возведения числа в степень

Примеры вариантов индивидуальных заданий для решения задачи возведения числа в степень в кольце по модулю методом повторного возведения в квадрат приведены в табл. 1.8.

Таблица 1.8 – Варианты заданий для повторного возведения в квадрат

№	Задание	№	Задание	№	Задание
1	$17^{31} \bmod 7$	7	$17^{121} \bmod 7$	13	$3^{213} \bmod 9$

Продолжение табл. 1.8

№	Задание	№	Задание	№	Задание
2	$3^{311} \bmod 7$	8	$3^{121} \bmod 7$	14	$17^{21} \bmod 9$
3	$17^{213} \bmod 7$	9	$2^{213} \bmod 5$	15	$5^{63} \bmod 7$
4	$2^{61} \bmod 5$	10	$3^{255} \bmod 7$	16	$5^{21} \bmod 7$
5	$2^{111} \bmod 5$	11	$5^{213} \bmod 7$	17	$5^{213} \bmod 7$
6	$3^{61} \bmod 7$	12	$2^{63} \bmod 7$	18	$7^{17} \bmod 5$
19	$19^{31} \bmod 7$	23	$19^{121} \bmod 7$	27	$19^{213} \bmod 9$
20	$9^{311} \bmod 7$	24	$9^{121} \bmod 7$	28	$23^{21} \bmod 9$
21	$7^{213} \bmod 5$	25	$2^{213} \bmod 7$	29	$17^{63} \bmod 5$
22	$2^{61} \bmod 11$	26	$3^{255} \bmod 11$	30	$5^{21} \bmod 11$

1.10. Комментарий по переводу множества символов алфавита в цифровое представление

При представлении сочетания символов («слова») единым числом используется позиционная система нумерации символов алфавита. Большие сообщения шифруются и передаются целевому абоненту партиями по несколько символов, например по 3, по 6 символов. При этом каждая партия переводится в цифровое представление, т.е. представляется единым числом. Провокация состоит в том, что при использовании основания кольца 26 (как в примерах шифрования «вручную» классическими шифрами), т.е. при нумерации символов латинского алфавита от 0 до 25, невозможно зашифровать единым числом сочетание символов, начинающееся с буквы А (т.к. ее номер будет 0). Если символ с номером 0 стоит не первым, эта проблема не возникает.

Для шифрования сообщений единым словом требуется принять основание кольца больше, чем количество букв как минимум на 1, т.е. для латинского алфавита таким основанием будет число, не меньшее 27. Сами буквы алфавита при этом нумеруются от 1 до 26. В качестве символа с номером 0 может быть добавлен, например, символ «пробел», или другой специальный символ, с которого слова сообщений не начинаются.

Далее приведены примеры перевода множества символов («слова») в единое число по основанию 27. Требуемые соответствия символов, их номеров в алфавите и позиций в «слове» представлены в табл. 1.9 – 1.13.

Пример 1.9.1.

Пусть нужно перевести в число «слово» «*GSK*».

Таблица 1.9 – Соответствие символов и их номеров в алфавите (кольце)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
_	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*

Таблица 1.10 – Соответствие символов и их числовых значений

Символ в алфавите	*G*	*S*	*K*
Номер в алфавите	7	19	11
Разряд в «слове»	2	1	0

Перевод «слова» *GSK* в единое число:

$$7 \cdot 27^2 + 19 \cdot 27 + 11 = 5627.$$

Обратный перевод из числа во множество символов осуществляется следующим образом. Делим 5627 на 27, результат опять на 27 и т.д. пока не получим число, меньшее 27. в данном случае разделить получается 2 раза, значит 5627 делится на 27^2 (в квадрате). Одновременно определили старший разряд и количество символов (2+1=3).

Находим остаток от деления на 27^2 нацело:

$$5627-\left[\frac{5627}{27^2}\right]\cdot 27^2 = 5627-7\cdot 27^2 = 5627-7\cdot 729 = 5627-5103 = 524.$$

Число 524 пытаемся разделить на 27 (на 27 в степени следующего младшего разряда). При делении нацело получаем 19. Находим остаток:

$$524-\left[\frac{524}{27}\right]\cdot 27 = 524-19\cdot 27 = 524-513 = 11.$$

Это число всегда меньше основания 27 и равно значению в самом младшем разряде единого числа.

Проверка:

$$7\cdot 27^2+19\cdot 27+11 = 5103+513+11 = 5627.$$

Таблица 1.11 – Сопоставление результатов разложения с символами

Разряд в «слове»	2	1	0
Значение в разряде, равно номеру символа в алфавите	1	7	11
Символ в алфавите	*G*	*S*	*K*

Пример 1.9.2. Пусть нужно перевести в число «слово» с пробелом *AG_Z*

Таблица 1.12 – Соответствие символов и их числовых значений

Символ в алфавите	*A*	*G*	_	*Z*
Номер символа в алфавите	1	7	0	26
Разряд в «слове»	3	2	1	0

Перевод *AG_Z* в единое число:

$$1\cdot 27^3+7\cdot 27^2+0\cdot 27+26 = 24812.$$

Осуществляем обратный перевод из числа во множество символов. Делим 24812 на 27, результат опят на 27 и т.д. пока не получим число, меньшее 27. в данном случае разделить получается 3 раза, значит 24812

делится на 27^3 (в кубе). Одновременно определили старший разряд и количество символов (3+1=4). Находим остаток от деления на 27^3 нацело:

$$24812-\left[\frac{24812}{27^3}\right]\cdot 27^3 = 24812-1\cdot 27^3 = 24812-7\cdot 729 =$$

$$= 24812-19683 = 5129.$$

Число 5129 пытаемся разделить на 27^2 (на 27 в степени следующего младшего разряда). Делится и при делении нацело получаем 7. Находим остаток:

$$5129-\left[\frac{5129}{27^2}\right]\cdot 27^2 = 5129-7\cdot 27^2 = 5129-7\cdot 729 = 5129-5103 = 26.$$

Число 26 пытаемся разделить на 27 (на 27 в степени следующего младшего разряда). Число 26 меньше 27. Не делится. Это значит, что в данной позиции получаем 0. Это допустимо и вполне нормально.

Само число 26 меньше основания 27 и соответствует последнему остатку, т.е. самому младшему разряду.

Проверка:

$$1\cdot 27^3+7\cdot 27^2+0\cdot 27+26 = 19683+5103+0+26 = 24812.$$

Таблица 1.13 – Соответствие символов и их числовых значений

Разряд в «слове»	3	2	1	0
Значение в разряде =Номер символа в алфавите	1	7	0	26
Символ в алфавите	*A*	*G*	_	*Z*

Внимание! Основания «*p*» для mod (*p*) в криптографических системах для практического применения шифров строго ограничивают возможные для передачи числовые представления исходных («символьных») сообщений. Можно передать только число, меньшее чем «*p*» для mod (*p*) при шифровании и расшифровки, иначе оно однозначно не расшифруется.

При шифровании биграммами (по два символа) самое большое число получим для «*ZZ*». Вычисляем его:

$$26 \cdot 27 + 26 = 728.$$

Следовательно, для шифрования и расшифровки символов латинского алфавита биграммами нужно, чтобы «*p*» в mod (*p*) было строго больше числа 728.

При шифровании триграммами (по три символа) самое большое число получим для «*ZZZ*». Вычисляем его:

$$26 \cdot 27^2 + 26 \cdot 27 + 26 = 19682.$$

Для шифрования триграммами нужно выбирать $p > 19682$.

И так далее – чем больше символов хотим передать в одном сообщении, тем больше нужно брать «*p*».

В «алфавит» для исходных сообщений могут быть включены также и цифры, и знаки препинания, и спецсимволы. Все это требуется учитывать при выборе «оснований кольца для нумерации символов», а потом – при выборе оснований mod (*p*) и т.д. для шифрования и расшифровки.

2. АЛГОРИТМИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ

2.1. Криптографические протоколы

Для успешного выполнения любых целей по защите информации необходимо участие в процессе защиты нескольких субъектов, которые по определённым правилам будут выполнять технические или организационные действия, криптографические операции, взаимодействовать друг с другом, например, передавая сообщения или проверяя личности друг друга.

Формализация подобных действий делается через описание *протокола*.

Протокол **–** описание распределённого алгоритма, в процессе выполнения которого два или более участников последовательно выполняют определённые действия и обмениваются сообщениями.

Следует учитывать, что в прикладной криптографии все практически значимые системы являются *асимметричными*. *Криптосистемы с открытым ключом* (асимметричные криптосистемы) – это алгоритмы, в которых ключи шифрования и дешифрования различны. Более того, ключ дешифрования не может быть вычислен из ключа шифрования. Такие алгоритмы называют криптосистемами с открытым ключом, так как любой абонент, зная открытый ключ, может зашифровать сообщение, однако расшифровать шифртекст сможет только абонент, владеющий секретным ключом.

Открытый ключ – тот из двух ключей асимметричной системы, который распространяется свободно. Этот ключ, как правило, является шифрующим для секретной переписки и расшифровывающим – для электронной подписи (ЭЦП).

Секретный ключ (закрытый ключ) – тот из двух ключей асимметричной криптосистемы, который хранится в секрете.

При описании протокола криптографической системы важно учитывать, что в ней выступает в качестве открытых (пересылаемых в открытом доступе) ключей и кто из абонентов *инициирует* процесс передачи данных.

В дальнейшем будем обозначать абонентов следующим образом: абонент *A* имеет исходную открытую информацию, должен ее зашифровать и отправить в зашифрованном виде абоненту *B*, который хочет эту информацию получить и иметь возможность расшифровать полученное сообщение.

При этом в различных криптографических системах (в зависимости от применяемого протокола передачи данных) процесс первоначального определения ключей для шифрования и расшифровки сообщений, передачи открытых ключей, самого зашифрованного сообщения и/или подтверждающей авторство этого сообщения дополнительной информации в виде электронной цифровой подписи (ЭЦП) может являться различным.

Важно также учитывать, для какого количества абонентов предназначен протокол криптосистемы. В ряде базовых криптосистем предусмотрен обмен данными непосредственно между двумя абонентами, но также возможны протоколы с ЭЦП, не ограничивающие столь жестко количество адресатов передаваемого сообщения. При этом существенно предусмотреть защиту передаваемой информации при помощи многоэтапной передачи данных несколькими партиями и взаимно согласованное соединение частей той же ЭЦП из различных частей полученных сообщений.

2.2. Криптографический протокол Шамира

В качестве простейшего, но весьма надежного базового протокола передачи данных вообще без пересылки ключей можно рассмотреть протокол Шамира.

Трехэтапный (трехпроходный) протокол Шамира предназначен для защищенной (зашифрованной) передачи сообщения от абонента *A* абоненту *B* без обмена ими ключами шифрования и расшифровки.

Алгоритм протокола Шамира:

1) абонент *A* зашифровывает открытое сообщение своим ключом и передает это зашифрованное сообщение абоненту *B*;

2) абонент *B* зашифровывает полученное сообщение своим ключом и передает это (дважды зашифрованное) сообщение абоненту *A*;

3) абонент *A* расшифровывает сообщение от абонента *B* своим ключом, тем самым снимая свое шифрование, и возвращает зашифрованное только собственным ключом абонента *B* сообщение абоненту *B*;

4) абонент *B* расшифровывает сообщение своим ключом и тем самым восстанавливает исходное открытое сообщение.

Этот протокол зачастую называют «ящиком с двойным замком».

На практике для передачи больших шифруемых сообщений протокол Шамира использовать крайне нерационально. Тем не менее, он может быть использован для обеспечения дополнительной степени защиты при пересылке самих открытых ключей шифрования для других криптографических систем.

2.3. Этапы криптографической схемы «Протокол Шамира»

Краткое изложение основ и правил применения криптографической системы «Протокол Шамира» рассмотрим на конкретном примере.

Пример 2.3.1. Дано: $p = 38177,\ m = "XYZ"$.

Этап 1 – генерация ключей.

Абонент *A*.

Выберем значение $e_A = 51407$ по условию НОД$(e_A, p-1) = 1$.

Находим значение d_A:

$$d_A = e_A^{-1} (\mathrm{mod}\, p-1).$$

Абонент *B*.

Выберем значение $e_B = 42239$ по условию НОД$(e_B, p-1) = 1$.

Находим значение d_B:

$$d_B = e_B^{-1} (\mathrm{mod}\, p-1).$$

Закрытый ключ абонента A: $e_A = 51407$.

Закрытый ключ абонента B: $e_B = 42239$.

Этап 2 – преобразование сообщения в числовой эквивалент.

Преобразуем передаваемую триграмму «*XYZ*» в числовой эквивалент для последующей обработки

$$m = "XYZ" = 23 \cdot 26^2 + 24 \cdot 26^1 + 25 \cdot 26^0 = 16197.$$

Этап 3 – трехпроходный алгоритм протокола Шамира.

Используя обозначения, применяемые в протоколе Шамира, введем значения α и β: $\alpha = e_A;\ \beta = e_B$.

Выполним 1-й шаг алгоритма Шамира:

$$C_1 = E_\alpha(m) = m^{e_A} (\mathrm{mod}\, p).$$

Выполним 2-й шаг алгоритма Шамира:

$$C_2 = E_\beta(C_1) = C_1^{e_B} (\mathrm{mod}\, p).$$

Выполним 3-й шаг алгоритма Шамира:

$$C_3 = D_\alpha(C_2) = C_2^{d_A} (\mathrm{mod}\, p).$$

Вычислим передаваемое абоненту *B* значение из выражения:

$$m = D_{\beta}\left(C_3\right) = D_{\beta}\left(E_{\beta}\left(m\right)\right);$$

$$m = D_{\beta}(C_3) = C_3^{d_B} (\mathrm{mod}\, p).$$

Этап 4 – преобразование полученного значения к формату текста:

$$m = 216197 = 3 \cdot 26^2 + 24 \cdot 26^1 + 25 \cdot 26^0 = "XYZ".$$

Полученное значение: "*XYZ*".

2.4. Криптографический протокол Шамира на примере с аналитическим выполнением действий

Подробное выполнение действий криптографической системы «Протокол Шамира» рассмотрим на упрощенном примере, допускающем наглядное выполнение расчетов аналитически («вручную»).

Пример 2.4.1. Пусть задано: $p = 23$, Требуется зашифровать и передать сообщение $m = "J"$.

Определяем: $p - 1 = 22.$

Этап 1 – производится генерация ключей у обоих абонентов.

Абонент *A*: выберем значение $e_A = 5$ по условию НОД$(e_A, p-1) = 1$.

Находим значение d_A:

$$d_A = e_A^{-1} \bmod (p-1) = 5^{-1} (\mathrm{mod}\, 22) = 9.$$

Для расчета используем расширенный алгоритм Евклида (все расчетные действия представлены ниже, в разделе «Расчеты»).

Абонент *B*: выберем значение $e_B = 7$ по условию НОД$(e_B, p-1) = 1$.

Находим значение d_B:

$$d_B = e_B^{-1} \bmod (p-1) = 7^{-1} (\mathrm{mod}\, 22) = 19.$$

Для расчета используем расширенный алгоритм Евклида.

Закрытые ключи абонента A:

$e_A = 5$ – для шифрования, $d_A = 9$ – для расшифровки.

Закрытые ключи абонента B:

$e_B = 7$ – для шифрования, $d_B = 19$ – для расшифровки.

Этап 2 – преобразование сообщения в числовой эквивалент.

Преобразуем передаваемое сообщение “*J*” в числовой эквивалент для последующей обработки.

В буквенном представлении $m = "J"$. В числовом представлении $m = 10$.

Важно! При выбранном основании модуля $p = 23$ числовое значение большее 22 однозначно зашифровать не сможем, как показано в табл. 2.1.

Таблица 2.1 – Соответствие символов и их номеров в алфавите (кольце)

_	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Этап 3 – трехпроходный алгоритм Шамира для передачи данных.

Процесс инициирует абонент *A*.

Используя обозначения, применяемые в протоколе Шамира, введем значения α и β: $\alpha = e_A$; $\beta = e_B$.

Выполним ***1-й шаг*** алгоритма Шамира у абонента А:

$C_1 = E_\alpha(m) = \left(m^{e_A}\right)(\mathrm{mod}\, p) = \left(10^5\right) \mathrm{mod}\, 23 = 19$. Это буква «*S*».

Отправляем абоненту *B*.

Расчет C_1 производится по методу «повторного возведения в квадрат».

Выполним ***2-й шаг*** алгоритма Шамира:

$C_2 = E_\beta(C_1) = \left(C_1^{e_B}\right)(\mathrm{mod}\, p) = \left(19^7\right) \mathrm{mod}\, 23 = 15$. Это буква «*O*».

Возвращаем абоненту *A*.

Расчет C_2 производится по методу «повторного возведения в квадрат».

Выполним ***3-й шаг*** алгоритма Шамира:

$C_3 = D_\alpha(C_2) = \left(C_2^{d_A}\right)(\mathrm{mod}\, p) = \left(15^9\right)\mathrm{mod}\, 23 = 14$. Это буква «*N*».

Опять отправляем абоненту *B*.

Расчет C_3 производится по методу «повторного возведения в квадрат».

У абонента *B* вычислим окончательное значение из выражения:

$m = D_\beta\left(C_3\right) = D_\beta\left(E_\beta\left(m\right)\right)$:

$m = D_\beta(C_3) = \left(C_3^{d_B}\right)(\mathrm{mod}\, p) = \left(14^{19}\right)\mathrm{mod}\, 23 = 10$.

Этап 4 – преобразование полученного значения к формату текста:

$$m = 10 = "J".$$

Полученное значение именно то, что хотел передать абонент A: *“J*”.

Расчеты для примера 2.4.1.

Расчет ключей расшифровки у абонентов *A* и *B* с использованием расширенного алгоритма Евклида приведен в табл. 2.2, 2.3.

У абонента *A* вычисляем d_A:

$$d_A = 5^{-1}(\mathrm{mod}\, 22):\ \text{НОД}(5,22) = 1 \text{ – важно!}$$

Таблица 2.2 – Реализация расширенного алгоритма Евклида для d_A

Шаг		Начальные условия	Остаток
$j = -1$		$r_{-1} = A = 5;$ $x_{-1} = 1,$ $y_{-1} = 0,$	$r_{-1} = A = 5;$

Продолжение табл. 2.2

Шаг		Начальные условия	Остаток
$j=0$		$r_0 = B = 22;$ $x_0 = 0,$ $y_0 = 1$	$r_0 = B = 22;$
Шаг	Кратность	Разложение	Остаток
$j=1$	$d_1 = \left[\frac{r_{-1}}{r_0}\right] = \left[\frac{A}{B}\right] = \left[\frac{5}{22}\right] = 0$	$r_1 = r_{-1} - d_1 \cdot r_0 = 5 - 0 \cdot 22 = 5;$ $x_1 = x_{-1} - d_1 x_0 = 1 - 0 \cdot 0 = 1;$ $y_1 = y_{-1} - d_1 y_0 = 0 - 0 \cdot 1 = 0;$	$r_1 = 5$
$j=2$	$d_2 = \left[\frac{r_0}{r_1}\right] = \left[\frac{22}{5}\right] = 4$	$r_2 = r_0 - d_2 \cdot r_1 = 22 - 4 \cdot 5 = 2;$ $x_2 = x_0 - d_2 \cdot x_1 = 0 - 4 \cdot 1 = -4;$ $y_2 = y_0 - d_2 \cdot y_1 = 1 - 4 \cdot 0 = 1;$	$r_2 = 2$
$j=3$	$d_3 = \left[\frac{r_1}{r_2}\right] = \left[\frac{5}{2}\right] = 2$	$r_3 = r_1 - d_3 \cdot r_2 = 5 - 2 \cdot 2 = 1;$ $x_3 = x_1 - d_3 \cdot x_2 = 1 - 2 \cdot (-4) = 9;$ $y_3 = y_1 - d_3 \cdot y_2 = 1 - 4 \cdot 0 = 1;$	$r_3 = 1$–*stop* Решение: $x_3 = 9$

Проверка:

$$A \cdot A^{-1} (\mathrm{mod}\, B) = 1: \ (5 \cdot 9) \bmod 22 = 45 (\mathrm{mod}\, 22) = \left(22 \cdot \left[\frac{45}{22}\right] + 1\right) \bmod 22 =$$

$$= (2 \cdot 22 + 1) \bmod 22 = (44 + 1) \bmod 22 = 1 \ - \text{верно.}$$

Или можно сразу записать как вычет:

$$(5 \cdot 9) \bmod 22 = 45 (\mathrm{mod}\, 22) = 45 - \left[\frac{45}{22}\right] \cdot 22 = 45 - 2 \cdot 22 = 45 - 44 = 1 \ - \text{верно.}$$

$$d_A = 5^{-1} (\mathrm{mod}\, 22) = 9.$$

У абонента В вычисляем d_B.

$$d_B = 7^{-1} \pmod{22}:$$

$$\text{НОД}(7,22) = 1 \text{ – Важно!}$$

Таблица 2.3 – Реализация расширенного алгоритма Евклида для d_B

Шаг		Начальные условия	Остаток
$j = -1$		$r_{-1} = A = 7;$ $x_{-1} = 1,$ $y_{-1} = 0,$	$r_{-1} = 7;$
$j = 0$		$r_0 = B = 22;$ $x_0 = 0,$ $y_0 = 1$	$r_0 = 22;$
Шаг	Кратность	Разложение	Остаток
$j = 1$	$d_1 = \left[\frac{7}{22}\right] = 0$	$r_1 = 7 - 0 = 7;$ $x_1 = 1 - 0 = 1;$ $y_1 = 0 - 0 = 0;$	$r_1 = 7$
$j = 2$	$d_2 = \left[\frac{22}{7}\right] = 3$	$r_2 = 22 - 3 \cdot 7 = 22 - 21 = 1;$ $x_2 = 0 - 3 \cdot 1 = -3;$ $y_2 = 1 - 3 \cdot 0 = 1;$	$r_2 = 1 -$ *stop* Решение: $x_2 = -3;$

Вычет $A^{-1} = 22 - 3 = 19$

Проверка:

$$A \cdot A^{-1} \pmod{B} = 1: \ (7 \cdot 19) \bmod 22 = 133 \pmod{22} = \left(22 \cdot \left[\frac{133}{22}\right] + 1\right) \bmod 22 =$$

$$= (22 \cdot 6 + 1) \bmod 22 = (132 + 1) \bmod 22 = 1 - \text{верно.}$$

Или можно сразу записать как вычет:

$$(7\cdot 19)\bmod 22 = 133(\bmod 22) = 133 - \left[\frac{133}{22}\right]\cdot 22 = 133 - 6\cdot 22 = 133 - 132 = 1 -$$

верно.

$$d_B = 7^{-1}(\bmod 22) = 19.$$

Расчет числовых значений передаваемых сообщений у абонентов *A* и *B* по алгоритму повторного возведения в квадрат приведен в табл. 2.4 – 2.11.

Первый раз у абонента *A* производим шифрование: $C_1 = 10^5(\bmod 23)$.

Таблица 2.4 – Бинарное представление степени 5

Четное число без остатка			4
Делим на 2	1	2	5
Остаток (бинарное представление числа)	1	0	1
Позиция (разряд)	2	1	0

Проверка: $n = 1\cdot 2^0 + 1\cdot 2^2 = 1 + 4 = 5$ – верно

Таблица 2.5 – Процедура «повторного возведения в квадрат» для C_1

j	n_j	b_j	a_j
0	1	$b_0 = 10(\bmod 23) = 10$	$a = b_0 = 10$
1	0	$b_1 = 10^2(\bmod 23) = 100(\bmod 23) =$ $= 100 - \left[\frac{100}{23}\right]\cdot 23 = 100 - 4\cdot 23 =$ $= 100 - 92 = 8$	$a = a = 10$
2	1	$b_2 = 8^2[\bmod 23] = 64(\bmod 23) =$ $= 64 - \left[\frac{64}{23}\right]\cdot 23 = 64 - 2\cdot 23 =$ $= 64 - 2\cdot 23 = 64 - 46 = 18$	$a = (a\cdot b_2)\bmod p =$ $(10\cdot 18)\bmod 23 =$ $= 180 - \left[\frac{180}{23}\right]\cdot 23 = 180 - 7\cdot 23 =$ $= 180 - 161 = 19$

$$C_1 = 10^5 \bmod 23 = 19.$$

У абонента *B* производим повторное шифрование сообщения:

$C_2 = 19^7 \pmod{23}$.

Таблица 2.6 – Бинарное представление степени 7

		2	6
1	1	3	7
1	1	1	1
4	2	1	0

Проверка: $n = 1 + 1 \cdot 2 + 1 \cdot 2^2 = 1 + 2 + 4 = 7$ – верно

Таблица 2.7 – Процедура «повторного возведения в квадрат» для C_2

j	n_j	b_j	a_j
0	1	$b_0 = 19 \pmod{23} = 19$	$a = 19$
1	1	$b_1 = 19^2 \pmod{23} = 361 \pmod{23} =$ $= 361 - 15 \cdot 23 = 361 - 345 = 16$	$a = (a \cdot b_1) \bmod p =$ $(19 \cdot 16) \bmod 23 =$ $= 304 \pmod{23} =$ $= 304 - 13 \cdot 23 = 304 - 299 = 5$
2	1	$b_2 = 16^2 \pmod{23} = 256 \pmod{23} =$ $= 256 - 11 \cdot 23 = 256 - 253 = 3$	$a = (a \cdot b_2) \bmod p =$ $(5 \cdot 3) \bmod 23 = 15 \pmod{23} = 15$

$$C_2 = 19^7 \pmod{23} = 15$$

Второй раз у абонента *A* производим промежуточную расшифровку:

$$C_3 = 15^9 \pmod{23}.$$

Таблица 2.8 – Бинарное представление степени 9

				8
	1	2	4	9
	1	0	0	1
	4	2	1	0

Проверка: $n = 1 + 1 \cdot 2^3 = 1 + 8 = 9$ – верно

Таблица 2.9 – Процедура «повторного возведения в квадрат» для C_3

j	n_j	b_j	a_j
0	1	$b_0 = 15 \pmod{23} = 15$	$a = 15$
1	0	$b_1 = 15^2 \pmod{23} = 225 \pmod{23} = = 225 - 9 \cdot 23 = 225 - 207 = 18$	$a = 15$
2	0	$b_2 = 18^2 \pmod{23} = 324 \pmod{23} = = 324 - 14 \cdot 23 = 324 - 322 = 2$	$a = 15$
3	1	$b_3 = 2^2 \pmod{23} = 4 \pmod{23} = 4$	$a = (a \cdot b_3) \bmod p = (15 \cdot 4) \bmod 23 = = 60 \pmod{23} = 60 - 2 \cdot 23 = 60 - 46 = 1$

$$C_3 = 15^9 (\bmod 23) = 14.$$

Второй раз у абонента *B* производим окончательную расшифровку:

$$m = 14^{19} \pmod{23}.$$

Таблица 2.10 – Бинарное представление степени 19

			8	18
1	2	4	9	19
1	0	0	1	1
4	3	2	1	0

Проверка: $n = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^4 = 1 + 2 + 16 = 19$ – верно.

Таблица 2.11 – Процедура «повторного возведения в квадрат» для окончательной расшифровки сообщения

j	n_j	b_j	a_j
0	1	$b_0 = 14 \pmod{23} = 14$	$a = 14$

Продолжение табл. 2.11

j	n_j	b_j	a_j
1	1	$b_1 = 14^2 \pmod{23} = 196 \pmod{23} = = 196 - 8 \cdot 23 = 196 - 184 = 12$	$a = (a \cdot b_1) \bmod p = (14 \cdot 12) \bmod 23 = = 168 \pmod{23} = 168 - 7 \cdot 23 = = 168 - 161 = 7$
2	0	$b_2 = 12^2 \pmod{23} = 144 \pmod{23} = = 144 - 6 \cdot 23 = 144 - 138 = 6$	$a = 7$
3	0	$b_3 = 6^2 \pmod{23} = 36 \pmod{23} = = 36 - 23 = 13$	$a = 7$
4	1	$b_4 = 13^2 \pmod{23} = 169 \pmod{23} = = 169 - 7 \cdot 23 = 169 - 161 = 8$	$a = (a \cdot b_4) \bmod p = (7 \cdot 8) \bmod 23 = = 56 \pmod{23} = 56 - 2 \cdot 23 = = 56 - 46 = 10$

$$m = 14^{19} \pmod{23} = 10.$$

2.5. Криптографическая система *RSA*

Криптосистемы с открытым ключом (асимметричные криптосистемы) – это алгоритмы, в которых ключи шифрования и дешифрования различны. При этом ключ дешифрования не может быть вычислен из ключа шифрования. Такие алгоритмы называют криптосистемами с открытым ключом, так как любой абонент, зная открытый ключ, может зашифровать сообщение, однако расшифровать зашифрованное сообщение сможет только абонент, владеющий секретным ключом.

Одним из универсальных (допускающих использование, как для шифрования, так и для цифровой подписи) асимметричных алгоритмов является алгоритм *RSA*, названный по имени разработчиков – ученых Массачусетского технологического института Рона Ривеста (*Ron Rivest*),

Ади Шамира (*Adi Shamir*) и Леонарда Адлемана (*Leonard Adleman*). Стойкость криптосистемы *RSA* основывается на сложности задачи факторизации (разложения натурального числа на простые сомножители).

Для генерации используемых в алгоритме ключей применяют два больших простых числа p и q. Для максимальной безопасности p и q должны иметь сопоставимую разрядность.

На их основе рассчитывается элемент n:

$$n = p \cdot q.$$

Затем случайным образом выбирается элемент e, такой, чтобы e и $\varphi(n)$ являлись взаимно простыми числами, т.е.

$$\text{НОД}\left(e, \varphi\left(n\right)\right) = 1.$$

По свойствам функции Эйлера (п. 1.4):

$$\varphi(n) = (p-1)\cdot(q-1).$$

Данный метод вычисления имеет сложность порядка $\underline{O}\left(\log(n)\right)$ двоичных операций. Скорость работы алгоритма шифрования *RSA* существенным образом зависит от выбора e.

Пара чисел (e,n) будет, таким образом, являться открытым ключом алгоритма.

Далее с помощью расширенного алгоритма Евклида определяется закрытый ключ d, такой что:

$$ed \equiv 1\left(\text{mod}\,\varphi(\text{n})\right), \text{ т.е. } d = e^{-1}\left(\text{mod}\,\varphi(\text{n})\right).$$

Важно отметить, что d и n также являются взаимно простыми. Числа p и q в дальнейшем не используются, однако они должны быть сохранены в секрете, т.к. их разглашение будет означать вскрытие криптосистемы.

При шифровании сообщение M разбивается на блоки фиксированной длины (меньшей, чем n). Шифрпреобразование полученных блоков M_i выполняется согласно выражению

$$C_i = M_i^e \bmod n.$$

При расшифровке сообщения для каждого зашифрованного блока C_i вычисляется

$$M_i = C_i^d \bmod n.$$

Стойкость *RSA* существенным образом зависит от выбора p и q. Как правило, практическую ценность имеет шифрование с ключами длины порядка сотен бит. Другим важным условием стойкости является требование, чтобы значение

$$\text{НОД}(p-1, q-1)$$

было по возможности наименьшим.

Криптосистема *RSA* является простой в применении, но вполне устойчивой и выступает одной из базовых схем при построении более сложных криптографических систем.

Существуют многочисленные модификации схемы *RSA*, отличающиеся повышенным быстродействием, большей стойкостью и т.д.

2.6. Этапы криптографической схемы *RSA*

Краткое изложение основ и правил применения криптографической системы *RSA* рассмотрим на конкретном примере.

Пример 2.6.1. Дано:

$p = 35279,\ q = 34361,\ m = "ROSTOV"$.

Этап 1 – генерация ключей.

Вычислим n из выражения

$$n = p \cdot q = 35279 \cdot 34361 = 1212221719.$$

Найдем $\varphi(n)$:

$$\varphi(n)=(p-1)\cdot(q-1)=1212152080.$$

Выберем значение $e=65537$.

На практике наиболее используемыми вариантами являются значения 17, 65537. Каждое из этих чисел содержит в двоичном представлении только две единицы, поэтому алгоритм повторного возведения в квадрат позволяет выполнить шифрование быстро.

Определим значение d:

$$d=e^{-1}\left(\mathrm{mod}\,\varphi(n)\right)=65537^{-1}\left(\mathrm{mod}1212152080\right)=942299953.$$

Открытый ключ:

$$(e,n)=(65537,1212221719).$$

Секретный ключ:

$$d=942299953.$$

Этап 2 **–** преобразование открытого текста в числовой эквивалент.

Выполним преобразование открытого текста в числовой эквивалент, поставив в соответствие латинскому алфавиту систему счисления по основанию 26 и выполнив перевод значения открытого текста из данной системы счисления в десятичную систему счисления:

$$m="ROSTOV"=$$

$$=17\cdot26^5+14\cdot26^4+18\cdot26^3+19\cdot26^2+14\cdot26^1+21\cdot26^0=208710653.$$

Этап 3 **–** шифрование.

Выполним шифрпреобразование:

$$c=m^e\left(\mathrm{mod}\,n\right)=208710653^{65537}\left(\mathrm{mod}1212221719\right)=301941131.$$

Этап 4 **–** преобразование шифртекста в символьное представление.

Преобразуем полученный числовой эквивалент шифртекста:

$$c = 301941131 = 25 \cdot 26^5 + 10 \cdot 26^4 + 19 \cdot 26^3 + 4 \cdot 26^2 + 12 \cdot 26^1 + 11 \cdot 26^0 =$$

$$= "ZKTEML".$$

Шифртекст: $c = "ZKTEML"$.

Этап 5 – преобразование символьного представления шифртекста в числовой эквивалент.

Выполним преобразование шифртекста в числовую форму:

$$c = "ZKTEML" = 25 \cdot 26^5 + 10 \cdot 26^4 + 19 \cdot 26^3 + 4 \cdot 26^2 + 12 \cdot 26^1 + 11 \cdot 26^0 =$$

$$= 301941131.$$

Этап 6 – дешифрование.

Выполним дешифрование:

$$m = c^d \pmod{n} = 301941131^{942299953} \pmod{1212221719} = 208710653.$$

Этап 7 – восстановление символьного представления открытого текста.

Преобразуем полученный числовой эквивалент открытого текста:

$$m = 208710653 =$$

$$= 17 \cdot 26^5 + 14 \cdot 26^4 + 18 \cdot 26^3 + 19 \cdot 26^2 + 14 \cdot 26^1 + 21 \cdot 26^0 = "ROSTOV"$$

.

Расшифрованный открытый текст:

$$m = "ROSTOV".$$

2.7. Шифрование при помощи криптосистемы *RSA* на примере с аналитическим выполнением действий

Подробное выполнение действий криптографической системы *RSA* рассмотрим на упрощенном примере, допускающем наглядное выполнение расчетов аналитически («вручную»).

Пример 2.7.1.

Сообщение хочет получить абонент *B* от абонента *A*.

Процесс инициирует абонент *B*.

Определение ключей и для шифровки и для расшифровки производится у абонента *B*.

Абонент *B* выбирает p и q, производит генерацию ключей и передает открытый ключ абоненту *A*.

Пусть: $p=7,\ q=11.$

Этап 1 – генерация ключей.

Вычислим n из выражения:

$$n=p\cdot q=7\cdot 11=77.$$

Найдем $\varphi(n)$:

$$\varphi(n)=(p-1)\cdot(q-1)=6\cdot 10=60.$$

Выбираем значение e. Пусть $e=17$.

Определим значение d:

$$d=e^{-1}\left(\mathrm{mod}\,\varphi(n)\right)=17^{-1}\left(\mathrm{mod}\,60\right)=53.$$

Для расчета используем расширенный алгоритм Евклида (все расчетные действия представлены ниже, в разделе «Расчеты»).

Открытый ключ:

$$(e,n)=(17,77).$$

Секретный ключ:

$$d=53.$$

Абонент *B* передает открытый ключ абоненту *A*.

Шифрование основного сообщения производится у абонента *A* по полученному от *B* открытому ключу.

Этап 2 – преобразование открытого текста в числовой эквивалент.

Шифруемое сообщение:

$$m="R".$$

Для примера выполним преобразование открытого текста в числовой эквивалент, поставив в соответствие латинскому алфавиту (как показано в табл. 2.12) систему счисления по основанию 26 и выполнив перевод значения открытого текста из данной системы в обычную систему счисления.

Таблица 2.12 – Соответствие символов и их номеров в алфавите (кольце)

_	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	

В числовом представлении $m = "R" = 18$.

Этап 3 – шифрование.

Выполняет шифрование:

$$c = m^e \left(\mathrm{mod}\, n\right) = 18^{17} \left(\mathrm{mod}\, 77\right) = 72.$$

Расчет производится по методу «повторного возведения в квадрат».

Этап 4 – преобразование шифртекста в символьное представление.

Преобразует полученный числовой эквивалент шифртекста в символьное представление:

$$c = 72 = 2 \cdot 26^1 + 20 \cdot 26^0 = "CU".$$

Шифртекст в символьном представлении:

$$c = "CU".$$

Абонент *A* передает зашифрованное сообщение $c = "CU"$ абоненту *B*.

У абонента *B* производится расшифровка полученного зашифрованного сообщения.

Этап 5 – преобразование полученного символьного представления шифртекста в числовой эквивалент:

$$c = "CU" = 2 \cdot 26^1 + 20 \cdot 26^0 = 72.$$

Этап 6 – расшифровка сообщения.

Для расшифровки абонент *B* использует свой секретный ключ d.

Выполняем расшифровку:

$$m = c^d \pmod{n} = 72^{53} \pmod{77} = 18.$$

Расчет производится по методу «повторного возведения в квадрат».

Этап 7 – восстановление символьного представления открытого текста.

Преобразуем полученный числовой эквивалент открытого текста

$$m = 18 = 18 \cdot 26^0 = "R".$$

Расшифрованный открытый текст:

$$m = "R".$$

Расчеты для примера 2.7.1.

Расчет ключа расшифровки у абонента *B* с использованием расширенного алгоритма Евклида приведен в таблице 2.13.

$$d = 17^{-1} \pmod{60}.$$

$$\text{НОД}(17, 60) = 1 \text{ – Важно!}$$

Таблица 2.13 – Реализация расширенного алгоритма Евклида для d

Шаг		Начальные условия	Остаток
$j = -1$		$r_{-1} = A = 17;$ $x_{-1} = 1,\ y_{-1} = 0,$	$r_{-1} = A = 17;$
$j = 0$		$r_0 = B = 60;$ $x_0 = 0,\ y_0 = 1$	$r_0 = B = 60;$
Шаг	Кратность	Разложение	Остаток
$j = 1$	$d_1 = \left[\frac{r_{-1}}{r_0}\right] = \left[\frac{A}{B}\right] =$ $= \left[\frac{17}{60}\right] = 0$	$r_1 = r_{-1} - d_1 \cdot r_0 = 17 - 0 \cdot 60 = 17;$ $x_1 = x_{-1} - d_1 x_0 = 1 - 0 \cdot 0 = 1;$ $y_1 = y_{-1} - d_1 y_0 = 0 - 0 \cdot 1 = 0;$	$r_1 = 17$

Продолжение табл. 2.13

Шаг	Кратность	Разложение	Остаток
$j=2$	$d_2=\left[\frac{r_0}{r_1}\right]=\left[\frac{60}{17}\right]=3$	$r_2=r_0-d_2\cdot r_1=60-3\cdot 17=9;$ $x_2=x_0-d_2\cdot x_1=0-3\cdot 1=-3;$ $y_2=y_0-d_2\cdot y_1=1-3\cdot 0=1;$	$r_2=9$
$j=3$	$d_3=\left[\frac{r_1}{r_2}\right]=\left[\frac{17}{9}\right]=1$	$r_3=r_1-d_3\cdot r_2=17-1\cdot 9=8;$ $x_3=x_1-d_3\cdot x_2=-3-1\cdot 4=-7;$ $y_3=y_1-d_3\cdot y_2=1-1\cdot(-1)=2;$	$r_3=1-stop$ Решение: $x_3=-7$

Вычет: $A^{-1}=60-7=53.$

Проверка:

$$A\cdot A^{-1}(\mathrm{mod}\,B)=1:\ (17\cdot 53)\mathrm{mod}\,60=901(\mathrm{mod}\,60)=$$

$$=901-\left[\frac{901}{60}\right]\cdot 60=901-15\cdot 60=901-900=1\ -\text{верно.}$$

$$d=17^{-1}(\mathrm{mod}\,60)=53.$$

Расчет числовых значений передаваемых сообщений по алгоритму повторного возведения в квадрат приведен в табл. 2.14 – 2.17.

Действия у абонента A (шифрование):

$$c=18^{17}\ (\mathrm{mod}\,77).$$

Таблица 2.14 – Бинарное представление степени 17

				16
1	2	4	8	17
1	0	0	0	1
4	3	2	1	0

Проверка: $n=1\cdot 2^0+1\cdot 2^4=1+16=17$ – верно.

Таблица 2.15 – Процедура «повторного возведения в квадрат» для c

j	n_j	b_j	a_j
0	1	$b_0 = 18 \pmod{77} = 18$	$a = 18$
1	0	$b_1 = 18^2 \pmod{77} = 324 \pmod{77} = = 324 - 4 \cdot 77 = 324 - 308 = 16$	$a = 18$
2	0	$b_2 = 16^2 \pmod{77} = 256 \pmod{77} = = 256 - 3 \cdot 77 = 256 - 231 = 25$	$a = 18$
3	0	$b_3 = 25^2 \pmod{77} = 625 \pmod{77} = = 625 - 8 \cdot 77 = 625 - 616 = 9$	$a = 18$
4	1	$b_4 = 9^2 \pmod{77} = 81 \pmod{77} = = 81 - 77 = 4$	$a = (a \cdot b_4) \bmod n = = (18 \cdot 4) \bmod 77 = = 72 \pmod{77} = 72$

$$c = 18^{17} \pmod{77} = 72.$$

Действия у абонента В (расшифровка):

$$m = 72^{53} \pmod{77}.$$

Таблица 2.16 – Бинарное представление степени 53

	2		12		52
1	3	6	13	26	53
1	1	0	1	0	1
5	4	3	2	1	0

Проверка: $n = 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^4 + 1 \cdot 2^5 = 1 + 4 + 16 + 32 = 53$ – верно.

Таблица 2.17 – Процедура «повторного возведения в квадрат» для m

j	n_j	b_j	a_j
0	1	$b_0 = 72 \pmod{77} = 72$	$a = 72$

Продолжение табл. 2.17

j	n_j	b_j	a_j
1	0	$b_1 = 72^2 \pmod{77} = 5184 \pmod{77} = = 5184 - 67 \cdot 77 = 5184 - 5159 = 25$	$a = 72$
2	1	$b_2 = 25^2 \pmod{77} = 625 \pmod{77} = = 625 - 8 \cdot 77 = 625 - 616 = 9$	$a = (a \cdot b_2) \bmod n = (72 \cdot 9) \bmod 77 = = 648 \pmod{77} = 648 - 8 \cdot 77 = = 648 - 616 = 32$
3	0	$b_3 = 9^2 \pmod{77} = 81 \pmod{77} = = 81 - 77 = 4$	$a = 32$
4	1	$b_4 = 4^2 \pmod{77} = 16 \pmod{77} = 16$	$a = (a \cdot b_4) \bmod n = (32 \cdot 16) \bmod 77 = = 512 \pmod{77} = 512 - 6 \cdot 77 = = 512 - 462 = 50$
5	1	$b_5 = 16^2 \pmod{77} = 256 \pmod{77} = = 256 - 3 \cdot 77 = 256 - 231 = 25$	$a = (a \cdot b_5) \bmod n = (50 \cdot 25) \bmod 77 = = 1250 \pmod{77} = 1250 - 16 \cdot 77 = = 1250 - 1232 = 18$

$$m = 72^{53} \pmod{77} = 18.$$

2.8. Электронная цифровая подпись в криптографической схеме Эль-Гамаля

В отличие от криптосистемы *RSA*, построенной на сложности задачи факторизации больших чисел, профессор Стэнфордского университета (США) Тахер Эль-Гамаль (*Taher El-Gamal*) разработал асимметричную криптосистему, основанную на сложности задачи дискретного логарифмирования.

Данная криптосистема (названная его именем) является универсальной, т.е. может использоваться как для шифрования данных, так и для цифровой подписи сообщений.

Для генерации пары ключей выбирается простое число p и два случайных числа g и x, меньших p:

$$g < p,\ x < p.$$

Затем вычисляется значение y из выражения

$$y = g^x \bmod p.$$

Открытым ключом является тройка чисел (y, g, p), причем g и p можно сделать общими для группы пользователей. Секретным ключом является x.

Для подписания сообщения M, сначала выбирается случайное число k, такое что

$$\text{НОД}(k, p-1) = 1.$$

Значение k в дальнейшем сохраняется в секрете. Затем вычисляется значение a из выражения

$$a = g^k \bmod p.$$

Далее с помощью расширенного метода Евклида для нахождения b решается следующее уравнение:

$$M = (xa + kb) \bmod (p-1).$$

Подписью является пара чисел (a, b). Для проверки подписи необходимо проверить справедливость равенства:

$$y^a \cdot a^b \bmod p = g^M \bmod p.$$

Электронную цифровую подпись (ЭЦП) и все ключи генерирует абонент *A* и пересылает раздельно двумя партиями абоненту *B*. Абонент *B* применяет присланные данные двух частей ЭЦП в двух формулах

раздельно и сравнивает результаты. Если они совпали, принимаем результат как успешный и подтверждающий ЭЦП.

Для каждой процедуры подписания необходимо новое значение k, выбранное случайным образом. Известны модификации данного алгоритма для доказательства идентичности, аутентификации и обмена ключами.

2.9. Этапы построения и использования цифровой подписи в криптографической схеме Эль-Гамаля

Правила построения и использования цифровой подписи в криптографической схеме Эль-Гамаля рассмотрим на конкретном примере.

Пример 2.9.1. Дано: $p = 34351,\ m = "DON"$.

Этап 1 – генерация ключей.

Выберем значения g и x, исходя из условий $g < p,\ x < p$:

$g = 32351,\ x = 32941.$

Вычислим значение y:

$y = g^x \bmod p = 32351^{32941} \bmod 34351 = 24814.$

Открытый ключ:

$(y, g, p) = (24814, 32351, 34351).$

Закрытый ключ:

$x = 32941.$

Этап 2 – преобразование сообщения в числовой эквивалент:

$m = "DON" = 3 \cdot 26^2 + 14 \cdot 26 + 13 = 2405.$

Этап 3 – подписание сообщения.

Из условия $\text{НОД}(k, p-1) = 1$ выберем значение k:

$k = 32911.$

Вычислим первую часть цифровой подписи:

$$a = g^k \bmod p = 32351^{32911} \bmod 34351 = 8941.$$

Найдем значение второй части цифровой подписи:

$$m = (x \cdot a + k \cdot b) \bmod (p-1) \Rightarrow b = (m - x \cdot a) k^{-1} \bmod (p-1);$$

$$2405 = (32941 \cdot 8941 + 32911 \cdot b) \bmod 34350;$$

$$b = 31084.$$

Цифровая подпись:

$$(a, b) = (8941, 31084).$$

Этап 4 – проверка цифровой подписи.

В выражении

$$y^a \cdot a^b \bmod p = g^m \bmod p$$

вычислим раздельно левую и правую части:

$$y^a \cdot a^b \bmod p = 24814^{8941} \cdot 8941^{31084} \bmod 34351 = 22541;$$

$$g^m \bmod p = 32351^{2405} \bmod 34351 = 22541.$$

Вывод: проверка показала идентичность подписи и сообщения.

2.10. Цифровая подпись в схеме Эль-Гамаля на примере с аналитическим выполнением действий

Подробное выполнение действий по созданию и использованию ЭЦП в схеме Эль-Гамаля рассмотрим на упрощенном примере, допускающем наглядное выполнение расчетов аналитически («вручную»).

Пример 2.10.1. У абонента *A* выберем исходные данные.

Пусть: $p = 23$, сообщение для создания основной части ЭЦП: $m = "K"$.

Этап 1 – генерация ключей.

Выберем значения g и x, исходя из условий $g < p$, $x < p$:

Пусть $g = 5,\ x = 7$.

Вычисляем значение y:

$$y = g^x \bmod p = 5^7 \bmod 23 = 17.$$

Вычисляем по алгоритму повторного возведения в квадрат (все расчетные действия представлены ниже, в разделе «Расчеты»).

Открытый ключ:

$$(y, g, p) = (17, 5, 23).$$

Закрытый ключ:

$$x = 7.$$

Отправляем открытый ключ абоненту *B*.

Продолжение действий у абонента *A*.

Этап 2 – преобразование сообщения в числовой эквивалент.

Выполним преобразование открытого текста подписи в числовой эквивалент как показано в табл. 2.18.

Таблица 2.18 – Соответствие символов и их номеров в алфавите (кольце)

_	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

В числовом представлении $m = "K" = 11.$

Это как бы «открытая подпись» абонента *A*, она то в конце концов и будет проверяться при помощи ЭЦП. Можно m тоже предварительно зашифровать, можно отправить как есть. Но если зашифровали, то нужно использовать для последующих расчетов и передавать зашифрованное числовое значение.

Этап 3 – генерация ЭЦП.

Из условия $\text{НОД}(k, p-1) = 1$ выберем значение k. Пусть $k = 3$.

Вычислим первую часть цифровой подписи:

$a = g^k \pmod p = 5^3 \pmod{23} = 10$.

Найдем значение второй части цифровой подписи:

$m = (x \cdot a + k \cdot b) \bmod (p-1) \Rightarrow$

$b = (m - x \cdot a) k^{-1} \bmod (p-1) = (11 - 7 \cdot 10) 3^{-1} \pmod{22} = 17.$

Цифровая подпись:

$(a,b) = (10,17)$.

Отправляем ЭЦП абоненту *B*.

Дальнейшие действия происходят у абонента *B*.

Этап 4 – проверка цифровой подписи.

В выражения

$$y^a \cdot a^b \bmod p = g^m \bmod p$$

вычислим раздельно левую и правую части.

Вычисление левой части уравнения:

$\left(y^a \cdot a^b\right) \bmod p = \left(17^{10} \cdot 10^{17}\right) \bmod 23 = 22$.

Используем метод цепочек:

$\left(17^{10} \cdot 10^{17}\right) \bmod 23 = \left(\left(17^{10} \pmod{23}\right) \cdot \left(10^{17} \pmod{23}\right)\right) \bmod 23.$

Вычисление правой части уравнения:

$g^m \pmod p = 5^{11} \pmod{23} = 22$.

Вывод: проверка показала идентичность ЭЦП и открытой подписи.

Для всех возведений чисел в степень используем алгоритм повторного возведения в квадрат, выполняемые при этом действия приведены ниже в табл. 2.19 – 2.27.

Расчеты для примера 2.10.1.

Расчеты у абонента *A*.

Расчет $y = g^x \bmod p$ для открытого ключа:

$$y = 5^7 \bmod 23 = 17.$$

Используем алгоритм повторного возведения в квадрат.

Таблица 2.19 – Бинарное представление степени 7

		2	6
1	1	3	7
1	1	1	1
4	2	1	0

Проверка: $n = 1 + 1 \cdot 2 + 1 \cdot 2^2 = 1 + 2 + 4 = 7$ – верно.

Таблица 2.20 – Процедура «повторного возведения в квадрат» для y

j	n_j	b_j	a_j
0	1	$b_0 = 5 \pmod{23} = 5$	$a = 5$
1	1	$b_1 = 5^2 \pmod{23} = 25 \pmod{23} =$ $= 25 - 23 = 2$	$a = (a \cdot b_1) \bmod p = (5 \cdot 2) \bmod 23 =$ $= 10 \pmod{23} = 10$
2	1	$b_2 = 2^2 \pmod{23} = 4 \pmod{23} = 4$	$a = (a \cdot b_2) \bmod p =$ $= (10 \cdot 4) \bmod 23 =$ $= 40 \pmod{23} = 40 - 23 = 17$

$y = 5^7 \bmod 23 = 17$.

Вычислим первую часть цифровой подписи:

$$a = g^k \pmod p = 5^3 \pmod{23} =$$

$$= 125 \pmod{23} = 125 - 5 \cdot 23 = 123 - 115 = 10.$$

При компьютерной реализации этого шага используется алгоритм повторного возведения в квадрат.

Найдем значение второй части цифровой подписи:

$$m = (x \cdot a + k \cdot b) \bmod (p-1) \Rightarrow b = (m - x \cdot a) k^{-1} \bmod (p-1).$$

Используем метод цепочек.

$$b=(11-7\cdot 10)3^{-1}(\mathrm{mod}\,22)=$$

$$=\left(\left((11-7\cdot 10)(\mathrm{mod}\,22)\right)\cdot\left(3^{-1}(\mathrm{mod}\,22)\right)\right)\mathrm{mod}\,22.$$

По методу цепочек раздельно считаем первую и вторую части выражения, перемножаем их и определяем вычет.

Для расчета первой части выражения используем расширенный алгоритм Евклида.

Ищем $3^{-1}(\mathrm{mod}\,22)$.

НОД$(3,22)=1$ – Важно!

Таблица 2.21 – Реализация расширенного алгоритма Евклида

Шаг		Начальные условия	Остаток
$j=-1$		$r_{-1}=A=3;$ $x_{-1}=1,\ y_{-1}=0,$	$r_{-1}=3;$
$j=0$		$r_0=B=22;$ $x_0=0,\ y_0=1$	$r_0=22;$
Шаг	Кратность	Разложение	Остаток
$j=1$	$d_1=\left[\frac{3}{22}\right]=0$	$r_1=7-0=3;$ $x_1=1-0=1;$ $y_1=0-0=0;$	$r_1=3$
$j=2$	$d_2=\left[\frac{22}{3}\right]=7$	$r_2=22-7\cdot 3=22-21=1;$ $x_2=0-7\cdot 1=-7;$ $y_2=1-7\cdot 0=1;$	$r_2=1-$ *stop* Решение: $x_2=-7;$

Вычет $A^{-1}=22-7=15$.

Проверка: $A\cdot A^{-1}(\mathrm{mod}\,B)=1$:

$(3\cdot 15)\bmod 22 = 45(\bmod 22) = 45 - \left[\frac{45}{22}\right]\cdot 22 = 45 - 2\cdot 22 = 45 - 44 = 1 -$ верно.

Получили: $3^{-1}(\bmod 22) = 15.$

Расчет второй части выражения.

$$(11 - 7\cdot 10)(\bmod 22) = (-59)\bmod 22 = \left(-(22\cdot 2 + 15)\right)(\bmod 22) =$$
$$= -15(\bmod 22) = (22 - 15)\bmod 22 = 7(\bmod 22) = 7.$$

Расчет второй части цифровой подписи:

$b = (7\cdot 15)\bmod 22 = 105\bmod 22 = 105 - 4\cdot 22 = 105 - 88 = 17.$

То же число b можно вычислить иначе:

$$b = (-15\cdot 15)\bmod 22 = -225(\bmod 22) = \left(-(22\cdot 10 + 5)\right)\bmod 22 =$$
$$= (-5)\bmod 22 = (22 - 5)\bmod 22 = 17(\bmod 22) = 17.$$

Проверка b:

$(7\cdot 10 + 3\cdot 17)\bmod 22 = 121(\bmod 22) = 121 - 5\cdot 22 = 121 - 110 = 11 (= m).$

Действия у абонента *B*.

Считаем

$\left(y^a \cdot a^b\right)\bmod p = \left(17^{10}\cdot 10^{17}\right)\bmod 23.$

Используем метод цепочек

$\left(17^{10}\cdot 10^{17}\right)\bmod 23 = \left(\left(17^{10}(\bmod 23)\right)\cdot\left(10^{17}(\bmod 23)\right)\right)\bmod 23.$

Для первой и второй части выражения используем раздельно алгоритм повторного возведения в квадрат.

Находим $17^{10}(\bmod 23)$.

Таблица 2.22 – Бинарное представление степени 10

		4	
1	2	5	10
1	0	1	0
3	2	1	0

Проверка: $n = 1 \cdot 2^1 + 1 \cdot 2^3 = 2 + 8 = 10$ – верно.

Таблица 2.23 – Процедура «повторного возведения в квадрат»

j	n_j	b_j	a_j
0	0	$b_0 = 17 \pmod{23} = 17$	$a = 1$
1	1	$b_1 = 17^2 \pmod{23} = 289 \pmod{23} =$ $= 289 - 12 \cdot 23 = 13$	$a = (a \cdot b_1) \bmod p =$ $= (1 \cdot 13) \bmod 23 = 13$
2	0	$b_2 = 13^2 \pmod{23} = 169 \pmod{23} =$ $= 169 - 7 \cdot 23 = 8$	$a = 13$
3	1	$b_3 = 8^2 \pmod{23} = 64 \pmod{23} =$ $= 46 - 2 \cdot 23 = 18$	$a = (a \cdot b_3) \bmod p =$ $= (13 \cdot 18) \bmod 23 =$ $= 234 \pmod{23} = 234 - 10 \cdot 23 = 4$

$17^{10} \pmod{23} = 4.$

Находим $10^{17} \pmod{23}$.

Таблица 2.24 – Бинарное представление степени 17

				16
1	2	4	8	17
1	0	0	0	1
4	3	2	1	0

Проверка: $n = 1 \cdot 2^0 + 1 \cdot 2^4 = 1 + 16 = 17$ – верно

Таблица 2.25 – Процедура «повторного возведения в квадрат»

j	n_j	b_j	a_j
0	1	$b_0 = 10 \pmod{23} = 10$	$a = 10$
1	0	$b_1 = 10^2 \pmod{23} = 100 \pmod{23} =$ $= 100 - 4 \cdot 23 = 8$	$a = 10$

Продолжение табл. 2.25

j	n_j	b_j	a_j
2	0	$b_2 = 8^2 \pmod{23} = 64 \pmod{23} =$ $= 46 - 2 \cdot 23 = 18$	$a = 10$
3	0	$b_3 = 18^2 \pmod{23} = 324 \pmod{23} =$ $= 324 - 14 \cdot 23 = 2$	$a = 10$
4	1	$b_4 = 2^2 \pmod{23} = 4 \pmod{23} = 4$	$a = (a \cdot b_4) \bmod p =$ $= (10 \cdot 4) \bmod 23 =$ $= 40 \pmod{23} = 40 - 23 = 17$

$10^{17} \pmod{23} = 17$.

Находим

$$(y^a \cdot a^b) \bmod p = (17^{10} \cdot 10^{17}) \bmod 23 = (4 \cdot 17) \bmod 23 =$$
$$= 68 \pmod{23} = 68 - 23 = 22.$$

Находим

$$g^m \pmod{p} = 5^{11} \pmod{23}.$$

Таблица 2.26 – Бинарное представление степени 11

		4	10
1	2	5	11
1	0	1	1
3	2	1	0

Проверка: $n = 1 + 2^1 + 2^3 = 1 + 2 + 8 = 11$ – верно.

Таблица 2.27 – Процедура «повторного возведения в квадрат»

j	n_j	b_j	a_j
0	1	$b_0 = 5 \pmod{23} = 5$	$a = 5$

Продолжение табл. 2.27

j	n_j	b_j	a_j
1	1	$b_1 = 5^2 \pmod{23} = 25 \pmod{23} =$ $= 25 - 23 = 2$	$a = (a \cdot b_1) \bmod p =$ $= (5 \cdot 2) \bmod 23 = 10$
2	0	$b_2 = 2^2 \pmod{23} = 4 \pmod{23} = 4$	$a = 10$
3	1	$b_3 = 4^2 \pmod{23} = 16 \pmod{23} = 16$	$a = (a \cdot b_3) \bmod p =$ $= (10 \cdot 16) \bmod 23 =$ $= 160 - 6 \cdot 23 = 22$

Получили:

$$g^m \pmod{p} = 5^{11} \pmod{23} = 22.$$

Сравниваем:

$$(y^a \cdot a^b) \bmod p = (17^{10} \cdot 10^{17}) \bmod 23 = 22.$$

Вывод: Значения выражений совпадают, ЭЦП подтверждена.

2.11. Односторонние хеш-функции

Для подтверждения достоверности пришедшего сообщения (его целостности и полного соответствия исходному сообщению) широко используются «свертки», реализуемые при помощи хеш-функций.

Хеш-функция – односторонняя функция, которая преобразует сообщение произвольной длины в число («свёртку») фиксированной длины. Для криптографической хеш-функции (в отличие от хеш-функции общего назначения) сложно вычислить обратную и даже найти два сообщения с одинаковой хеш-функцией. Для того, чтобы хеш-функция *H* считалась криптографически стойкой, она должна удовлетворять трём основным требованиям, на которых основано большинство применений хеш-функций в криптографии:

– необратимость или стойкость к восстановлению прообраза, т.е. для заданного значения хеш-функции m не должен быть вычислен блок данных X, для которого $H(X)=m$;

– стойкость к коллизиям первого рода или восстановлению вторых прообразов, т.е. для заданного сообщения M должно быть вычислительно невозможно подобрать другое сообщение N, для которого $H(N)=H(M)$;

– стойкость к коллизиям второго рода, т.е. должно быть вычислительно невозможно подобрать пару сообщений (M,N), имеющих одинаковое значение хеш.

Для криптографических хеш-функций также важно, чтобы при малейшем изменении аргумента значение хеш-функции сильно изменялось (лавинный эффект). В частности, значение хеш не должно давать утечки информации даже об отдельных битах аргумента. Это требование является залогом криптостойкости алгоритмов хеширования пользовательских паролей для получения ключей.

В общем случае в основе построения хеш-функции лежит итеративная последовательная схема. Ядром алгоритма является сжимающая функция – преобразование *k* входных в *n* выходных бит. В качестве сжимающей функции можно использовать симметричный блочный алгоритм шифрования. Для обеспечения большей безопасности можно использовать в качестве ключа блок данных, предназначенный к хешированию на данной итерации, а результат предыдущей сжимающей функции использовать в качестве входа. Тогда результатом последней итерации будет выход алгоритма. В таком случае безопасность хеш-функции базируется на безопасности используемого алгоритма.

ЗАКЛЮЧЕНИЕ

Обеспечение информационной безопасности в настоящее время является актуальной задачей, что настоятельно требует подготовки специалистов в данной области. Особое внимание для обучения студентов направлений информационных технологий требует изучение методов криптографии, практически значимых при разработке и использовании современных информационных систем.

В данном учебно-методическом пособии представлен структурированный и подробно иллюстрированный примерами материал, значимый в процессе проведения практических и лабораторных занятий. Издание является полезным дополнением к учебным пособиям, представляющим теоретические основы криптографии и защиты информации [1–5].

Подробный разбор выполнения алгоритмов криптографических систем (раздел 2) на конкретных примерах, в том числе примеры наглядного пошагового аналитического выполнения криптографических алгоритмов для упрощенных задач позволяет студенту освоить и полноценно представить принципы и особенности решения существенных задач криптографии с использованием ее математических основ и ключевых алгоритмов (раздел 1).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Нестеров С.А. Основы информационной безопасности : учеб. пособие. – СПб. : Лань, 2016. – 324 с.

2. Рябко Б.Я., Фионов А.Н. Криптографические методы защиты информации: учеб. пособие. – М. : Горячая линия–Телеком, 2012. – 229 с.

3. Шнайер Б. Прикладная криптография: Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Изд-во Триумф, 2002. – 816 с.

4. Зайцев Р.Г., Спиридонова И.А. Защита информации и основы прикладной криптографии : учеб. пособие. – Новочеркасск: ЮРГПУ (НПИ), 2016. – 80 с.

5. Макогоненко Г.И., Мохов В.А. Информационная безопасность вычислительных систем и сетей : учеб. пособие. – Новочеркасск : ЮРГТУ (НПИ), 2011. – 224 с.

6. Электронные материалы Интернет-портала «Википедия»: https://ru.wikipedia.org

ПРИЛОЖЕНИЕ

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

Варианты индивидуальных заданий для программной реализации криптографических систем приведены в табл. П1 – П3.

Таблица П1 – Варианты заданий для реализации протокола Шамира

№	p	M	№	p	M
0	39097	*ADV*	25	43787	*MID*
1	45641	*AKM*	26	45319	*MOD*
2	36497	*ALT*	27	42463	*MPI*
3	62423	*APP*	28	53017	*MVD*
4	58169	*ASP*	29	55021	*MVS*
5	64109	*AWT*	30	56663	*NFS*
6	51413	*BAT*	31	52183	*NMT*
7	63929	*CHR*	32	58111	*ORD*
8	57719	*COS*	33	53527	*PPT*
9	34231	*CSS*	34	58391	*QOS*
10	63131	*DBF*	35	34693	*RMI*
11	45833	*DIV*	36	63929	*RND*
12	61153	*DOS*	37	43399	*RPC*
13	32887	*DSP*	38	38189	*RUR*
14	57943	*EXP*	39	35159	*SIN*
15	54323	*FAT*	40	59957	*SKS*
16	47491	*FOR*	41	40583	*SMS*
17	35291	*GCD*	42	46153	*SSL*
18	62219	*GPS*	43	43613	*TAN*
19	34259	*GSM*	44	38713	*TBL*
20	35059	*INT*	45	44207	*TDK*
21	64871	*JDK*	46	47659	*UDP*
22	40637	*LBR*	47	48239	*VMS*
23	55163	*LOG*	48	41117	*XML*
24	63857	*MDB*	49	43283	*XSL*

Таблица П2 – Варианты заданий для реализации системы *RSA*

№	p	q	M	№	p	q	M
0	32911	38189	*APACHE*	25	57143	59957	*PRIMUS*
1	38189	60127	*ARCSIN*	26	59957	36563	*VOLUME*
2	60127	46619	*CAESAR*	27	36563	57731	*PUBLIC*
3	46619	41113	*CIPHER*	28	57731	35251	*RANDOM*
4	41113	43711	*CRYPTO*	29	35251	47947	*REDHAT*
5	43711	55439	*DIFFIE*	30	47947	59693	*REVERS*
6	55239	65147	*EUCLID*	31	59693	38699	*RIVEST*
7	65147	41131	*EUROPE*	32	38699	60589	*RUBINE*
8	41131	55799	*FERMAT*	33	60589	61717	*RUSSIA*
9	55799	65141	*GEPARD*	34	61717	49739	*SAFETY*
10	65141	36209	*GOOGLE*	35	49739	53693	*SATURN*
11	36209	54419	*LEGION*	36	53693	63781	*SECURE*
12	54419	59221	*LEMANN*	37	63781	61673	*SERVER*
13	59221	33829	*LESSON*	38	61673	56131	*SHAMIR*
14	33829	41611	*MOBILE*	39	37799	50287	*STUDIO*
15	41611	41381	*MOSAIC*	40	50287	43991	*SURFER*
16	41381	48079	*MOSCOW*	41	43991	61667	*TELNET*
17	48079	55487	*NATIVE*	42	61667	52733	*VACUUM*
18	55487	63853	*NEPTUN*	43	52733	60913	*VELVET*
19	63853	42181	*NEVADA*	44	60913	45763	*VENERA*
20	42181	32911	*NORMAN*	45	45763	50383	*PRINCE*
21	32911	53441	*OXYGEN*	46	50383	34061	*WANTED*
22	53411	33247	*PARSER*	47	34061	60719	*WINDOW*
23	33247	61613	*PLUTON*	48	60719	63367	*WORKER*
24	61613	57143	*PORTAL*	49	63367	32911	*XERXES*

Таблица П3 – Варианты заданий для реализации цифровой подписи в схеме Эль-Гамаль

№	p	M	№	p	M
0	45989	*ART*	25	42019	*NTV*
1	58109	*ASP*	26	40283	*ORB*
2	53101	*BMW*	27	54403	*ORT*
3	36973	*BNC*	28	63377	*OSI*
4	41959	*CGI*	29	36637	*PAS*
5	47501	*CPP*	30	60821	*PHP*
6	49559	*CRT*	31	41507	*RMI*
7	42467	*CTC*	32	64373	*RSA*
8	54841	*DES*	33	63901	*RTR*
9	35149	*DNS*	34	37619	*RUS*
10	53653	*DOC*	35	43579	*SAP*
11	34763	*DSS*	36	64781	*SAS*
12	61861	*EJB*	37	47653	*SDK*
13	40459	*FTP*	38	64237	*SOS*
14	45817	*GAZ*	39	59263	*SUN*
15	49207	*HTM*	40	52627	*TCP*
16	57901	*IBM*	41	63589	*TFT*
17	50867	*IDE*	42	63593	*TNT*
18	47381	*ISO*	43	46643	*UAZ*
19	48017	*JSP*	44	56239	*URI*
20	60899	*KDC*	45	38723	*URL*
21	51361	*LAN*	46	59771	*UTP*
22	58679	*LCD*	47	54449	*VAZ*
23	64109	*MSN*	48	49627	*WEB*
24	38281	*NPI*	49	45943	*WWW*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1. МАТЕМАТИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ 5
1.1. Наибольший общий делитель двух целых чисел. Классический алгоритм Евклида 5
1.2. Модулярная арифметика. Вычеты в кольце 6
1.3. «Метод цепочек» в модулярной арифметике 7
1.4. Функция Эйлера и теорема Эйлера 8
1.5. Обратное число по модулю 11
1.6. Расширенный алгоритм Евклида 13
1.7. Примеры индивидуальных заданий для обращения числа 17
1.8. Метод повторного возведения в квадрат 18
1.9. Примеры индивидуальных заданий для возведения числа в степень 20
1.10. Комментарий по переводу множества символов алфавита в цифровое представление 21
2. АЛГОРИТМИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ 26
2.1. Криптографические протоколы 26
2.2. Криптографический протокол Шамира 28
2.3. Этапы криптографической схемы «Протокол Шамира» 28
2.4. Криптографический протокол Шамира на примере с аналитическим выполнением действий 30
2.5. Криптографическая система *RSA* 38
2.6. Этапы криптографической схемы *RSA* 40
2.7. Шифрование при помощи криптосистемы *RSA* на примере с аналитическим выполнением действий 42
2.8. Электронная цифровая подпись в криптографической схеме Эль-Гамаля 49
2.9. Этапы построения и использования цифровой подписи в криптографической схеме Эль-Гамаля 51
2.10. Цифровая подпись в схеме Эль-Гамаля на примере с аналитическим выполнением действий 52
2.11. Односторонние хеш-функции 60
ЗАКЛЮЧЕНИЕ 62
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 63
ПРИЛОЖЕНИЕ. ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ 64

Учебно-методическое издание

Спиридонова Ирина Артуровна
Кущий Дарья Николаевна

МАТЕМАТИЧЕСКИЕ И АЛГОРИТМИЧЕСКИЕ
ОСНОВЫ ПРИКЛАДНОЙ КРИПТОГРАФИИ
Учебно-методическое пособие

Редактор Я. В. Максименко
Подписано в печать _30.07.2021_
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. _3,72_. Уч.-изд.л. _4,0_. Тираж 50 экз. Заказ ____.

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

АДМИНИСТРИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2020

УДК 004.71:004.72 (075.8)

Рецензент – кандидат технических наук, доцент, заведующий кафедры
«Программное обеспечение вычислительной техники»
Гринченков Дмитрий Валерьевич

Синецкий Р. М.

Администрирование компьютерных сетей : учеб. пособие / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова.– Новочеркасск: ЮРГПУ (НПИ), 2020. – 89 с.

В данном учебном пособии представлен теоретический и практический материал по ряду вопросов администрирования компьютерных сетей, который позволит студентам овладеть навыками использования специализированного программного обеспечения и командной строки для моделирования, настройки, поиска и устранения неисправностей сетевого оборудования и компьютерной сети.

Учебное пособие предназначено для студентов очной формы обучения по программе бакалавриата 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия, а также студентов заочной формы обучения по программе бакалавриата 09.03.01 Информатика и вычислительная техника

УДК 004.71:004.72 (075.8)

Содержание

Кабельная система Ethernet **4**
Стандарты и схемы подключения кабелей *Ethernet* 4
Изготовление прямого кабеля *Ethernet* 6
Проверка работоспособности прямого кабеля *Ethernet* 9
Подключение сетевого кабеля *Ethernet* к розетке 10
Проверка правильности подключения к розетке 13
Подключение узлов сети изготовленным кабелем 14
Диагностика IP-протокола **16**
Проверка настройки конфигурации компьютера 16
Проверка связи с удаленным компьютером 18
Выяснение источника неполадок в маршруте 21
Проверка настроек маршрутизации узла 22
Моделирование компьютерной сети **27**
Сетевые эмуляторы 27
Режим симуляции сети 30
Базовая конфигурация сетевого маршрутизатора **35**
Подключение и настройка маршрутизатора 35
Настройка и проверка *DHCP* 38
Сохранение конфигурации на внешнем сервере 39
Внутриполосное подключение к маршрутизатору 41
Удаление конфигурации оборудованиям 41
Анализ сетевого трафика **43**
Сбор и анализ данных протокола *ICMP* 43
Анализ обмена сообщениями *ARP* 48
Перехват установки сессии TCP 50
Настройка протокола маршрутизации *RIP* **53**
Тестовая сеть 53
Базовая настройка маршрутизатора 54
Проверка общей связности в сети 55
Настройка динамической маршрутизации 56
Планирование адресация *IPv*4 в компьютерных сетях **57**
Двоичные побитовые операции над *IP*-адресами 57
Вычисление адресов для заданной сети 60
Классификация *IPv*4-адресов 62
Определение количества подсетей 65
Разбиение сети на подсети 71
Планирование сетевой адресации 77
Библиографический список **88**

КАБЕЛЬНАЯ СИСТЕМА ETHERNET

Стандарты и схемы подключения кабелей *Ethernet*

Правила использования неэкранированных витых пар в локальных средах определяют стандарты *TIA/EIA*. Существует два коммерческих кабельных стандарта для локальных сетей: *TIA/EIA* 568-*A* и 568-*B*, которые широко применяются в разводке локальных сетей для организаций и, кроме прочего, определяют цвет каждого кабеля для разных контактов. Оба стандарта могут применяться для сетей 10/100/1000*Base-TX Ethernet*.

Для подключения кабеля к сетевым устройствам применяется 8-контактный разъем *RJ*-45, назначение выводов которого определяет стандарт.

Таблица 1 – Схема 568-*A TIA/EIA*

Номер контакта	Номер пары	Цвет провода	10Base-T 100Base-TX	1000Base-T
1	2	Белый/зелёный	Передача	BI_DA+
2	2	Зелёный	Передача	BI_DA-
3	3	Белый/оранжевый	Приём	BI_DB+
4	1	Синий	Не используется	BI_DC+
5	1	Белый/синий	Не используется	BI_DC-
6	3	Оранжевый	Приём	BI_DB-
7	4	Белый/коричневый	Не используется	BI_DD+
8	4	Коричневый	Не используется	BI_DD-

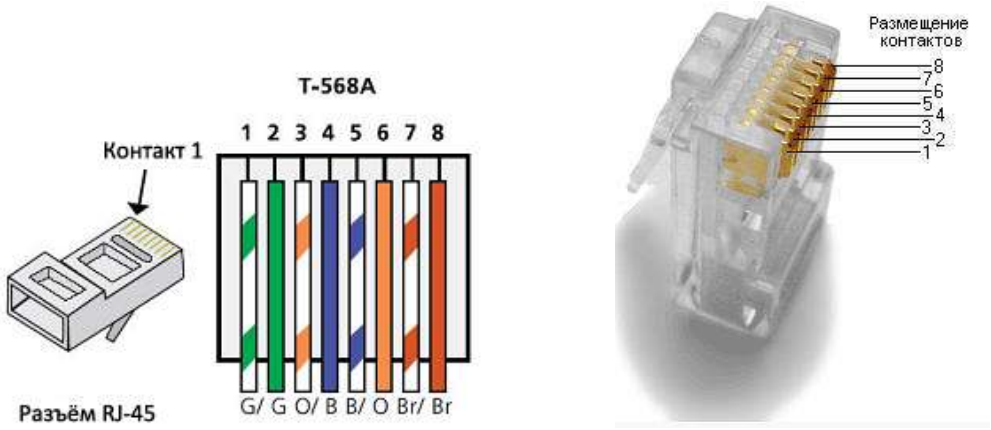


Рисунок 1 – Расположение контактов *RJ*-45

Таблица 1 демонстрирует цветовую схему и расположение выводов, а также работу четырёх пар проводов, предусмотренных стандартом *TIA/EIA* 568-*A*.

На рисунке 1 показано, как цвета и расположение выводов разъёма *RJ*-45 соотносятся со стандартом 568-*A*.

Цветовая схема и расположение выводов для стандарта **TIA/EIA 568-B** показана в таблице 2.

Таблица 2 – Схема 568-*B TIA/EIA*

Номер контакта	**Номер пары**	**Цвет провода**	**10Base-T 100Base-TX**	**1000Base-T**
1	2	Белый/оранжевый	Передача	BI_DA+
2	2	Оранжевый	Передача	BI_DA-
3	3	Белый/зелёный	Приём	BI_DB+
4	1	Синий	Не используется	BI_DC+
5	1	Белый/синий	Не используется	BI_DC-
6	3	Зелёный	Приём	BI_DB-
7	4	Белый/коричневый	Не используется	BI_DD+
8	4	Коричневый	Не используется	BI_DD-

Проанализируйте цветовое расположение контактов стандартов 568-A и 568-B. В чем разница между стандартами?

В сетевых кабельных системах применяет два типа кабеля: прямой и кроссовый.

Прямой сетевой кабель на обоих концах имеет разъем *RJ*-45 к которому подключены провода по одинаковой цветовой схеме. То есть на обоих концах прямого кабеля применен один и тот же стандарт 568-*A* и 568-*B*. Прямой кабель используется для подключения узлов сети *Ethernet* к коммутирующему оборудованию, например, компьютера к концентратору, компьютера к коммутатору, маршрутизатора к коммутатору и т.п. Подключать узлы сети друг к другу прямым кабелем не рекомендуется (например, компьютер к компьютеру или компьютер к маршрутизатору). В этом случае некоторые сетевые узлы могут не работать.

В ***кроссовом кабеле*** вторая и третья пары разъёма *RJ*-45 на одном конце кабеля переворачиваются на другом конце, что меняет местами пары отправки и приёма. На одном конце кабеля исполь-

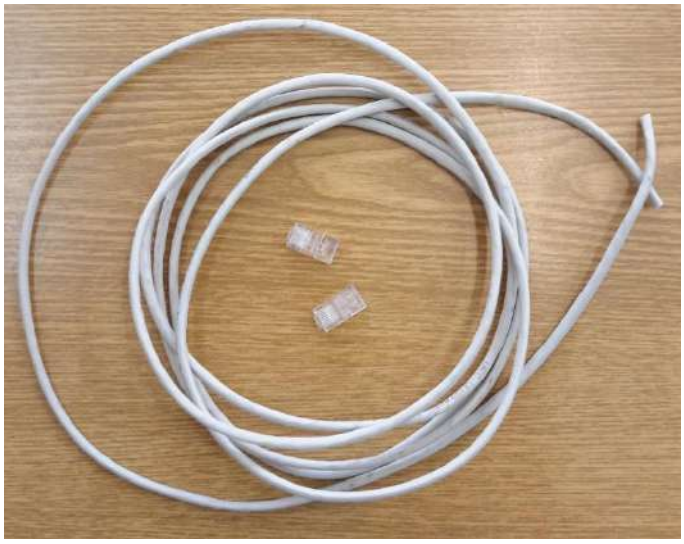
зуется схема подключения кабеля со стандартом 568-*A*, а на другом – со стандартом 568-*B*. Кроссовые кабели обычно используются для подключения концентраторов к концентраторам, коммутаторов к коммутаторам, маршрутизаторов к маршрутизаторам, а также компьютеров к компьютерам напрямую.

Современные сетевые устройства могут иметь функцию автоматического определения типа кабеля и инвертирования приемопередающих пар, что позволяет использовать для их подключения как прямой, так и кроссовый кабель.

Объясните, почему кроссовый кабель позволяет связать узлы сети друг с другом, минуя коммутирующее оборудование.

Изготовление прямого кабеля *Ethernet*

Если необходимо изготовить кабель *Ethernet*, то вам понадобятся: обжимное устройство (рисунок 2, *а*), кабель *UTP* категории 5 и два разъема *RJ*-45 (рисунок 2, *б*).



а *б*

Рисунок 2 – Оборудование и материалы для изготовления кабеля

Последовательность действий выглядит следующим образом:

1) Отрежьте примерно 30–40 см кабеля.

2) Очистите от оболочки оба конца кабеля на 2 см. Раскрутите и распрямите провода, как показано на рисунке 3.

3) Выберите стандарт 568-*A* или 568-*B*, по которому будете соединять кабель. Выстройте провода в порядке, соответствующем

выбранному стандарту. При необходимости обращайтесь к рисунку.

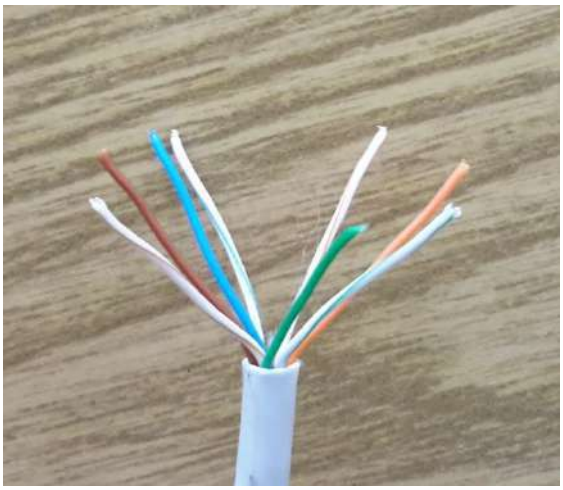


Рисунок 3 – Очистка кабеля от изоляции

4) Сплющите, выпрямите и выровняйте провода.

5) Убедитесь в том, что провода кабеля расположены в правильном порядке, соответствующем стандарту. С помощью кусачек обрежьте четыре пары в прямую линию до длины 1,3–1,7 см. Должно получиться так, как показано на рисунке 4.

Рисунок 4 – Распрямление проводов

6) На конце кабеля установите разъём *RJ*-45, выступ-зажим которого должен быть направлен вниз. Нумерация контактов начинается слева. Плотно вставьте провода в разъём *RJ*-45. Все провода должны быть видны в конце разъёма на соответствующих местах. Если провода не достигают конца разъёма, извлеките кабель, ровно обрежьте провода и вставьте провода обратно в разъём *RJ*-45. Пример правильно установленного разъема показан на рисунке 5.

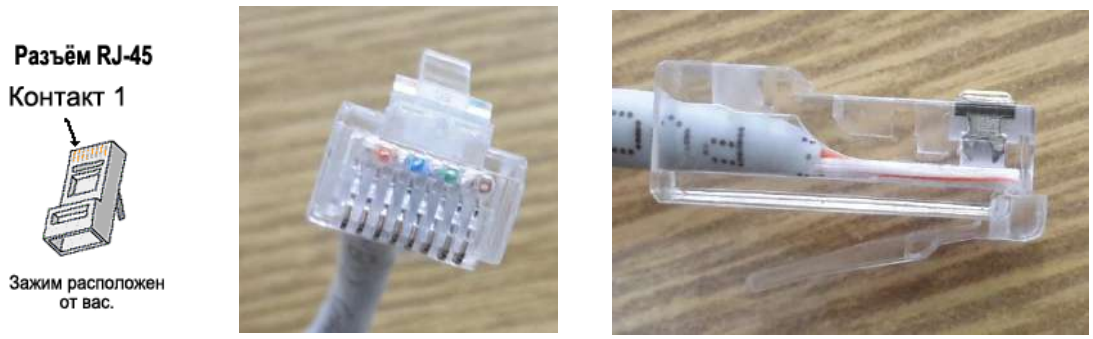


Рисунок 5 – Установка разъема

7) Если всё сделано правильно, вставьте разъём *RJ*-45 с кабелем в обжимной инструмент, ка показано на рисунке 6.

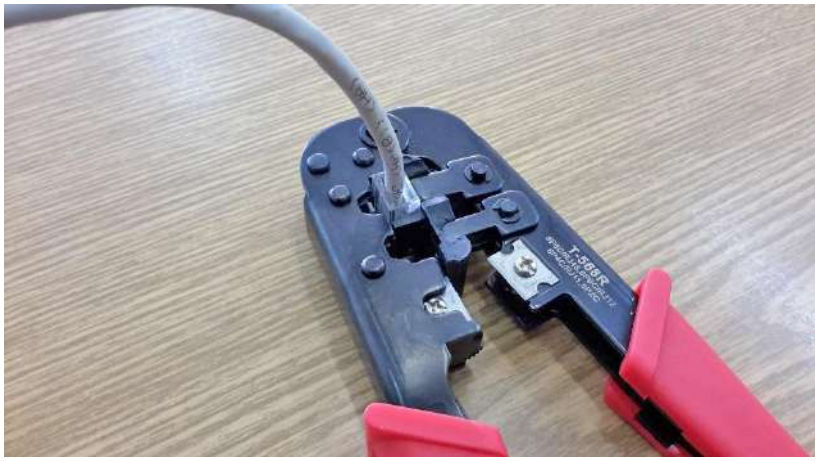


Рисунок 6 – Вставка разъема в обжимной инструмент

8) Сожмите кабель в инструменте до упора, так чтобы контакты на разъёме *RJ*-45 прошли через изоляцию проводов, обеспечивая таким образом электрическую связь. Зажим в нижней части разъема фиксирует оболочку кабеля, не давая ему выпасть из разъема. Рисунок 7 иллюстрирует этот этап.

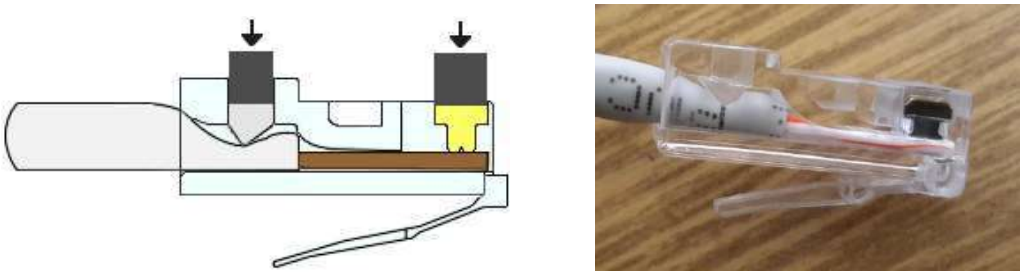


Рисунок 7 – Фиксация разъема

9) Повторите все операции с другом концом кабеля. Используйте одинаковый стандарт на обоих концах.

В результате вы получили готовый прямой кабель *Ethernet*. Прежде, чем использовать его в компьютерной сети, необходимо убедиться в его работоспособности.

Проверка работоспособности прямого кабеля *Ethernet*

Для проверки работоспособности вам понадобится кабельный тестер для кабелей *Ethernet*. Пример кабельного тестера показан на рисунке 8.

Рисунок 8 – Кабельный тестер

Вставьте оба конца готового кабеля в гнезда кабельного тестера. Передвиньте тумблер на боковой панели тестера в положение On. Светодиоды на передней панели тестера отображают соединение между проводами концов кабеля. При правильно обжатом прямом кабеле светодиоды должны загораться напротив друг друга, как показано на рисунке 9.

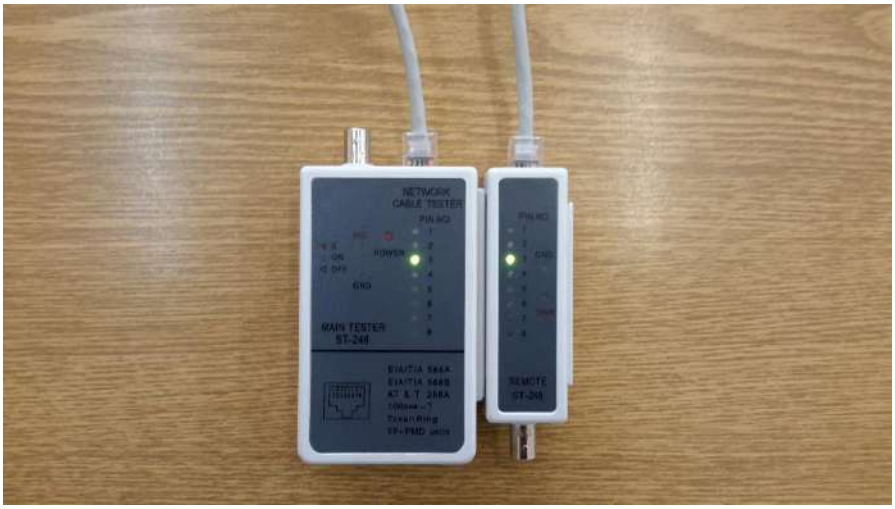


Рисунок 9 – Проверка кабеля тестером

Если какая-то пара светодиодов не загорается, это означает, что электрический контакт по этому проводу отсутствует. Если светодиоды загораются не напротив друг друга, это означает, что цвета проводов были перепутаны, или что концы кабеля обжаты

по разным стандартам (кроссовый кабель).

Ваш кабель изготовлен правильно? Если нет, опишите его неисправности и причины их возникновения.

Подключение сетевого кабеля *Ethernet* к розетке

Нередко при прокладке структурированной кабельной сети возникает необходимость в подключении сетевого кабеля к розетке. Розетки удобны тем, что располагаются на стене, к ним можно при помощи прямого кабеля подключить любой сетевой узел.

Для подключения кабеля к сетевой розетке вам понадобится: крепежный инструмент – импактор (рисунок 10, *а*), сетевая розетка (рисунок 10, *б*), обжатый с одной стороны кабель (рисунок 10, *в*).

а *б* *в*

Рисунок 10 – Оборудование и материалы
для подключения к сетевой розетке

Откройте сетевую розетку. Изнутри в ней расположены контакты для подключения сетевого кабеля, которые подведены к гнезду *Ethernet*. Чаще всего встречаются розетки на 1 или 2 гнезда, как показано на рисунке 11.

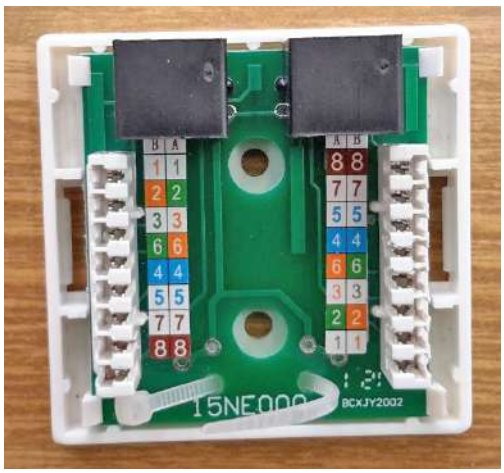


Рисунок 11 – Сетевая розетка изнутри

Возьмите кабель, обжатый с одной стороны. На одном конце кабеля уже установлен разъем *RJ*-45, обжатый по стандарту 568-*B*. С другой стороны кабеля снимите изоляцию на 3–4 см, раскрутите и распрямите провода.

Подключать кабель к розетке необходимо по стандарту 568-*B*. Порядок следования цветов провода может не совпадать с порядком для разъема *RJ*-45, потому что определяется разводкой печатной платы розетки. На самой плате нарисована правильная для этой розетки цветовая схема, необходимо ориентироваться на колонку, подписанную как *B*.

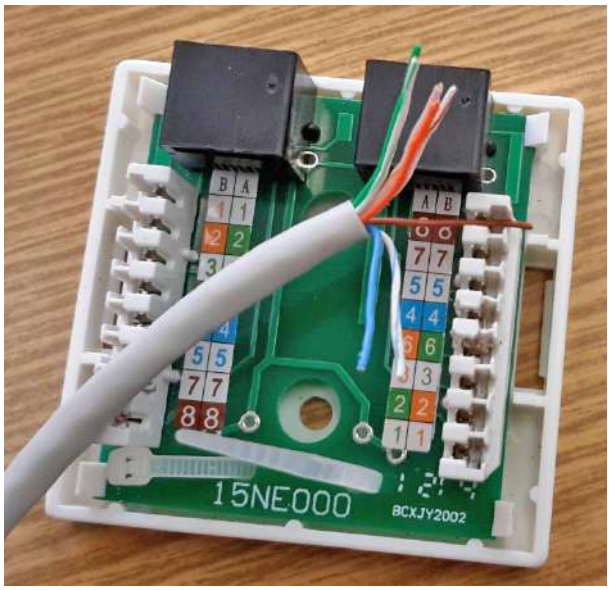


Рисунок 12 – Вставка провода в розетку

Для закрепления провода в колодке розетки заведите его изнутри розетки наружу и поместите в прорезь напротив нужного цвета, как показано на рисунке 12.

Для закрепления провода в колодке необходимо использовать крепежный инструмент, который обычно называют импактором (в иностранной литературе встречаются названия «*impact tool*», «*punch tool*» или «*krone tool*»). На конце инструмента находится вилочка, одна из сторон который имеет острую кромку для отрезания провода (рисунок 13).

Рисунок 13 – Рабочая часть импактора

Установите инструмент вилочкой в прорезь колодки розетки. *Острый край вилочки инструмента должен быть снаружи колодки*, как показано на рисунке 14. Он отрезает излишки провода. Если импактором пользовались достаточно часто, его острая кромка может затупиться, тогда провод отрезаться не будет. У некоторых моделей импакторов в комплекте имеются запасные части.

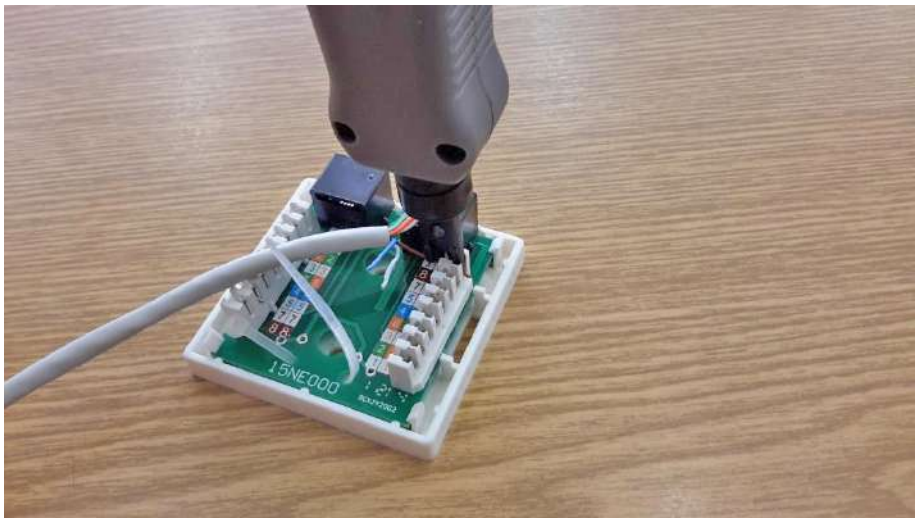


Рисунок 14 – Фиксация провода импактором

Надавите на инструмент сверху вниз до щелчка. Провод закреплен.

Закрепите при помощи инструмента все восемь проводов в соответствии с цветом по стандарту 568-*B*.

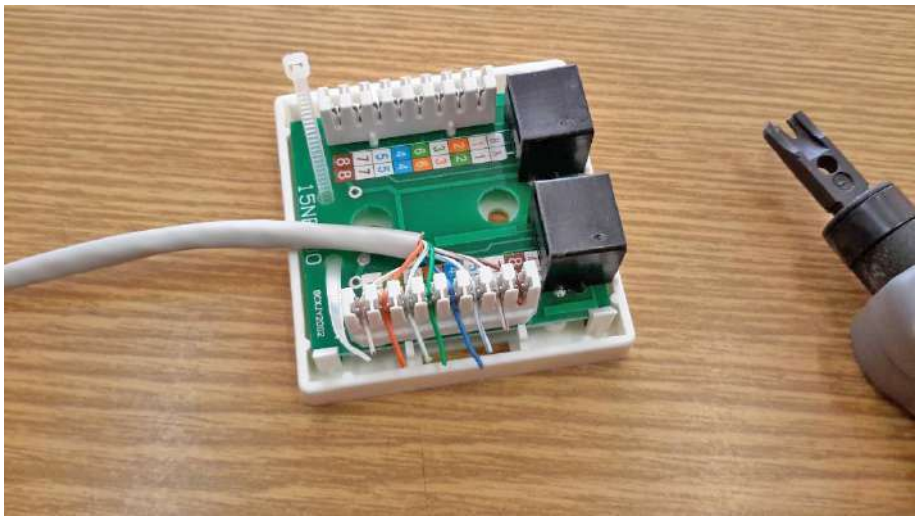


Рисунок 15 – Готовая розетка

В результате вы подключили кабель к сетевой розетке, как показано на рисунке 15. Перед использованием полученной розетки при построении сети, необходимо протестировать ее в работе.

Проверка правильности подключения к розетке

Для проверки работоспособности сетевой розетки вам понадобится кабельный тестер и готовый прямой кабель.

Вставьте один конце готового кабеля в гнездо розетки, второй конец готового кабеля вставьте в кабельный тестер. Обжатый конец провода, подключенного к розетке, вставьте в свободный разъем кабельного тестера.

Переключите тумблер тестера в положение ***On***. Аналогично проверке прямого кабеля *Ethernet*, светодиоды на передней панели тестера должны загораться напротив друг друга, как показано на рисунке 16.

Если какая-то пара светодиодов не загорается, это означает, что электрический контакт по этому проводу отсутствует. Если светодиоды загораются не напротив друг друга, это означает, что цвета проводов были перепутаны.

Ваша розетка подключена правильно? Если нет, опишите неисправности и причины их возникновения.

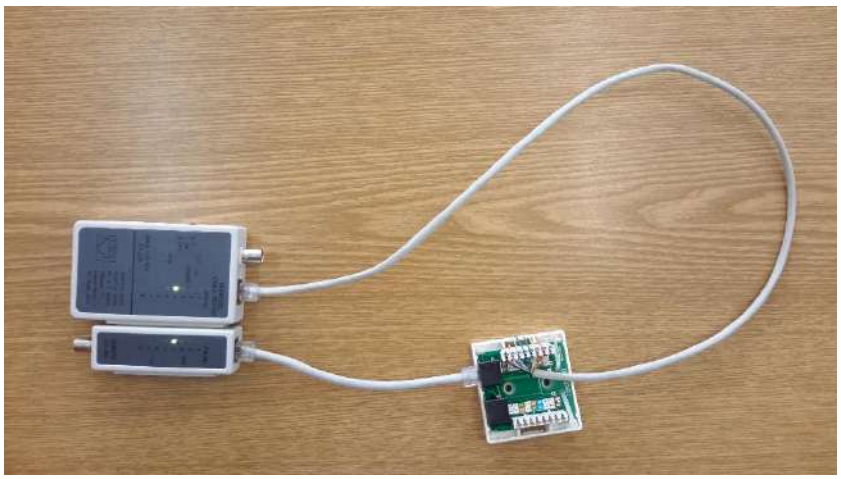


Рисунок 16 – Проверка правильности подключения к розетке

Подключение узлов сети изготовленным кабелем

Имея готовый обжатый прямой сетевой кабель можно использовать его для построения компьютерной сети. Одна из простейших топологий компьютерной сети показана на рисунке 17, а соответствующая ей схема адресации – в таблице 3.

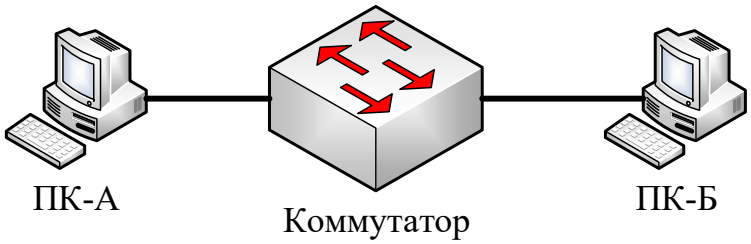


Рисунок 17 – Топология тестовой сети

Таблица 3 – Таблица адресации

Узел	Порт	IP-адрес	Маска	Шлюз	DNS
ПК 1	LAN	192.168.10.2	255.255.255.0	192.168.10.1	192.168.10.1
ПК 2	LAN	192.168.10.3	255.255.255.0	192.168.10.1	192.168.10.1

Для построения такой сети понадобится четыре исправных прямых кабеля *Ethernet* (при необходимости можно использовать готовые заводские патч-корды), два компьютера, патч-панель с подключенными розетками, коммутатор.

Соедините одним кабелем гнездо сетевой карты компьютера (интерфейс *LAN* или *CiscoLAN*) со свободным гнездом сетевой розетки на стене.

Вторым кабелем соедините на патч-панели в стойке выход сетевой розетки со свободным портом коммутатора. Аналогично подключите второй компьютер.

Установите на каждом компьютере сетевой адрес в соответствии с таблицей адресации. Для этого запустите программу *ipconf.exe* (рисунок 18) или стандартный интерфейс операционной системы для настройки IP-адреса.

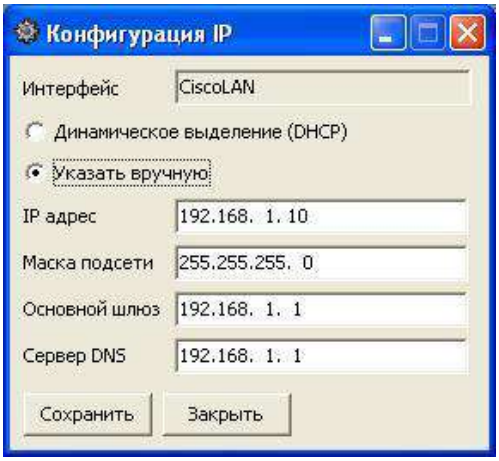


Рисунок 18 – Настройка сетевого адреса

В программе укажите адрес, маску, шлюз и *dns* в соответствии с таблицей для каждого компьютера. Нажмите кнопку сохранить и дождитесь появления окна с сообщением «Параметры установлены».

Через 15–30 секунд после подключения на коммутаторе рядом с портами, в которые подключены кабели, должны загореться зеленые светодиоды. Если они не горят, или горят оранжевым, в кабельной системе неисправность (проверьте правильность кабелей, правильность номеров портов и сетевых карт на компьютерах).

Для проверки связи между компьютерами можно использовать стандартную утилиту *ping*. Она выполняет отправку нескольких служебных пакетов другому компьютеру в сети, дожидается от него ответа и отображает время ожидания и результат. Если программа показывает, что время ожидания истекло и ответа от другого узла не получено, связи между компьютерами нет.

Запустите командную строку *Windows* (Пуск → Выполнить → *cmd.exe*).

На первом компьютере подайте команду:

```
ping 192.168.10.3
```

На втором компьютере подайте команду:

```
ping 192.168.10.2
```

Была ли проверка связи между узлами успешной? Если нет, опишите неисправности и причины их возникновения.

ДИАГНОСТИКА IP-ПРОТОКОЛА

Проверка настройки конфигурации компьютера

В операционных системах *Microsoft Windows* имеется утилита командной строки *ipconfig* для вывода деталей текущего соединения и управления клиентскими сервисами *DHCP* и *DNS*. Утилита *ipconfig* позволяет определять, какие значения конфигурации были получены с помощью *DHCP* либо заданы администратором вручную. Утилита имеет множество функций, которые задаются через ключи и параметры командной строки. Некоторые из этих ключей представлены в таблице 4.

Таблица 4 – Доступные ключи ipconfig

Ключ	Описание
/*all*	Отображение полной информации по всем адаптерам.
/*release* [адаптер]	Отправка сообщения *DHCPRELEASE* серверу *DHCP* для освобождения текущей конфигурации *DHCP* и удаления конфигурации *IP*-адресов для всех адаптеров (если адаптер не задан) или для заданного адаптера. Этот ключ отключает протокол *TCP/IP* для адаптеров, настроенных для автоматического получения *IP*-адресов.
/*renew* [адаптер]	Обновление *IP*-адреса для определённого адаптера или если адаптер не задан, то для всех. Доступно только при настроенном автоматическим получением *IP*-адресов.
/*flushdns*	Очищение *DNS* кэша.
/*registerdns*	Обновление всех зарезервированых адресов *DHCP* и перерегистрация имен *DNS*.
/*displaydns*	Отображение содержимого кэша *DNS*.
/*showclassid* адаптер	Отображение кода класса *DHCP* для указанного адаптера. Доступно только при настроенном автоматическим получением *IP*-адресов.
/*setclassid* адаптер [код_класса]	Изменение кода класса *DHCP*. Доступно только при настроенном автоматическим получением *IP*-адресов.
/?	Справка.

Для применения *ipconfig* на практике нажмите кнопку *Пуск*, выберите строку меню *Выполнить*, наберите символы *cmd* и нажмите клавишу *Enter* на клавиатуре.

В открывшемся окне наберите *ipconfig /all*. При нормальной

работе компьютера на экран должен вывестись примерно такой листинг:

```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : techstation
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Atheros L1 Gigabit
Ethernet 10/100/1000Base-T Controller
   Physical Address. . . . . . . . . : 00-1D-60-80-57-2B
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.1.20(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 192.168.1.1
   DNS Servers . . . . . . . . . . . : 87.117.0.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Tunnel adapter isatap.{6786B578-3A7B-4AEA-B733-B009EB8C7BB3}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft ISATAP
Adapter
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Teredo Tunneling
Pseudo-Interface
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . :
2001:0:5ef5:73bc:14db:d458:af01:98f1(Pref
erred)
   Link-local IPv6 Address . . . . . :
fe80::14db:d458:af01:98f1%13(Preferred)
   Default Gateway . . . . . . . . . : ::
   NetBIOS over Tcpip. . . . . . . . : Disabled
```

Попробуйте отключить сетевое подключение (можно также отключить сетевой кабель от сетевой карты компьютера), повторите команду. При отсутствующем соединении на экран выводится примерно такой листинг:

```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : techstation
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Teredo Tunneling
Pseudo-Interface
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
```

Попробуйте работу утилиты *ipconfig*. Сколько сетевых адаптеров на компьютере?

Проверка связи с удаленным компьютером

Если необходимо выяснить, работает ли удаленный компьютер, имеется ли с ним связь, то можно воспользоваться утилитой ping.

ping – утилита для проверки соединений в сетях на основе *TCP/IP*. Она отправляет запросы (*ICMP Echo-Request*) протокола *ICMP* указанному узлу сети и фиксирует поступающие ответы (*ICMP Echo-Reply*). Время между отправкой запроса и получением ответа (*RTT*, от англ. *Round Trip Time*) позволяет определять двусторонние задержки (*RTT*) по маршруту и частоту потери пакетов, то есть косвенно определять загруженность на каналах передачи данных и промежуточных устройствах.

Также ***пингом*** иногда называют время, затраченное на передачу пакета информации в компьютерных сетях от клиента к серверу и обратно от сервера к клиенту. Это время также называется

лагом (англ. *отставание*, *задержка*, *запаздывание*) или собственно задержкой и измеряется в миллисекундах. Лаг связан со скоростью соединения и загруженностью каналов на всём протяжении от клиента к серверу.

Полное отсутствие *ICMP*-ответов может также означать, что удалённый узел (или какой-либо из промежуточных маршрутизаторов) блокирует *ICMP Echo-Reply* или игнорирует *ICMP Echo-Request*.

Практическое использование утилиты *ping*:

– можно узнать, работает ли удаленный сервер.

– можно узнать завис ли только удаленный сервер или в сети глобальные проблемы.

– проблемы с настройкой *DNS* серверов на компьютере можно выявить, задав в *ping* сначала доменное имя, а потом *IP*-адрес.

– можно узнать скорость соединения, так как *ping* показывает сколько запросов удалось выполнить в секунду.

– можно узнать качество канала, посмотрев сколько ответов не пришло.

– и т.д.

Рассмотрим несколько примеров. Для проверки правильности установки протокола *TCP/IP* в системе откройте командную строку и выполните команду:

```
ping 127.0.0.1
```

Адрес 127.0.0.1 – это локальный адрес любого компьютера. Таким образом, эта команда проверяет прохождение пакетов *на самого себя*. Она может быть выполнена без наличия какого-либо сетевого подключения. Вы должны увидеть приблизительно следующие строки:

```
Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
```

```
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

По умолчанию команда посылает пакет 32 байта. Размер пакета может быть увеличен до 65 кбайт. Так можно обнаружить ошибки при пересылке пакетов больших размеров. За размером тестового пакета отображается время отклика удаленной системы (в нашем случае – меньше 1 миллисекунды). Потом показывается еще один параметр протокола – значение *TTL*. *TTL* – «время жизни» пакета. На практике это число маршрутизаторов, через которые может пройти пакет. Каждый маршрутизатор уменьшает значение *TTL* на единицу. При достижении нулевого значения пакет уничтожается. Такой механизм введен для исключения случаев зацикливания пакетов в сети.

Если будет показано сообщение о недостижимости пакетов, то это означает ошибку установки протокола IP.

Для проверки связи с другим компьютером сети выполните команду:

```
ping 192.168.100.1
```

На экран должны быть выведены примерно такие строки:

```
Pinging 192.168.100.1 with 32 bytes of data:
Reply from 192.168.100.1: bytes=32 time=491ms TTL=254
Reply from 192.168.100.1: bytes=32 time=493ms TTL=254
Reply from 192.168.100.1: bytes=32 time=490ms TTL=254
Reply from 192.168.100.1: bytes=32 time=488ms TTL=254

Ping statistics for 192.168.100.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 488ms, Maximum = 493ms, Average = 490ms
```

Наличие отклика свидетельствует о том, что канал связи установлен и работает.

Узнайте при помощи утилиты *ipconfig* адрес другого компьютера в сети и попробуйте команду *ping* с его адресом. Была ли связь успешной? Если нет, то какие могут быть причины отсутствия связи?

Выяснение источника неполадок в маршруте

При работе в Сети одни информационные серверы откликаются быстрее, другие медленнее, бывают случаи недостижимости узла. Для выяснения причин в подобных ситуациях можно использовать специальные утилиты.

Например, команда ***tracert*** показывает путь прохождения пакетов до желаемого узла. Зачастую это позволяет выяснить причины плохой связи с ним. Точка в маршруте до узла, до которой время отклика резко увеличено, свидетельствует о наличии в этом месте «узкого горлышка», не справляющегося с нагрузкой.

Утилита *tracert* выполняет отправку данных указанному узлу сети по протоколу *ICMP*, при этом отображая сведения о всех промежуточных маршрутизаторах, через которые прошли данные на пути к целевому узлу.

Программа работает только в направлении от источника пакетов. В силу особенностей работы протоколов маршрутизации в сети Интернет, обратные маршруты часто не совпадают с прямыми, поэтому *ICMP* ответ от каждого промежуточного узла может идти своим собственным маршрутом, затеряться или прийти с большой задержкой, хотя в реальности с пакетами, которые адресованы конечному узлу, этого не происходит. Кроме того, на промежуточных маршрутизаторах могут стоять ограничения протокола *ICMP*, что приводит к появлению ложных потерь. Тем не менее, в подавляющем большинстве случаев, *tracert* является очень полезным инструментом диагностики сети.

Принцип работы tracert следующий. Для определения промежуточных маршрутизаторов tracert отправляет серию (обычно три) пакетов *ICMP* целевому узлу, при этом каждый раз увеличивая на 1 значение поля *TTL*. Первая серия пакетов отправляется с *TTL*, равным 1, и поэтому первый же маршрутизатор возвращает обратно сообщение *ICMP*, указывающее на невозможность доставки данных. *Tracert* фиксирует адрес маршрутизатора, а также время между отправкой пакета и получением ответа. Затем *tracert* повторяет отправку серии пакетов, но уже с *TTL*, равным 2, что позволяет первому маршрутизатору пропустить их дальше, тогда мы получим ответ от второго маршрутизатора в пути до целевого компьютера и так далее.

Процесс повторяется до тех пор, пока при определённом значении *TTL* пакет не достигнет целевого узла. При получении ответа от этого узла процесс трассировки считается завершённым.

Рассмотрим пример использования утилиты *tracert*. В командной строке введите команду:

```
tracert ya.ru
```

Вы должны увидеть примерно такой листинг:

```
Tracing route to ya.ru [213.180.204.8]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  192.168.1.1
  2     5 ms     4 ms     5 ms  intor.novoch.ru [80.254.102.1]
  3     5 ms     5 ms     5 ms  8.108.254.80.static.donpac.ru
[80.254.108.8]
  4     6 ms     8 ms     7 ms  cs7-rmts.donpac.ru
[80.254.111.1]
  5    31 ms    28 ms    32 ms  platov-rnd-ix.yandex.net
[193.232.140.33]
  6    64 ms    66 ms    70 ms  korolev-vlan2301.yandex.net
[213.180.208.18]
  7    32 ms    32 ms    34 ms  popovich-vlan121.yandex.net
[87.250.233.110]
  8    34 ms    32 ms    32 ms  gallium-vlan2.yandex.net
[87.250.228.139]
  9    36 ms    40 ms    39 ms  ya.ru [213.180.204.8]

Trace complete.
```

Проверка настроек маршрутизации узла

Утилита ***route*** выводит на экран и изменяет записи в локальной таблице *IP*-маршрутизации. Запущенная без параметров, команда *route* выводит справку. Синтаксис команды:

```
route [-f] [-p] [команда [конечная_точка] [mask маска_сети]
[шлюз] [metric метрика]] [if интерфейс]]
```

Параметры имеют следующее значение:

-f – очищает таблицу маршрутизации от всех записей, которые не являются узловыми маршрутами (маршруты с маской подсети 255.255.255.255), сетевым маршрутом замыкания на себя (маршруты с конечной точкой 127.0.0.0 и маской подсети 255.0.0.0) или

маршрутом многоадресной рассылки (маршруты с конечной точкой 224.0.0.0 и маской подсети 240.0.0.0). При использовании данного параметра совместно с одной из команд (таких, как *add*, *change* или *delete*) таблица очищается перед выполнением команды;

-p – при использовании данного параметра с командой *add* указанный маршрут добавляется в реестр и используется для инициализации таблицы *IP*-маршрутизации каждый раз при запуске протокола *TCP/IP*. По умолчанию добавленные маршруты не сохраняются при запуске протокола *TCP/IP*. При использовании параметра с командой *print* выводит на экран список постоянных маршрутов. Все другие команды игнорируют этот параметр. Постоянные маршруты хранятся в реестре по адресу *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\PersistentRoutes*;

команда – указывает команду, которая будет запущена на удаленной системе. Допустимые команды:

add – добавление маршрута;

change – изменение существующего маршрута;

delete – удаление маршрута или маршрутов;

print – печать маршрута или маршрутов.

конечная_точка – определяет конечную точку маршрута. Конечной точкой может быть сетевой *IP*-адрес (где разряды узла в сетевом адресе имеют значение 0), *IP*-адрес маршрута к узлу, или значение 0.0.0.0 для маршрута по умолчанию;

mask **маска_сети** – указывает маску сети в соответствии с точкой назначения. Маска сети может быть маской подсети, соответствующей сетевому *IP*-адресу, например, 255.255.255.255 для маршрута к узлу или 0.0.0.0. для маршрута по умолчанию. Если данный параметр пропущен, используется маска подсети 255.255.255.255. Конечная точка не может быть более точной, чем соответствующая маска подсети. Другими словами, значение разряда 1 в адресе конечной точки невозможно, если значение соответствующего разряда в маске подсети равно 0;

шлюз – указывает *IP*-адрес пересылки или следующего перехода, по которому доступен набор адресов, определенный конечной точкой и маской подсети. Для локально подключенных марш-

рутов подсети, адрес шлюза – это *IP*-адрес, назначенный интерфейсу, который подключен к подсети. Для удаленных маршрутов, которые доступны через один или несколько маршрутизаторов, адрес шлюза – непосредственно доступный *IP*-адрес ближайшего маршрутизатора;

***metric* метрика** – задает целочисленную метрику стоимости для маршрута (в пределах от 1 до 9999), которая используется при выборе в таблице маршрутизации одного из нескольких маршрутов, наиболее близко соответствующего адресу назначения пересылаемого пакета. Выбирается маршрут с наименьшей метрикой. Метрика отражает количество переходов, скорость прохождения пути, надежность пути, пропускную способность пути и средства администрирования;

***if* интерфейс** – указывает индекс интерфейса, через который доступна точка назначения. Для вывода списка интерфейсов и их соответствующих индексов используется команда *route print*. Значения индексов интерфейсов могут быть как десятичные, так и шестнадцатеричные. Перед шестнадцатеричными номерами вводится 0x. В случае, когда параметр *if* пропущен, интерфейс определяется из адреса шлюза;

/? – отображает справку в командной строке.

Рассмотрим несколько примеров использования команды *route*. Чтобы вывести на экран все содержимое таблицы *IP*-маршрутизации, введите команду:

```
route print
```

Вывод команды на экран в этом случае приблизительно следующий:

```
===========================================================
Список интерфейсов
  9 ...00 19 d2 c4 a2 19 ...... Intel(R) PRO/Wireless 3945ABG
Network Connection

  8 ...00 16 d4 5b 0a 0c ...... Broadcom NetLink (TM) Gigabit
Ethernet
  1 ........................... Software Loopback Interface 1
 14 ...00 00 00 00 00 00 00 e0  isatap.{1DD5A5E7-25E5-4FCF-
B8E1-26FE0F2AA8EB}
 16 ...00 00 00 00 00 00 00 e0  isatap.{6E9A2AE5-1D46-40C7-
8626-9746808B4001}
 11 ...02 00 54 55 4e 01 ...... Teredo Tunneling Pseudo-Inter-
face
===========================================================
```

```
IPv4 таблица маршрута
===========================================================
Активные маршруты:
Сетевой адрес           Маска сети      Адрес шлюза       Ин-
терфейс  Метрика
          0.0.0.0          0.0.0.0    192.168.0.253
192.168.0.240     30
        127.0.0.0        255.0.0.0         On-link
127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link
127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link
127.0.0.1    306
      192.168.0.0    255.255.255.0         On-link
192.168.0.240    286
    192.168.0.240  255.255.255.255         On-link
192.168.0.240    286
    192.168.0.255  255.255.255.255         On-link
192.168.0.240    286
        224.0.0.0        240.0.0.0         On-link
127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link
192.168.0.240    286
  255.255.255.255  255.255.255.255         On-link
127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link
192.168.0.240    286
===========================================================
Постоянные маршруты:
  Отсутствует
```

Выполните команду *route print*. Сколько маршрутов на вашем компьютере?

Чтобы вывести на экран маршруты из таблицы *IP*-маршрутизации, которые начинаются с 10, введите команду:

```
route print 10.*
```

Чтобы добавить маршрут по умолчанию с адресом стандартного шлюза 192.168.12.1, введите команду:

```
route add 0.0.0.0 mask 0.0.0.0 192.168.12.1
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1
```

Чтобы добавить постоянный маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

```
route -p add 10.41.0.0 mask 255.255.0.0 10.27.0.1
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1 и метрикой стоимости 7, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 metric 7
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1 и использованием индекса интерфейса 0x3, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 if 0x3
```

Чтобы удалить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0, введите команду:

```
route delete 10.41.0.0 mask 255.255.0.0
```

Чтобы удалить все маршруты из таблицы IP-маршрутизации, которые начинаются с 10., введите команду:

```
route delete 10.*
```

Чтобы изменить следующий адрес перехода для маршрута с конечной точкой 10.41.0.0 и маской подсети 255.255.0.0 с 10.27.0.1 на 10.27.0.25, введите команду:

```
route change 10.41.0.0 mask 255.255.0.0 10.27.0.25
```

МОДЕЛИРОВАНИЕ КОМПЬЮТЕРНОЙ СЕТИ

Сетевые эмуляторы

Cisco Packet Tracer – это программа-эмулятор сетевой среды, который позволяет делать работоспособные модели сети, состоящие из рабочих персональных компьютеров (ПК), серверов и различного сетевого оборудовании (коммутаторы, маршрутизаторы и т.д.), настраивать маршрутизаторы и коммутаторы, взаимодействовать между узлами сети. Для связи между узлами сети поддерживаются различные виды кабелей (витая пара, коаксиальный кабель, оптоволокно и различные их модификации), а также беспроводные соединения *Wi-Fi*. Успешно позволяет создавать даже сложные макеты сетей, проверять на работоспособность топологии.

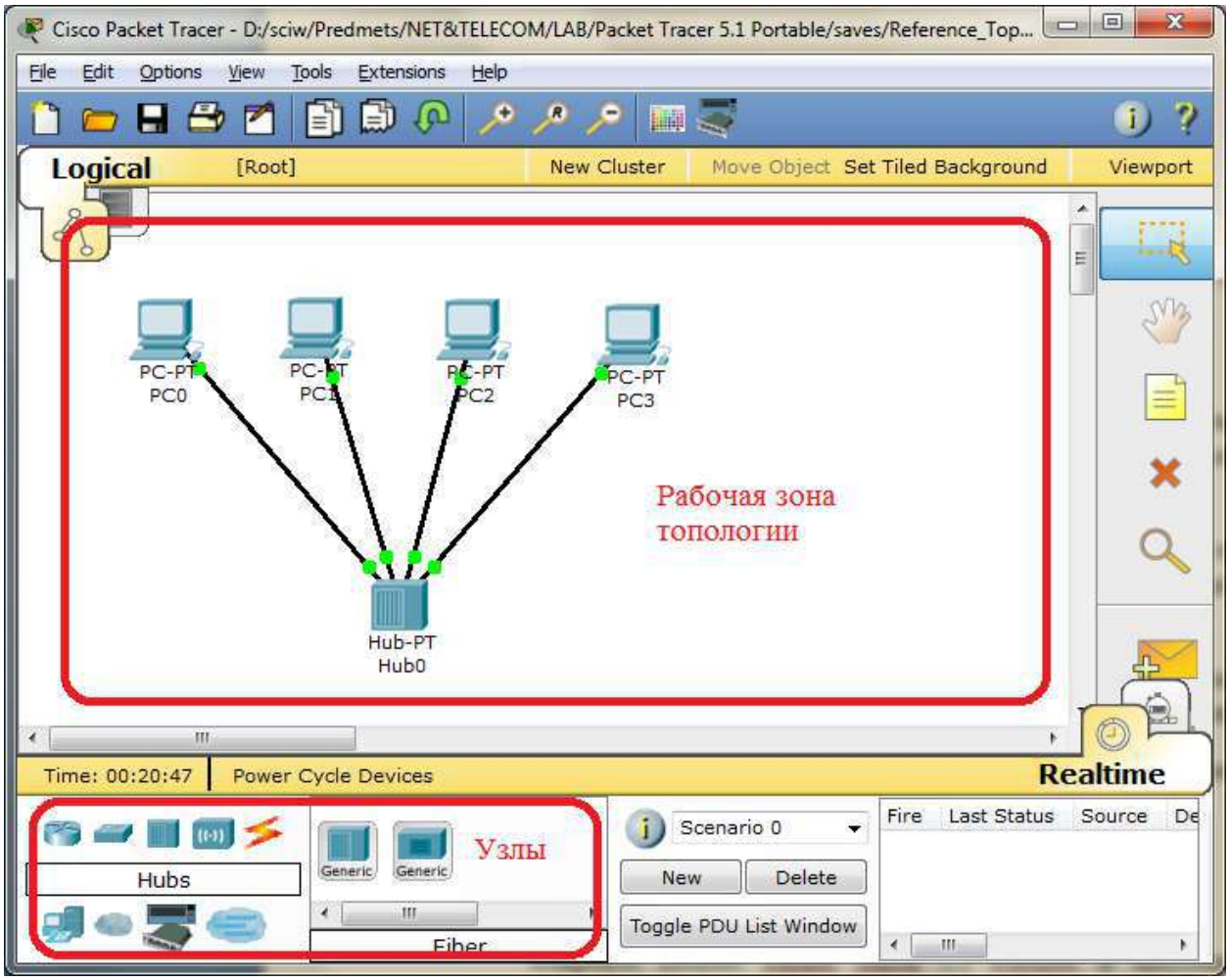


Рисунок 19 – Главное окно PacketTracer

В состав программы входит функция симуляции работы сети, которая позволяет наглядно посмотреть прохождение различных

типов пакетов по сети, изучить работу сетей в различных топологиях и с применением различного оборудования. Главное окно программы выглядит, как показано на рисунке 19.

Основными зонами в этом окне для нас являются две:

– рабочая зона топологии – содержит модель топологии конструируемой сети;

– панель узлов – содержит всевозможные сетевые устройства (ПК, хабы и т.д.), а также возможные типы связей между ними.

Панель узлов состоит из двух частей: левая часть панели содержит перечисление разделов (*Routers*, *Switches*, *Hubs*, *Wireless Devices*, *Connections*, *End Devices*, *WAN Emulation*, *Custom Made Devices*, *Multiuser Connection*), а в правой части панели для каждого раздела из левой части показаны возможные узлы сети из этого раздела. Например, если выбрать раздел *End Devices*, то будут перечислены следующие узлы: ПК, сервер, принтер, телефон, как показано на рисунке 20.

Для добавления узла выберите раздел *End Devices*, затем нажмите мышкой на иконку с изображением компьютера. Переместите курсор в зону топологии и нажмите левой кнопкой мыши на пустом месте. В этом месте будет помещен один узел сети, представляющий собой обычный ПК. Разместите недалеко еще один такой же компьютер, как показано на рисунке 21.

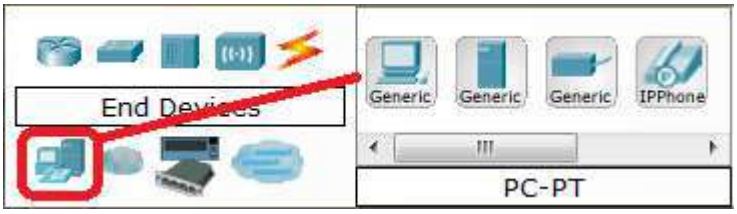


Рисунок 20 – Раздел *End Devices*

Рисунок 21 – Два узла в зоне эмуляции

Чтобы соединить узлы в сети кабелем откройте в панели узлов закладку *Connections* и выберите подключение *Copper Cross-Over* (рисунок 22).

Рисунок 22 – Закладка *Connections*

Нажмите левой кнопкой мыши на первом ПК в топологии, в появившемся ниспадающем списке выберите пункт *FastEthernet* (рисунок 23).

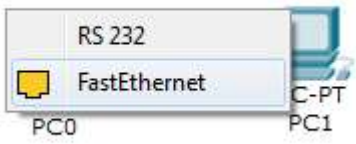


Рисунок 23 – Выбор порта подключения

Теперь подведите мышь ко второму ПК (за курсором будет тянуться линия соединения от первого ПК) и также нажмите на нем левой кнопкой мыши и выберите *FastEthernet* (рисунок 24). Теперь компьютеры подключены друг к другу.

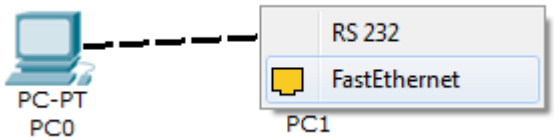


Рисунок 24 – Подключение к другому узлу

Следующим этапом для правильного функционирования сети компьютеры необходимо сконфигурировать – присвоить им различные *IP*-адреса из одной сети.

Нажмите левой кнопкой мыши на первом компьютере. Перед вами появится окно конфигурации этого компьютера. Здесь можно изменить множество параметров, среди которых *IP*-адрес и маска подсети. Они находятся на второй странице окна параметров. В левой части окна нажмите на кнопку *FastEthernet*, откроются параметры настройки сети. В поле *IP*-адреса (*IP Address*) введите адрес 192.168.1.1, а в поле маски подсети (*Subnet Mask*) введите 255.255.255.0 (рисунок 25).

После изменения окно можно закрыть стандартным крестиком в углу экрана. Изменения автоматически сохранятся. Аналогичные действия необходимо выполнить со вторым ПК, присвоить

ему *IP*-адрес 192.168.1.2 и маску 255.255.255.0.

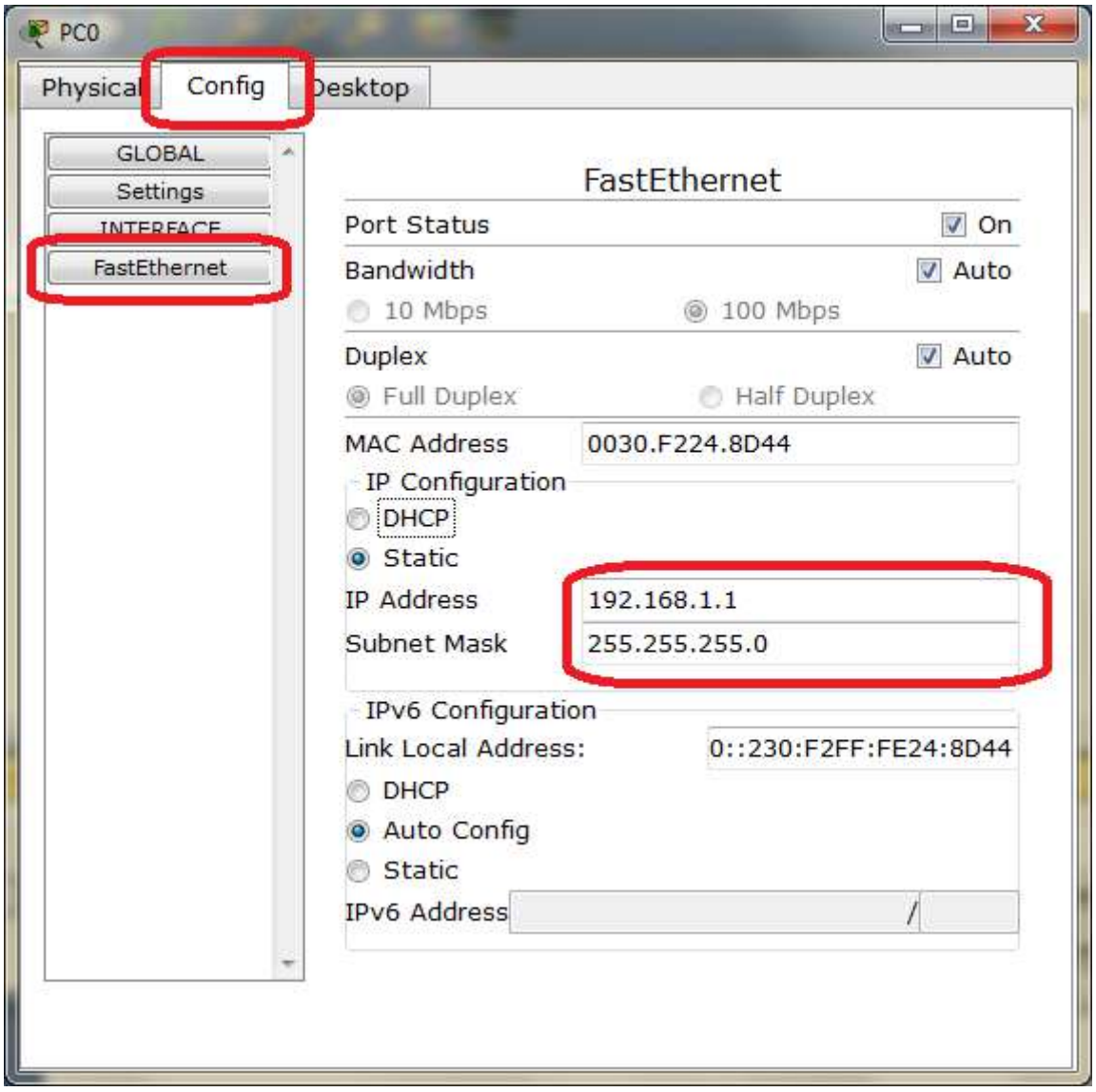


Рисунок 25 – Настройка конфигурации узла

В результате получена сеть из двух ПК, соединенных сетью Ethernet. Теперь можно симулировать прохождение пакетов по этой модели сети.

Режим симуляции сети

В правом нижнем углу программы нажмите на кнопку *Simulation Mode*, чтобы переключится в режим симуляции.

В зоне топологии появится дополнительное окно симуляции (рисунок 26). Для облегчения наблюдения над процессом симуляции в этом окне можно настроить фильтры. При стандартном функционировании сети *Ethernet* узлы обмениваются большим ко-

личеством различных пакетов. Для примера ограничимся лишь одним типом пакетов – *ICMP*. Этими пакетами обмениваются узлы при выполнении команд *ping* и *tracert* с командной строки ПК.

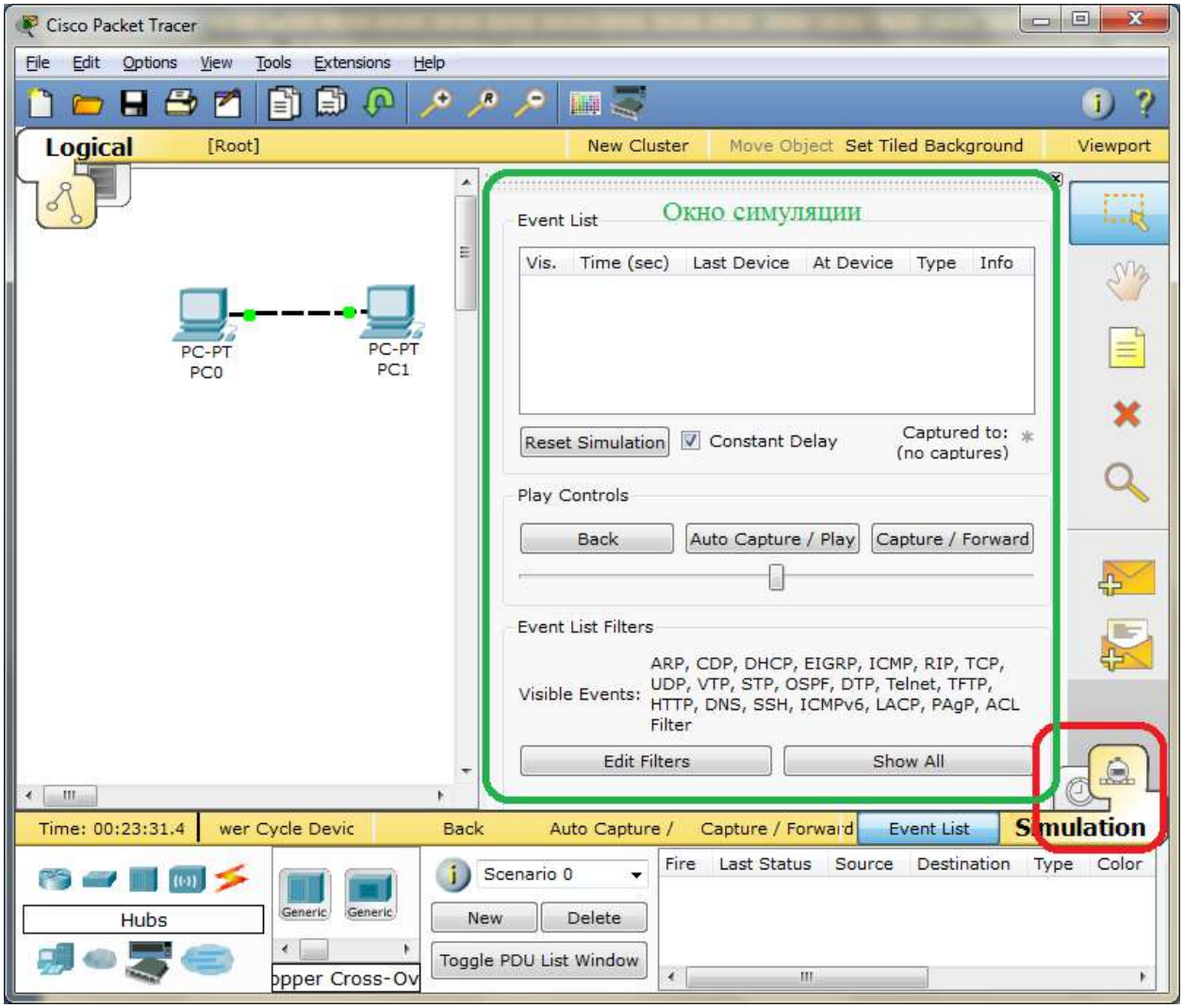


Рисунок 26 – Включение режима симуляции

Нажмите на кнопку *Edit Filters*, будет открыто окошко с множеством галочек для каждого из поддерживаемых типов пакетов, как показано на рисунке 27. Уберите все галочки, кроме *ICMP*. Это можно сделать, нажав на галочку *Show All/None*, чтобы убрать все, а потом выбрать *ICMP*. Нажатием левой кнопки мыши за пределами окошка настроек фильтра, его можно закрыть.

Теперь добавим в сеть пакет. Для этого нужно нажать на кнопку *Add Complex PDU* справа от окна симуляции, как показано на рисунке 28.

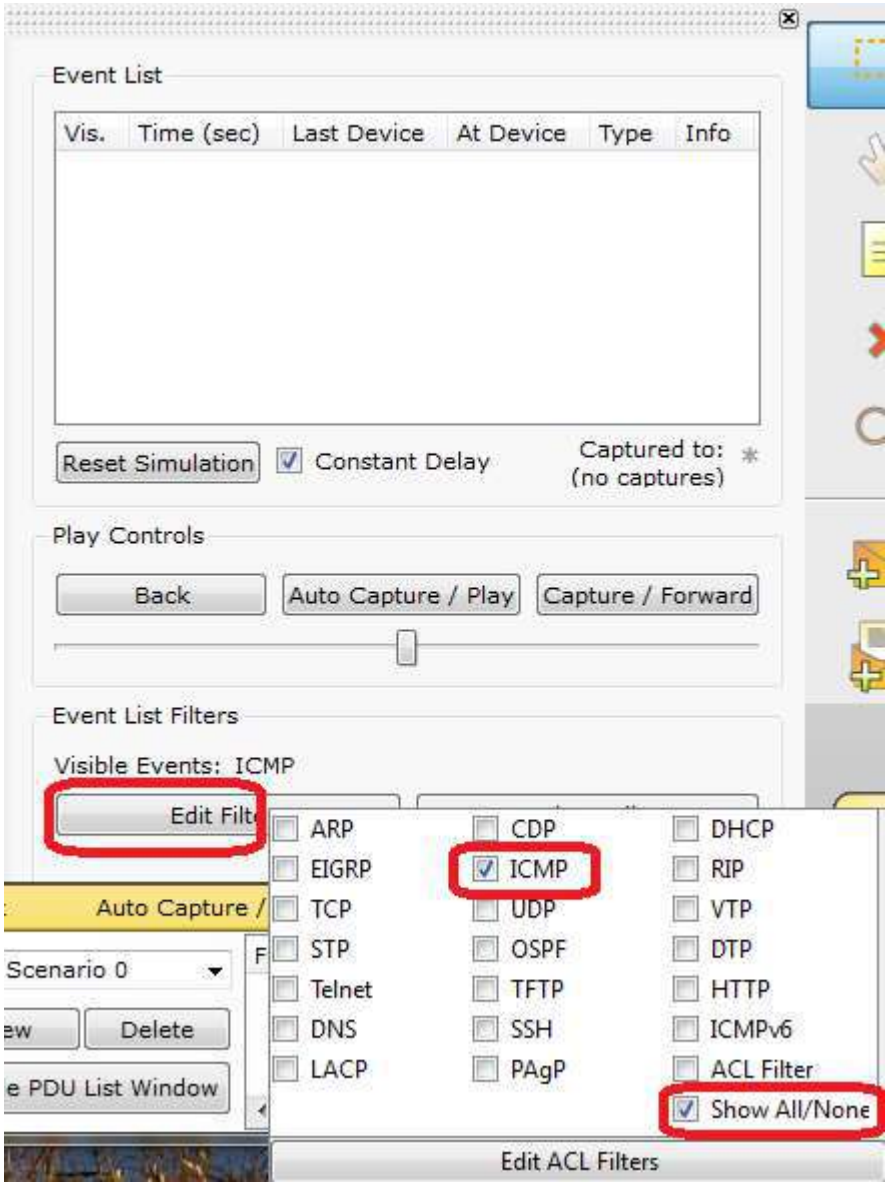


Рисунок 27 – Настройка фильтрации пакетов

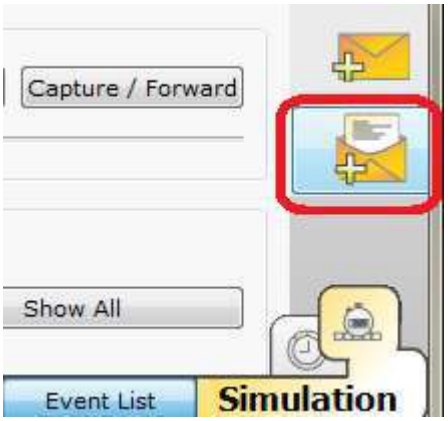


Рисунок 28 – Создание нового пакета

Курсор мыши пример вид изображения конверта. Подведите его к первому ПК (которому присваивали *IP*-адрес 192.168.1.1) и нажмите левую кнопку мыши. Появится окно с параметрами создаваемого пакета.

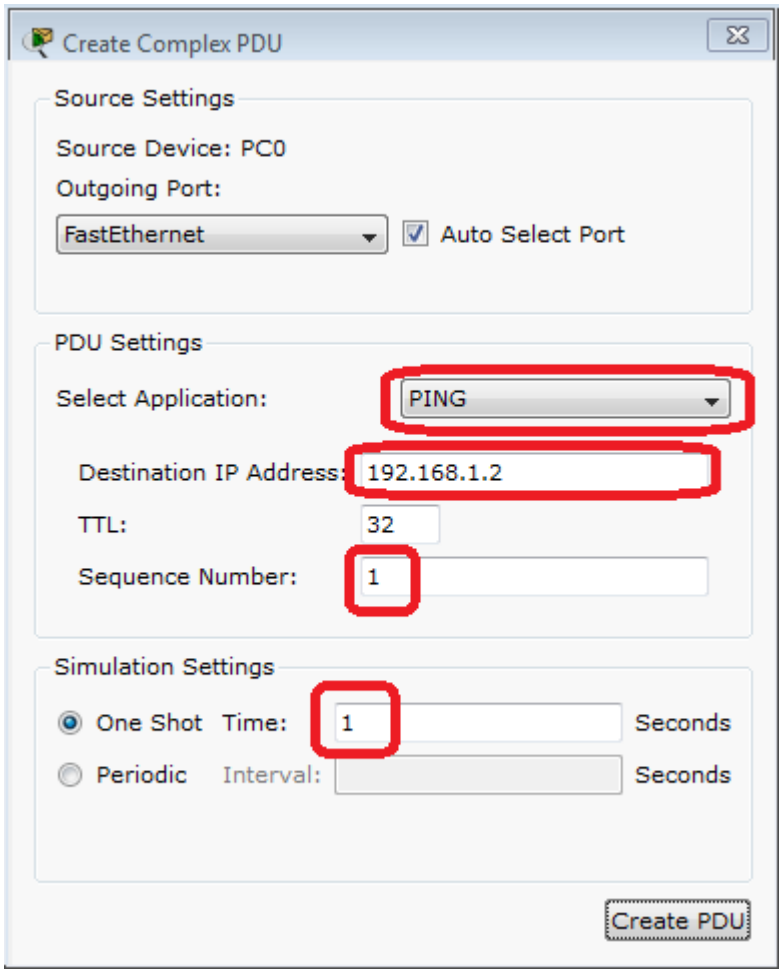


Рисунок 29 – Параметры пакета

Введите параметры аналогично рисунку 29 и нажмите кнопку *Create PDU*. Подготовка к симуляции завершена.

Для запуска симуляции нажмите кнопку *Auto Capture / Play* (рисунок 30) и проследите за движениями пакетов по сети. После того, как симуляция завершится, можно нажать кнопку *Reset Simulation* и запустить ее снова.

Кнопка *Auto Capture / Play* запускает автоматический режим симуляции, в котором можно проследить весь процесс симуляции до конца без остановок.

Для более детального изучения поведения сети удобнее воспользоваться пошаговым режимом. Для его выполнения используется кнопка *Capture / Forward*. Нажатие этой кнопки приводит к выполнению одного шага симуляции (создания пакета, передачи

пакета от одного узла к другому и т.д.).

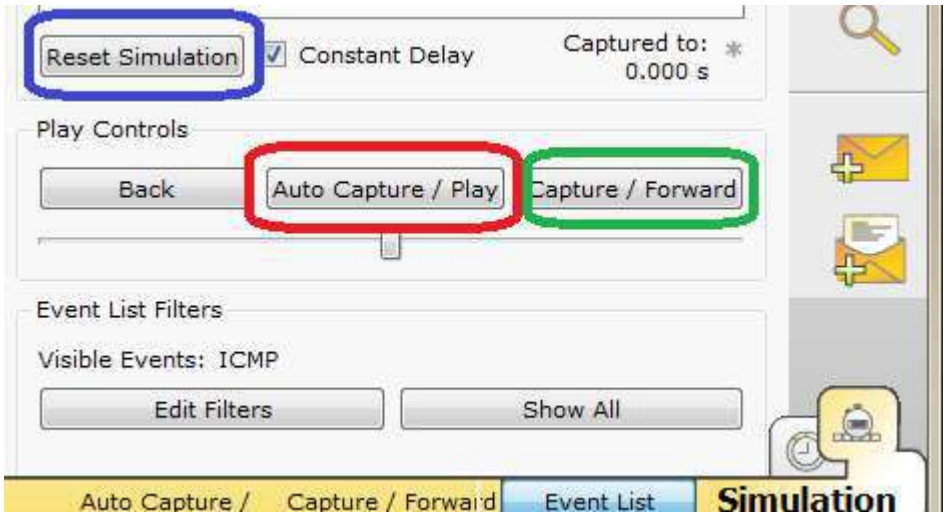


Рисунок 30 – Управление режимом симуляции

Рассмотрим подробнее процессы, которые происходят в нашей сети:

Шаг 1. Создание запроса *ICMP* на ПК1. (это запрос, который посылает ПК1 на ПК2, чтобы проверить его работу, например, командой *ping*).

Шаг 2. Передача запроса с ПК1 на ПК2.

Шаг 3. Передача ответа с ПК2 на ПК1.

Мы рассмотрели действия, которые происходят в сети при передаче *ICMP* пакетов, происходящей при выполнении команды ping с указанием *IP*-адреса соседней машины в сети.

Создайте одну модель сети, которая состоит из четырех узлов, подключенных к одному концентратору, и одну модель сети, состоящую также из четырех узлов, но с коммутатором. Проследите прохождение пакетов в обоих сетях при подаче команду *ping* от одного узла сети к другому. В чем разница в работе концентратора и коммутатора?

БАЗОВАЯ КОНФИГУРАЦИЯ СЕТЕВОГО МАРШРУТИЗАТОРА

Подключение и настройка маршрутизатора

Базовая настройка маршрутизатора является одной из основных задач при создании компьютерной сети. Рассмотрим конфигурирование устройства при примере маршрутизатора *Cisco* 1801, работающего в сети с топологией 31 и таблицей адресации 5.

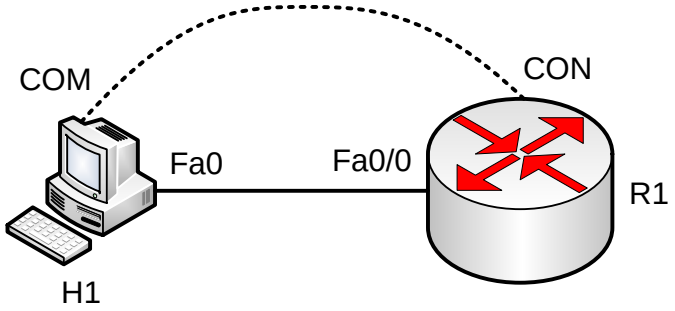


Рисунок 31 – Топология сети

Таблица 5 – Таблица адресации

Узел	Порт	IP-адрес	Маска	Шлюз	DNS
R1	Fa0/0	172.16.0.1	255.255.0.0	–	–
H1 (ПК)	FastEthernet 0	172.16.0.2	255.255.0.0	172.16.0.1	172.16.0.1

Подключите ПК (интерфейс *CiscoLAN*) перекрестным кабелем *Ethernet* к интерфейсу *Fa*0/0 маршрутизатора. Подключите последовательный интерфейс ПК (*COM*) к консольному порту (*CON*) маршрутизатора).

Задайте статический *IP*-адрес узла, используя данные из таблицы 5.

Запустите программу *HyperTerminal* и подключитесь к консоли маршрутизатора.

Войдите в конфигурационный режим и задайте имя узла:

```
Router>enable
Router#configure terminal
Router(config)#hostname R1
```

Задайте пароль ***cisco*** для консоли и включите вход с консоли:

```
R1(config)#line console 0
R1(config-line)#password cisco
R1(config-line)#login
R1(config-line)#exit
R1(config)#
```

Задайте пароль линий виртуального терминала:

```
R1(config)#line vty 0 4
R1(config-line)#password cisco
R1(config-line)#login
R1(config-line)#exit
R1(config)#
```

Задайте простой и секретный пароли привилегированного режима:

```
R1(config)#enable password cisco
R1(config)#enable secret class
R1(config)#exit
```

Секретный пароль привилегированного режима, в отличие от простого пароля, при просмотре конфигурации выводится в зашифрованном виде. Секретный пароль привилегированного режима имеет приоритет перед простым паролем привилегированного режима. После установки секретного пароля привилегированного режима простой пароль привилегированного режима больше не принимается. Можно установить только секретный пароль привилегированного режима.

Задайте заголовок новостей дня:

```
R1(config)#banner motd #Unauthorized Use Prohibited#
```

При подключении пользователя к маршрутизатору заголовок *MOTD* (*message of the day*) отображается перед запросом имени пользователя и пароля. В приведенном примере символ номера (#) используется для обозначения начала и конца сообщения. При отображении текущей конфигурации символ # превращается в ^C.

Настройте маршрутизатор таким образом, чтобы он не пытался выполнять преобразование имен узлов при использовании сервера *DNS*:

```
R1(config)#no ip domain-lookup
```

Если не выполнить эту настройку, маршрутизатор будет рассматривать команды, введенные с ошибками, как имена узлов и пытаться преобразовать их, используя сервер *DNS*. Может пройти много времени, прежде чем приглашение командной строки появится снова.

Настройте маршрутизатор таким образом, чтобы сообщения консоли не мешали введению команд:

```
R1(config)#line console 0
R1(config-line)#logging synchronous
```

Эта возможность удобна при выходе из режима конфигурации, поскольку она возвращает пользователя к командной строке

и блокирует появление сообщений в командной строке.

В командной строке привилегированного режима *EXEC* введите команду ***show running-config***. (эту команду можно сократить до ***sh run***):

```
R1#show running-config

Building configuration...
Current configuration : 605 bytes
!
hostname R1
!

*** часть выходных данных опущена ***
```

Просмотрите вывод конфигурации маршрутизатора. Есть ли зашифрованный пароль? Есть ли другие пароли? Есть ли среди других паролей зашифрованные?

Настройте интерфейс *Fast Ethernet* 0/0 на маршрутизаторе *R*1 в режиме глобальной конфигурации:

```
R1(config)#interface FastEthernet 0/0
R1(config-if)#description R1 LAN Default Gateway
R1(config-if)#ip address 172.16.0.1 255.255.0.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#exit
```

Для отображения сведений об интерфейсе введите команду ***show interfaces FastEthernet* 0/0**:

```
R1#show interfaces FastEthernet 0/0

FastEthernet0/0 is up, line protocol is up
Hardware is AmdFE, address is 000c.3076.8460 (bia
000c.3076.8460) Description: R1 LAN Default Gateway Internet
address is 172.16.0.1/16
*** часть выходных данных опущена ***
```

Просмотрите результат команды *show interfaces*. Какое состояние интерфейса *Ethernet* 0/0? Какой у него протокол канального уровня? Какой интернет-адрес? Какая инкапсуляция? На какой уровень OSI ссылается инкапсуляция?

Для проверки начальной конфигурации интерфейса можно воспользоваться командой ping.

Откройте командную строку на ПК. Для этого выберите *Start* > *Run* (Пуск > Выполнить) и введите *cmd*. Также можно выбрать

Start > *All programs* > *Accessories* > *Command Prompt* (Пуск > Все программы > Стандартные > Командная строка).

С помощью команды *ping* протестируйте подключение. Пошлите от ПК эхо-запрос интерфейсу *Fast Ethernet* маршрутизатора *R*1:

```
C:\>ping 172.16.0.1
```

Пошлите от маршрутизатора эхо-запрос на ПК:

```
R1#ping 172.16.0.2
```

Если какой-либо из пингов не успешен, найдите и устраните ошибки в конфигурации маршрутизатора или ПК, пока оба пинга не станут успешными.

Сохраните текущую конфигурацию в качестве начальной конфигурации:

```
R1#copy running-config startup-config
```

Настройка и проверка *DHCP*

Чтобы предотвратить назначение определенных адресов, их нужно исключить из пула. К таким адресам относится *IP*-адрес интерфейса *Fast Ethernet* 0/0 маршрутизатора (шлюз по умолчанию). Также будут исключены адреса в диапазоне от 172.16.0.101 до 172.16.0.254, чтобы зарезервировать их для других целей, например для назначения их серверам и принтерам, которые должны иметь фиксированные *IP*-адреса.

Для исключения адресов введите команду ***ip dhcp excluded-address***.

```
R1(config)#ip dhcp excluded-address 172.16.0.1
R1(config)#ip dhcp excluded-address 172.16.0.101 172.16.0.254
```

Для чего необходимо исключить адреса еще до того, как создан пул *DHCP*?

На маршрутизаторе настройте конфигурацию пула *DHCP* для внутренних клиентов:

```
R1(config)#ip dhcp pool INTERNAL
R1(dhcp-config)#network 172.16.0.0 255.255.0.0
R1(dhcp-config)#domain-name cisco.povt.npi-tu.ru
R1(dhcp-config)#default-router 172.16.0.1
R1(dhcp-config)#dns-server 172.16.0.1
```

Для тестирования настройки *DHCP* откройте настройки *IP*-адреса ПК и переключите на динамическую настройку. Затем откройте командную строку и введите команды ***ipconfig /release*** и ***ipconfig /renew***.

Введите в командной строке команду ***ipconfig /all***. Какой *IP*-адрес назначен узлу? Какая маска подсети? Какой шлюз по умолчанию назначен узлу? Какой *DNS*-суффикс (имя домена), зависящий от соединения, имеет узел? Какой *IP*-адрес у сервера *DHCP*? Какой *IP*-адрес у сервера *DNS*? Какой *MAC*-адрес имеет ПК?

Чтобы увидеть комбинацию *IP*-адреса и адреса аппаратного обеспечения (*MAC*), присвоенную *DHCP*-сервером, введите команду ***show ip dhcp binding*** на маршрутизаторе:

```
R1#show ip dhcp binding
```

Какой выделен адрес для ПК? Какой *MAC* адрес ПК? Через какое время заканчивается выделение *IP*-адреса?

На маршрутизаторе отобразите свойства пула *DHCP*, используя команду ***show ip dhcp pool***:

```
R1#show ip dhcp pool
```

Сколько адресов было выделено? Что означает ***Current Index*** в выходных данных команды?

Сохранение конфигурации на внешнем сервере

Запустите на ПК *Open TFTP Server*. Вы увидите текст, свидетельствующий, что сервер запущен. Сервер работает с каталогом документов пользователя на запущенном ПК.

Из сеанса *HyperTerminal* на маршрутизаторе *R*1 введите команду ***copy running-config tftp***, чтобы начать загрузку *TFTP* на *TFTP*-сервер. Отвечайте на запросы, как показано ниже. Имя файла назначения по умолчанию имеет следующий формат: имя устройства («*r*1»), тире и «*confg*». При успешном выполнении команды в окне терминала маршрутизатора отобразится сообщение, содержащее восклицательные знаки и количество скопированных байт.

```
R1#copy running-config tftp
Address or name of remote host []? 172.16.0.2
Destination filename [r1-confg]? <ENTER>
!!
1078 bytes copied in 1.188 secs (907 bytes/sec)
R1#
```

Откройте каталог пользователя на ПК, найдите файл *r*1-*confg*, просмотрите его содержимое. Что содержится в данном файле? В каком виде представлено?

Измените текущую конфигурацию *R*1. В режиме конфигурации смените имя маршрутизатора на *Router*. После этого приглашение консоли командной строки *iOS* должно смениться.

```
R1>enable
R1#configure terminal
R1(config)#hostname Router
Router(config)#end
```

Загрузите конфигурационный файл с *TFTP*-сервера с помощью команды ***copy tftp startup-config***. Отвечайте на запросы, как показано ниже. При успешном выполнении команды в окне терминала маршрутизатора отобразится сообщение, содержащее восклицательные знаки и количество скопированных байт.

```
Router#copy tftp startup-config
Address or name of remote host [172.16.0.2]? <ENTER>
Source filename [r1-confg]? <ENTER>
Destination filename [startup-config]? <ENTER>
Accessing tftp://172.16.0.2/r1-confg...
Loading r1-confg from 172.16.0.2 (via FastEthernet0/0):
!
[OK - 1078 bytes]
[OK]
1078 bytes copied in 12.780 secs (84 bytes/sec)
Router#
*Feb 17 02:18:33.551: %SYS-5-CONFIG_NV_I: Nonvolatile storage
configured from tftp://172.16.0.2/r1-confg by console
Router#
```

Просмотрите настройки в *NVRAM*, чтобы проверить, правильно ли выполнено преобразование, с помощью команды ***show startup-config*.**

Перезагрузите маршрутизатор и выберите ***No*** в ответ на запрос о сохранении измененной конфигурации.

```
Router#reload
```

Проследите процесс загрузки маршрутизатора до конца. После завершения загрузки имя должно быть *R*1. Проверьте *ping* между ПК и маршрутизатором, чтобы убедиться, что интерфейсы работают. Просмотрите текущую конфигурацию (***show running-config***), чтобы убедиться, что конфигурация загружена успешно.

Внутриполосное подключение к маршрутизатору

Запустите на ПК в командной строке клиентское приложение *telnet* с указанием IP-адреса маршрутизатора:

```
C:\>telnet.exe 172.16.0.1
```

Вы должны увидеть сообщение при входе на маршрутизатор, которое ранее было настроено командой ***banner motd*** и запрос на ввод пароля. При запросе пароля введите ***cisco*** (при вводе символа не отображаются!). Вы должны попасть в командную строку, аналогичную подключению к консоли.

Просмотрите текущую конфигурацию маршрутизатора. Отличается ли чем-нибудь интерфейс *CLI* через консоль и через *Telnet*? В каких случаях используется внутриполосное управление маршрутизатором? В каких случаях используется внеполосное управление маршрутизатором?

Для отключения от маршрутизатора по протоколу *Telnet*, наберите команду ***exit*** в *CLI*. Программа клиента *telnet* будет закрыта.

Удаление конфигурации оборудованиям

Перейдите в привилегированный режим *EXEC* с помощью команды ***enable***.

```
R1>enable
```

В привилегированном режиме *EXEC* введите команду ***erase startup-config***.

```
Router#erase startup-config
```

После этого отобразится следующее сообщение:

```
Erasing the nvram filesystem will remove all files! Continue?
[confirm]
```

Для подтверждения нажмите клавишу **ВВОД**.

После этого отобразится:

```
Erase of nvram: complete
```

В привилегированном режиме *EXEC* введите команду ***reload***.

```
R1#reload
```

После этого отобразится следующее сообщение:

```
System configuration has been modified. Save? [yes/no]:
```

Введите ***n***, а затем нажмите клавишу **ВВОД**.

Отобразится следующее сообщение:

```
Proceed with reload? [confirm]
```

Для подтверждения нажмите клавишу **ВВОД**.

Сначала отобразится следующий ответ:

```
Reload requested by console.
```

Проследите процесс начальной загрузки маршрутизатора. Какие этапы проходит маршрутизатор при включении?

После перезагрузки маршрутизатора отобразится следующее сообщение:

```
Would you like to enter the initial configuration dialog?
[yes/no]:
```

Введите ***n***, а затем нажмите клавишу **ВВОД**.

После этого отобразится следующее сообщение:

```
Press RETURN to get started!
```

Нажмите клавишу **ВВОД.**

АНАЛИЗ СЕТЕВОГО ТРАФИКА

Wireshark – это программа для анализа протоколов (анализатор пакетов), которая используется для поиска и устранения неполадок в сети, анализа, разработки программного обеспечения и протоколов, а также обучения. По мере движения потоков данных по сети анализатор перехватывает каждый протокольный блок данных (*PDU*), после чего расшифровывает или анализирует его содержание согласно соответствующему документу *RFC* или другим спецификациям.

Сбор и анализ данных протокола *ICMP*

Рассмотрим пример, как можно отправить эхо-запрос с помощью команды *ping* на другой ПК в локальной сети, перехватить *ICMP*-запросы и отклики в программе *Wireshark*, найти необходимую информацию в собранных кадрах. Этот анализ поможет понять, как используются заголовки пакетов для передачи данных по месту назначения.

Первым шагом необходимо узнать *IP*-адрес компьютера и физический адрес сетевого адаптера, который называется *MAC*-адресом.

Откройте окно командной строки, введите команду ***ipconfig /all*** и нажмите клавишу ВВОД. Пример вывода команды *ipconfig* показан на рисунке 32. Запишите *IP*-адрес интерфейса ПК и *MAC*-адрес.

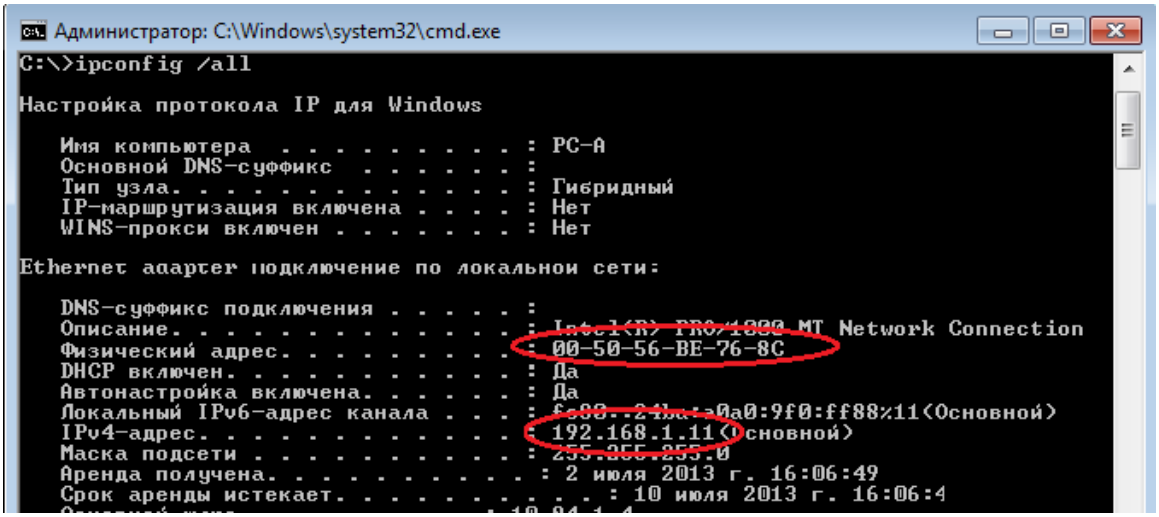


Рисунок 32 – Определение адреса компьютера

Запустите программу *Wireshark*, нажмите на параметр ***Interface list*** (Список интерфейсов, рисунок 33). Список интерфейсов можно также открыть, нажав на значок первого интерфейса в ряду значков.

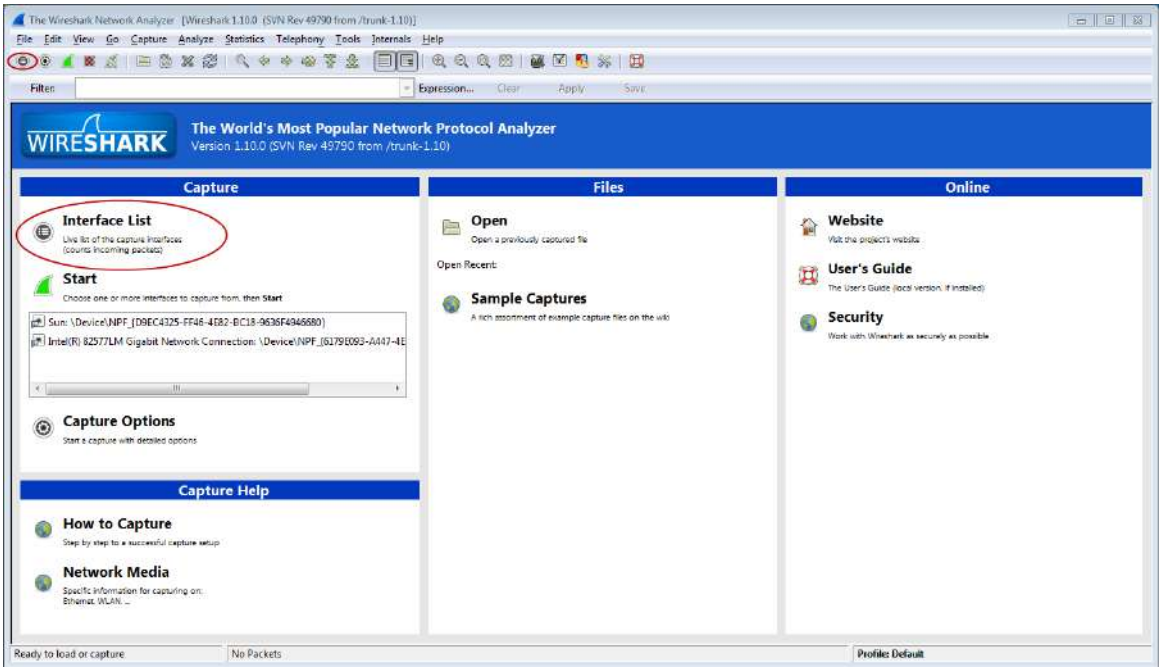


Рисунок 33 – Окно программы Wireshark

В окне ***Capture Interfaces*** (Перехват интерфейсов) установите флажок рядом с интерфейсом, подключённым к вашей локальной сети (рисунок 34).

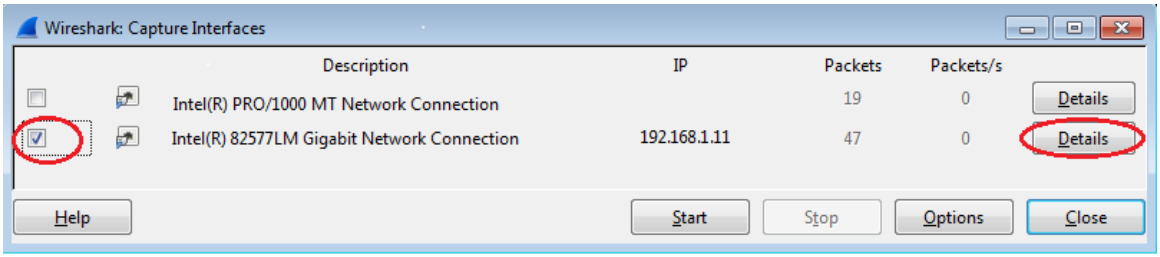


Рисунок 34 – Выбор интерфейсов

Если перечислено несколько интерфейсов и вы не уверены в том, какой из них нужно выбрать, нажмите кнопку ***Details*** (Подробнее) и откройте вкладку **802.3 (*Ethernet*).** Убедитесь в том, что *MAC*-адрес соответствует нужному интерфейсу (рисунок 35).

После этого нажмите кнопку ***Start*** (Начать), чтобы начать перехват данных (рисунок 36).

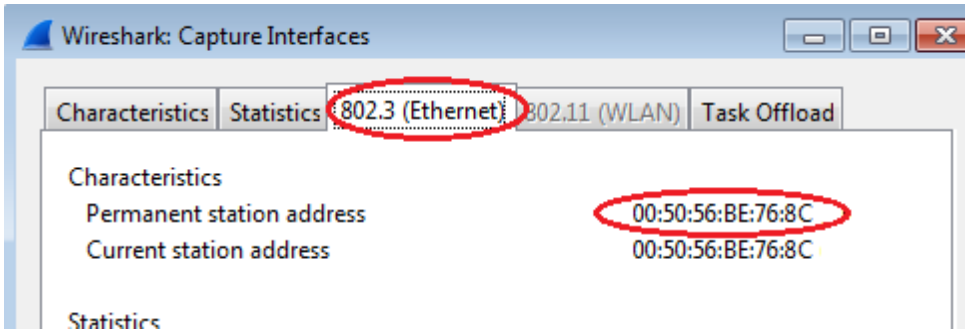


Рисунок 35 – Просмотр информации об интерфейсах

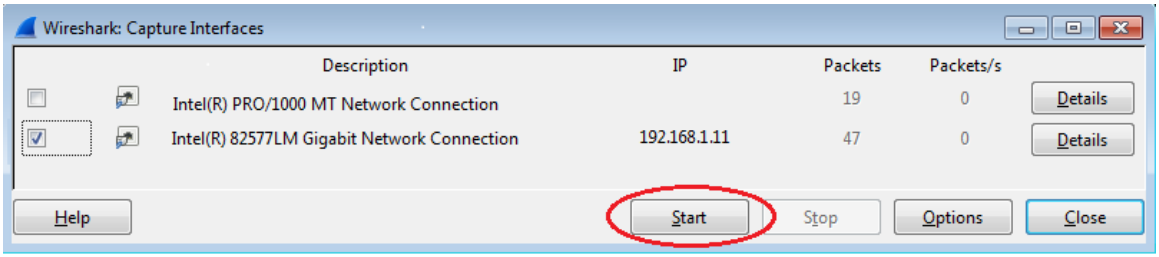


Рисунок 36 – Запуск захвата трафика

В верхней части окна программы *Wireshark* начнёт прокручиваться информация. Строки данных выделяются различными цветами в зависимости от протокола (рисунок 37).

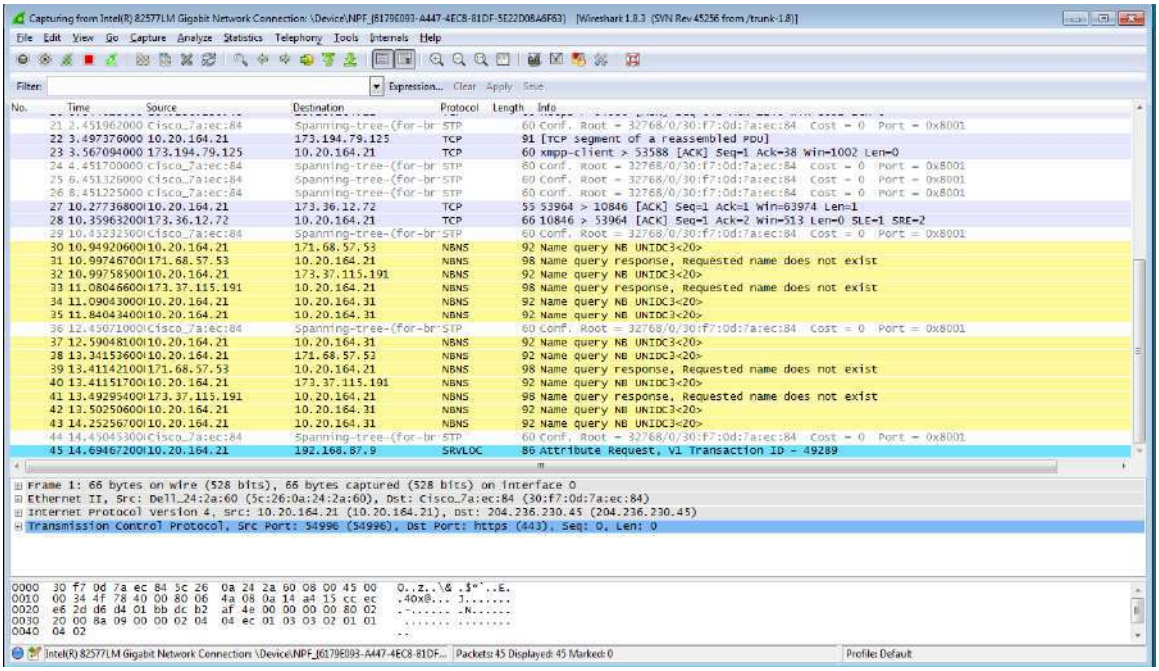


Рисунок 37 – Захваченные сетевые пакеты

Информация может прокручиваться очень быстро в зависимости от интенсивности сетевого обмена. Чтобы облегчить просмотр и работу с данными, собранными программой *Wireshark*, можно применить фильтр. Чтобы вывести на экран только протокольные блоки данных *ICMP* (эхо-запрос с помощью команды *ping*), в поле

фильтра в верхней части окна программы *Wireshark* (рисунок 38) введите ***icmp*** и нажмите клавишу ВВОД или кнопку ***Apply*** (Применить).

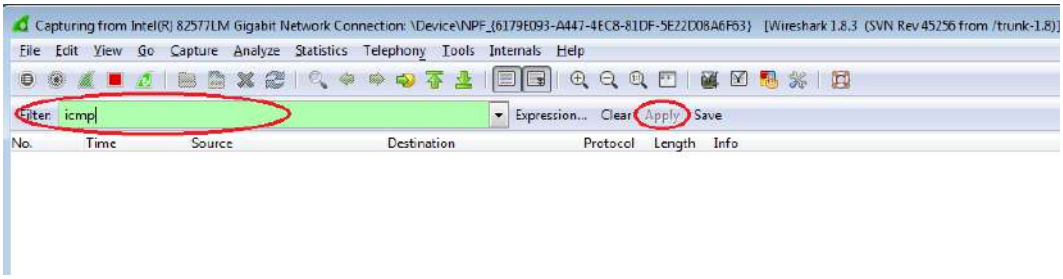


Рисунок 38 – Окно фильтрации

После этого все данные в верхнем окне исчезнут, однако перехват трафика в интерфейсе продолжится. Откройте окно командной строки и отправьте эхо-запрос с помощью команды *ping* на *IP*-адрес другого компьютера. В верхней части окна программы *Wireshark* появятся данные (рисунок 39).

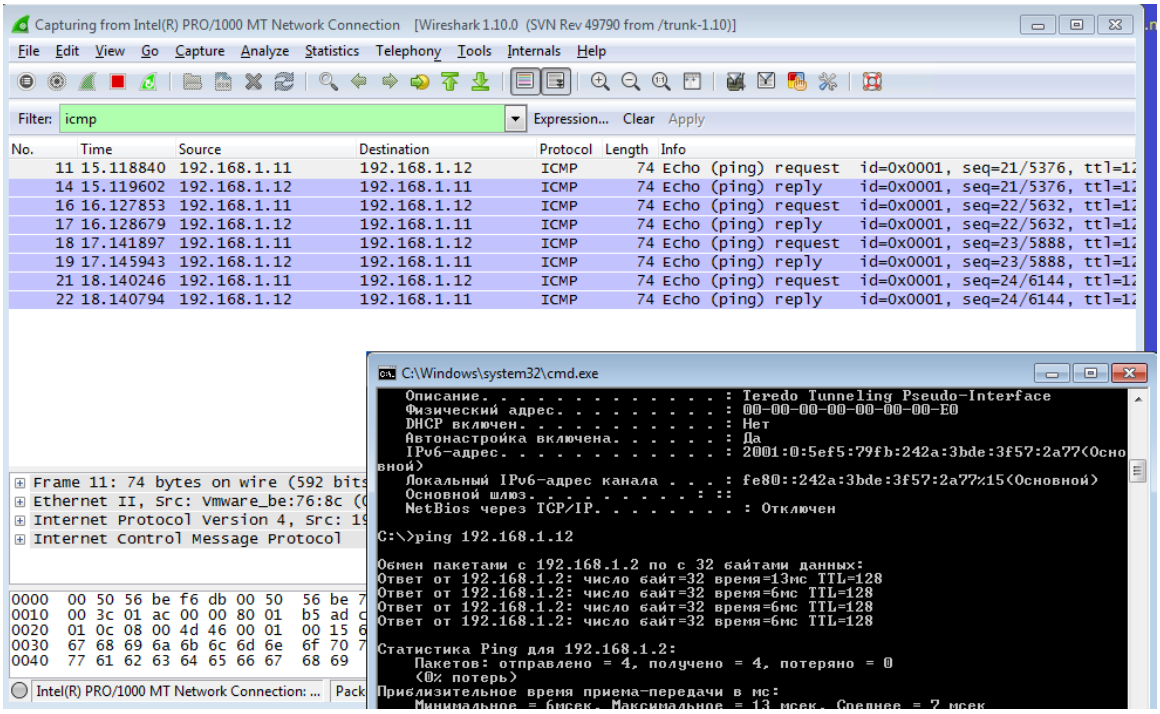


Рисунок 39 – Захваченные пакеты ICMP

Остановите перехват данных, нажав на значок ***Stop Capture*** (Остановить перехват), как показано на рисунке 40.

Изучим полученные пакеты протокола *ICMP*. Программа *Wireshark* отображает данные в трёх разделах (рисунок 41):

1) В верхнем разделе отображается список полученных кадров *PDU* со сводной информацией об *IP*-пакетах.

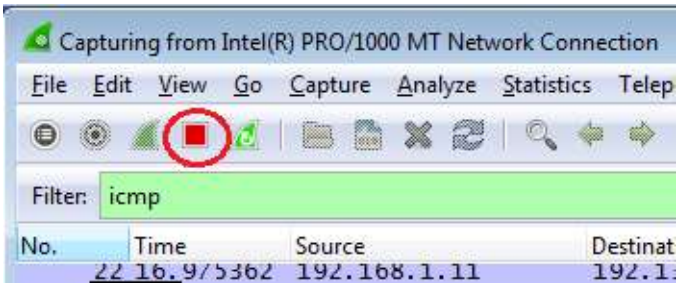


Рисунок 40 – Остановка захвата трафика

2) В среднем разделе приводится информация о *PDU* для кадра, выбранного в верхнем разделе экрана, и деление кадра *PDU* на слои протоколов.

3) В нижнем разделе показываются необработанные данные каждого уровня, как в шестнадцатеричном, так и десятичном форматах.

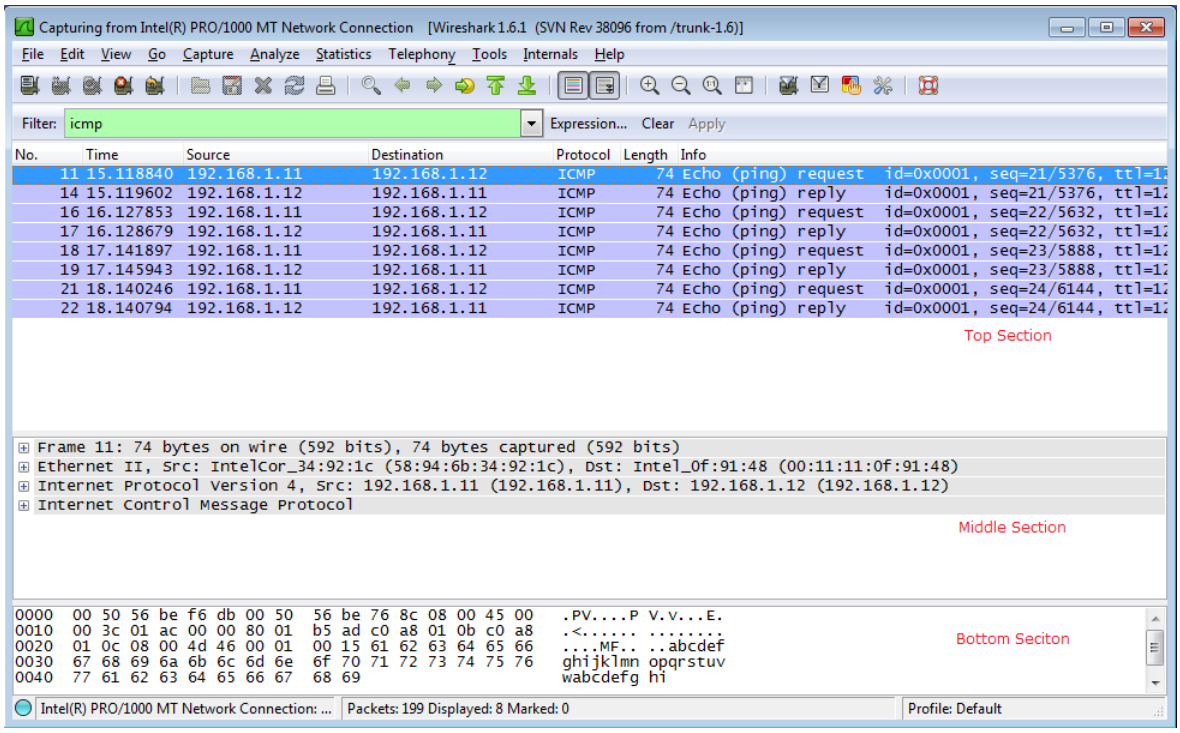


Рисунок 41 – Просмотр информации пакета

Выберите *PDU*-кадры первого запроса *ICMP* в верхнем разделе окна программы *Wireshark* (рисунок 42). В столбце *Source* (Источник) указывается *IP*-адрес вашего компьютера, а в столбце «*Destination*» (Назначение) – *IP*-адрес ПК, которому вы отправили эхо-запрос с помощью команды *ping*. Не меняя выбор *PDU*-кадра в верхнем разделе программы, перейдите в средний раздел. Нажмите на символ «+» слева от строки «*Ethernet* II», чтобы увидеть *MAC*-адреса источника и назначения (рисунок 43).

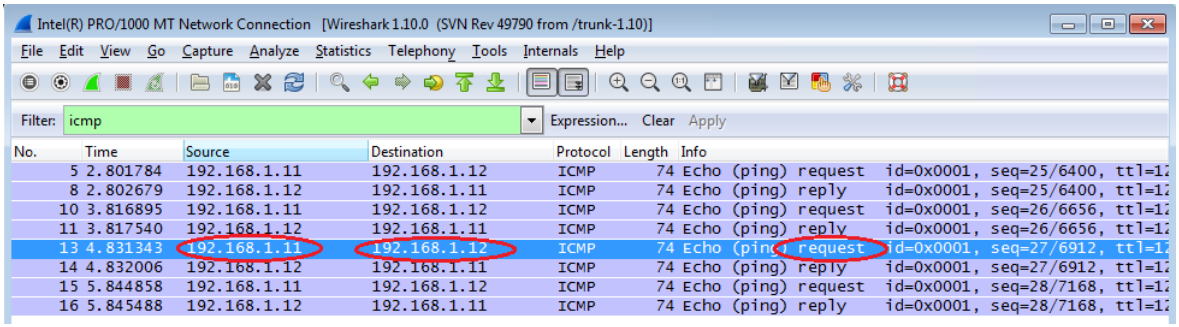


Рисунок 42 – Выбор нужного пакета *ICMP*

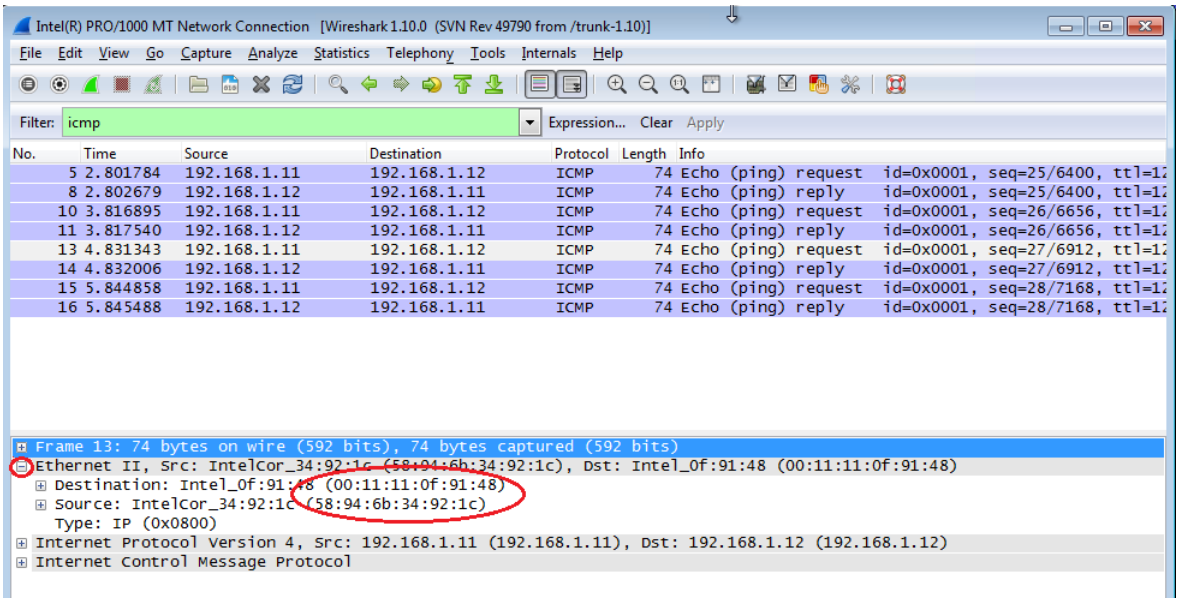


Рисунок 43 – Определение *MAC*-адресов

MAC-адрес источника должен совпадать с интерфейсом вашего компьютера, а *MAC*-адрес назначения – с *MAC*-адресом другого компьютера.

Как ваш ПК определил *MAC*-адрес ПК, на который был отправлен эхо-запрос с помощью команды *ping*?

Анализ обмена сообщениями *ARP*

Начните захват пакетов в программе *Wireshark*. С помощью фильтра отобразите только пакеты *ARP*.

Очистите *ARP*-кэш, набрав в командной строке команду:

```
arp -d *
```

Убедитесь в том, что *ARP*-кэш очищен:

```
arp -a
```

Отправьте эхо-запрос с помощью команды *ping* на шлюз по умолчанию с помощью команды:

```
ping 192.168.100.1.
```

После отправки эхо-запроса на шлюз по умолчанию остановите захват данных программой *Wireshark*.

В захваченных данных найдите сообщения *ARP* в панели сведений о пакетах. Найдите первый пакет, он является *ARP*-запросом (рисунок 44).

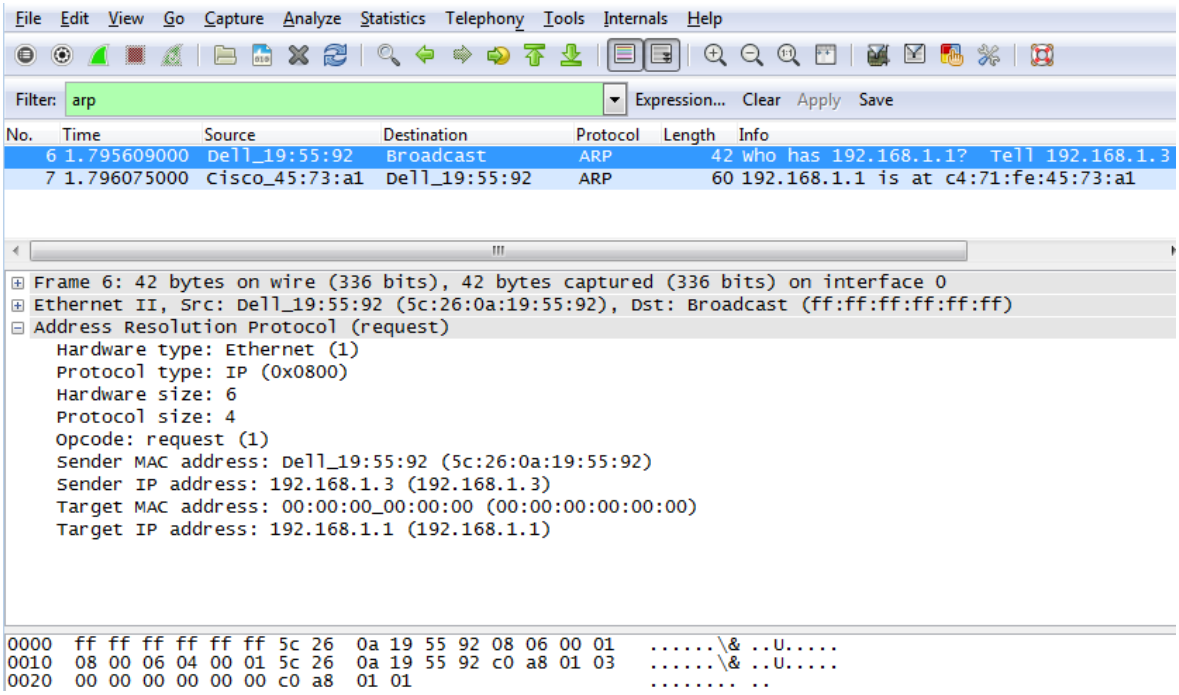


Рисунок 44 – Захват *ARP*-запроса

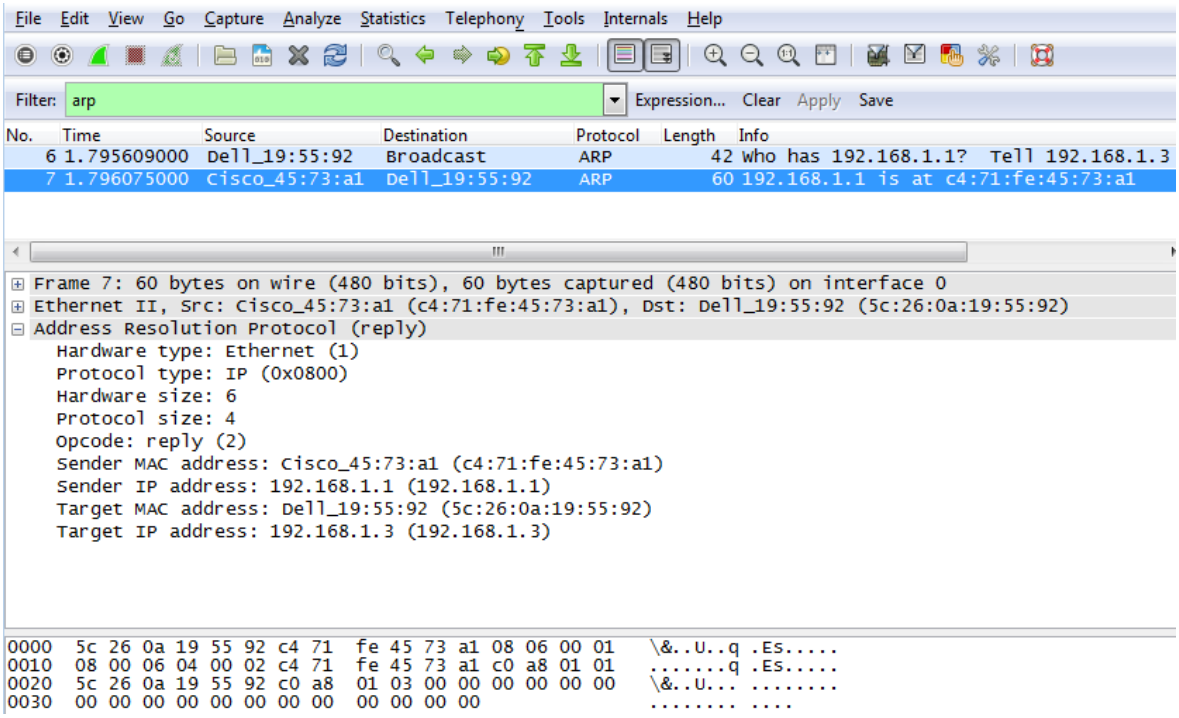


Рисунок 45 – Захват *ARP*-ответа

У *ARP*-запроса *MAC*-адрес отправителя является адресом вашего компьютера, а *MAC*-адрес получателя нулевой. Этот пакет является широковещательным, поскольку компьютер-отправитель не знает адреса получателя. Найдите второй пакет, он будет являться *ARP*-ответом (рисунок 45).

У *ARP*-ответа *MAC*-адресом отправителя является реальный адрес второго компьютера, а *MAC*-адресом получается – адрес первого компьютера, который отправлял *ARP*-запрос. Этот пакет уже не является широковещательным, он направляется первому компьютеру, который отправлял запрос.

Таким образом первый компьютер узнал *MAC*-адрес второго компьютера. Эти данные мы перехватили с помощью программы *Wireshark*.

Перехват установки сессии TCP

Запустите программу *Wireshark*. Выберите сетевой интерфейс, который будете использовать для захвата. Нажмите кнопку *Start* (Старт), чтобы начать захват данных.

Откройте в любом браузере веб-сайт, например *www.npi-tu.ru*. Сверните окно браузера и вернитесь в программу *Wireshark*. Остановите процесс захвата данных. Вы увидите захваченный трафик (рисунок 46). Найдите столбцы ***Source*** (Источник), ***Destination*** (Назначение) и ***Protocol*** (Протокол).

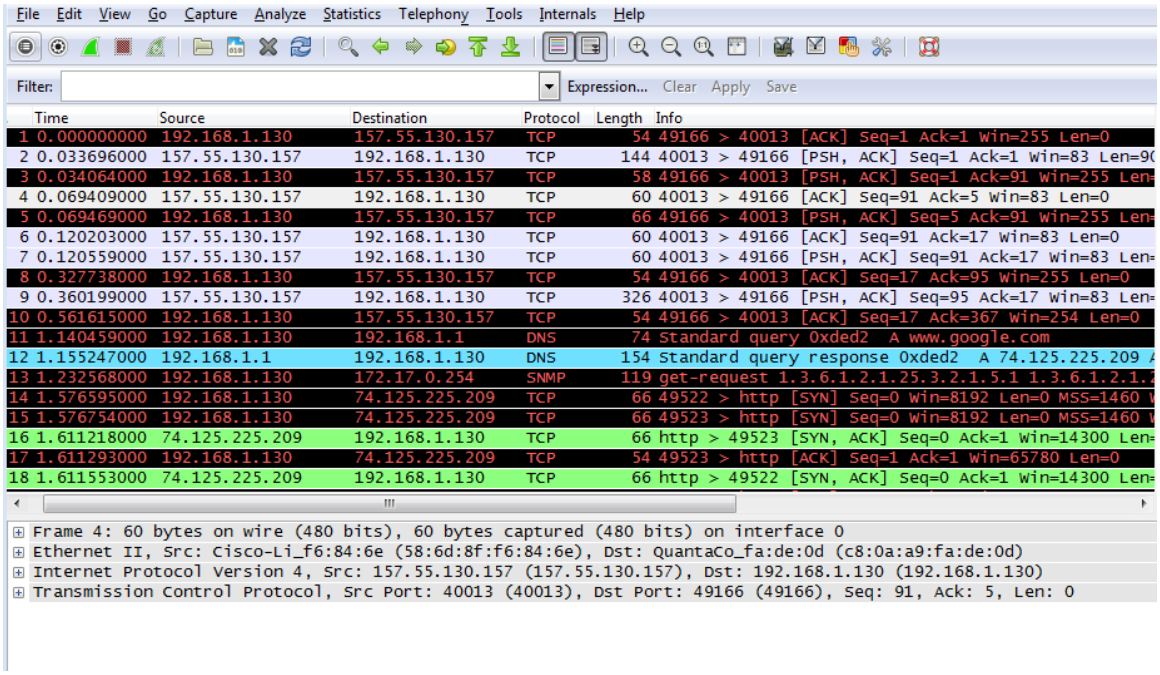


Рисунок 46 – Захват трафика при обращении к веб-сайту

В захваченных данных вы сможете увидеть весь процесс, включая протокол разрешения адресов (*ARP*), службу доменных имен (*DNS*) и трёхстороннее рукопожатие *TCP*, либо только часть этих данных.

Определите из рисунка 46 *IP*-адрес *DNS*-сервера, запрошенного компьютером.

Чтобы просмотреть только пакеты, связанные с *TCP*-соединением, воспользуйтесь фильтрами программы *Wireshark*. В поле фильтра программы *Wireshark* введите ***tcp*** и нажмите клавишу ВВОД (рисунок 47).

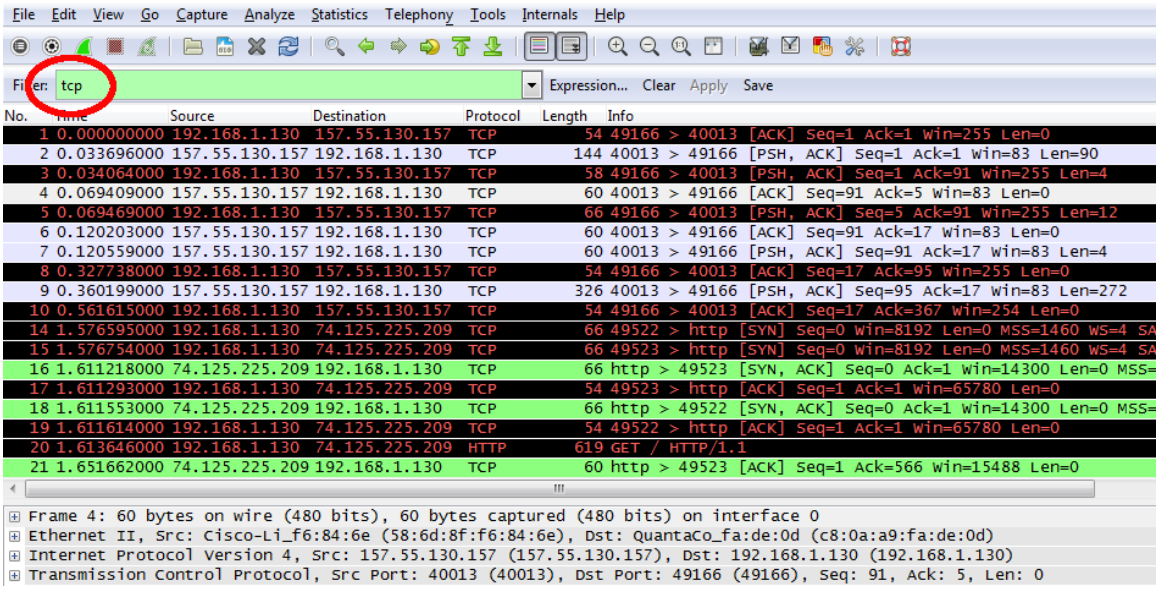


Рисунок 47 – Фильтрация протокола *TCP*

На панели списка пакетов (верхний раздел основного окна) выберите кадр, первый кадр должен иметь флаги [*SYN*] (рисунок 48). После этого будет выделена строка и отображена зашифрованная информация из пакета в двух нижних панелях. На панели нажмите на значок «+» слева от строки *Transmission Control Protocol* (Протокол управления передачей данных), чтобы увидеть подробную информацию о *TCP*. Слева от флажков нажмите на значок «+», чтобы увидеть подробную информацию о флагах *TCP*. Чтобы выбрать следующий кадр установки соединения, в меню программы *Wireshark* выберите параметр ***Go*** (Перейти), а затем ***Next Packet In Conversation*** (Следующий пакет коммуникации). Это ответ веб-сервера на исходный запрос начала сеанса (рисунок 49).

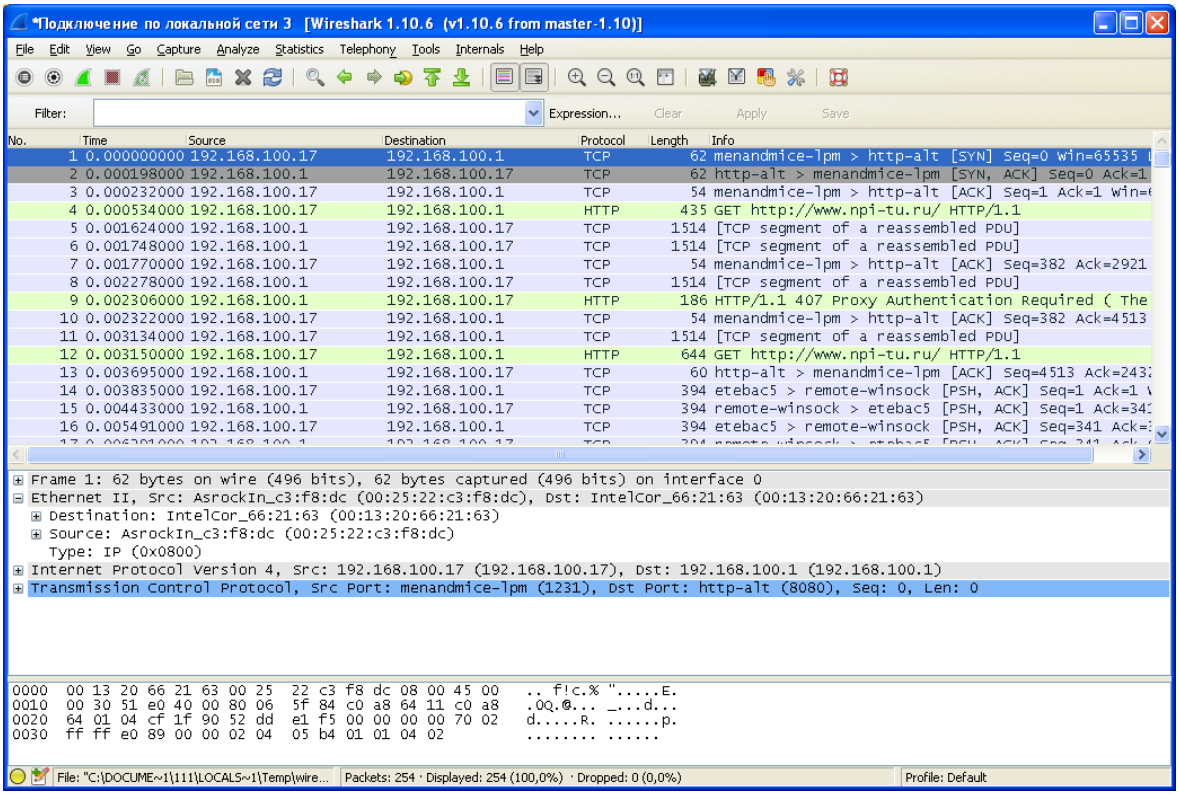


Рисунок 48 – Первый пакет установки связи *TCP*

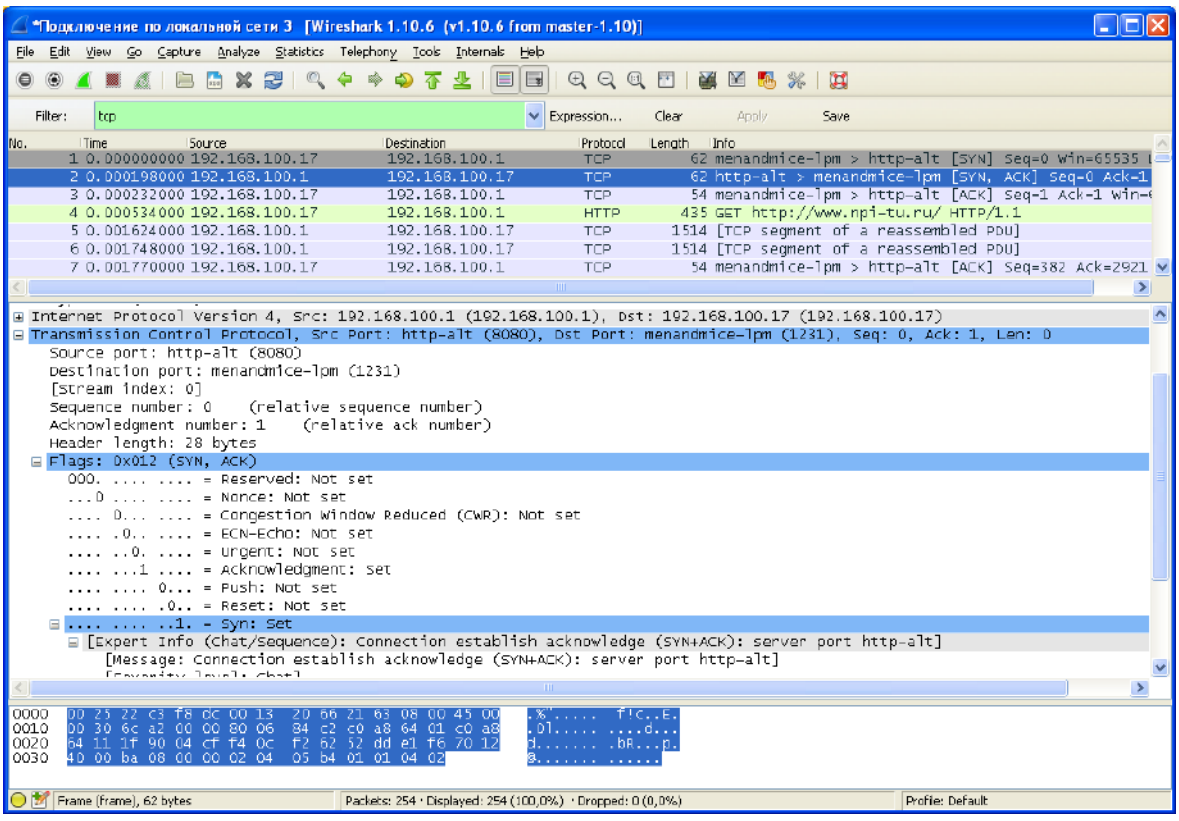


Рисунок 49 – Второй пакет установки связи *TCP*

Найдите и изучите третий пакет установки соединения. Какие установлены флаги *TCP*?

НАСТРОЙКА ПРОТОКОЛА МАРШРУТИЗАЦИИ *RIP*

Тестовая сеть

RIP – это один из самых распространенных и пользующихся широкой поддержкой протоколов маршрутизации в сетевой индустрии. Рассмотрим настройку *RIP* на примере небольшой сети, состоящей из трех подсетей. Топология примерной сети представлена на рисунке 50, а таблица адресации в таблице 6.

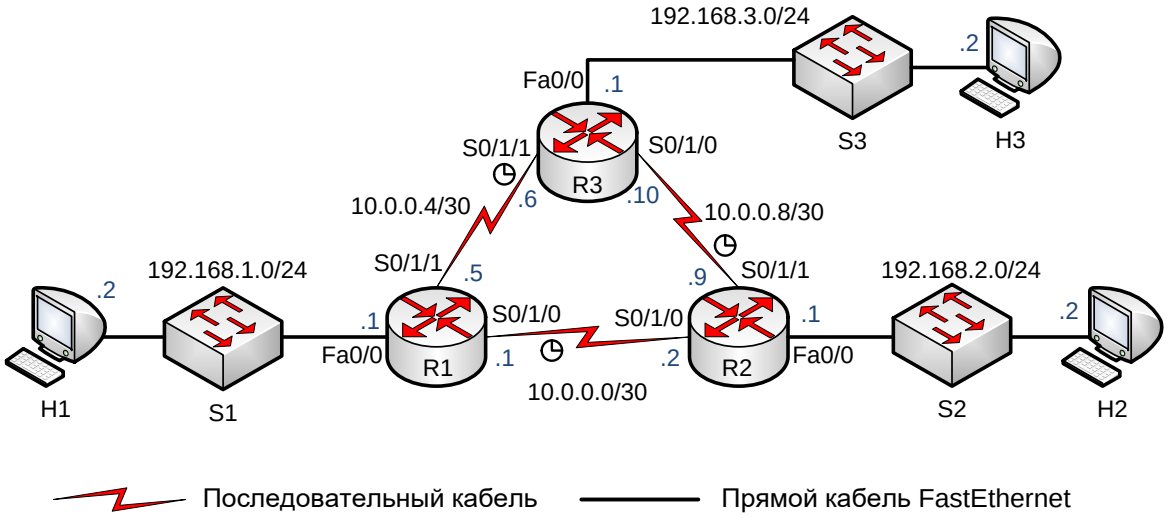


Рисунок 50 – Топология сети

Таблица 6 – Таблица адресации

Узел	Интерфейс	IP адрес	Маска	Шлюз
R1	FastEthernet0/0	192.168.1.1	255.255.255.0	–
	Serial0/1/0 (DCE)	10.0.0.1	255.255.255.252	–
	Serial0/1/1 (DTE)	10.0.0.5	255.255.255.252	–
R2	FastEthernet0/0	192.168.2.1	255.255.255.0	–
	Serial0/1/0 (DTE)	10.0.0.2	255.255.255.252	–
	Serial0/1/1 (DCE)	10.0.0.9	255.255.255.252	–
R3	FastEthernet0/0	192.168.3.1	255.255.255.0	–
	Serial0/1/0 (DTE)	10.0.0.10	255.255.255.252	–
	Serial0/1/1 (DCE)	10.0.0.6	255.255.255.252	–
H1	CiscoLAN	192.168.1.2	255.255.255.0	192.168.1.1
H2	CiscoLAN	192.168.2.2	255.255.255.0	192.168.2.1
H3	CiscoLAN	192.168.3.1	255.255.255.0	192.168.3.1

Подключите сетевые устройства и узлы, как показано на схеме топологии, используя последовательные кабели и прямые кабели

FastEthernet. Последовательные кабели подключаются соответствующими сторонами в зависимости от назначения интерфейса (*DCE*/*DTE*).

Подключите консольные кабели к консольным портам маршрутизаторов и ПК. Для доступа к консоли маршрутизатора будем использовать программу *Hyper Terminal*.

В программе *HyperTerminal* выбирите новое подключение, задаем любое имя, выбираем последовательный порт, скорость задаем 9600 бод, остальные параметры по умолчанию.

На узлах *H*1, *H*2, *H*3 настройте параметры *IP* протокола в соответствии с таблицей адресации.

Базовая настройка маршрутизатора

В этом параграфе описана последовательность действий по базовой настройке маршрутизатора *R*1. Для маршрутизаторов *R*2 и *R*3 действия аналогичны. Отличаются названия интерфейсов и *IP*-адреса, которые необходимо сверять с топологией и таблицей адресации.

В программе *HyperTerminal* для маршрутизатора переведите консоль в привилегированный режим, войдите в режим конфигурирования устройства и задайте ему имя:

```
Router>enable
Router#configure terminal
Router(config)#hostname R1
R1(config)#
```

Настройте интерфейс *FastEthernet* в соответствии с таблицей адресации:

```
R1(config)#interface FastEthernet 0/0
R1(config-if)#description R1 LAN Default Gateway
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
```

Настройте последовательные интерфейсы в соответствии с таблицей адресации:

```
R1(config)#interface Serial 0/1/0
R1(config-if)#description R1 to R2 link
R1(config-if)#ip address 10.0.0.1 255.255.255.252
```

Команда *clock rate* 64000 подается только для интерфейсов, помеченных как *DCE*:

```
R1(config-if)#clock rate 64000
R1(config-if)#no shutdown
R1(config-if)#exit
```

```
R1(config)#interface Serial 0/1/1
R1(config-if)#description R1 to R3 link
R1(config-if)#ip address 10.0.0.5 255.255.255.252
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#exit
```

Проверьте правильность настройка вашей подсети из общей топологии. Откройте на узле *H*1 окно командной строки *Windows* (Пуск -> Выполнить -> *cmd*). Выполните команду:

```
C:\>ping 192.168.1.1
```

Проверка достижимости маршрутизатора из узла должна быть успешной.

Проверка общей связности в сети

Проверьте правильность настройки интерфейсов на маршрутизаторе. В консоли маршрутизатора из привилегированного режима подайте команду:

```
R1#show ip interfaces brief
```

Изучите вывод, который показывает маршрутизатор. Сколько интерфейсов всего выводится? Сколько из них активны и правильно работают (статус *up* и протокол *up*)? Правильные ли *IP* адреса у всех интерфейсов?

Проверьте связь с другими маршрутизаторами в топологии. В консоли маршрутизатора *R*1 из привилегированного режима для проверки связи с маршрутизатором *R*2 подайте команду:

```
R1#ping 10.0.0.2
```

Для проверки связи с маршрутизатором *R*3:

```
R1#ping 10.0.0.6
```

Проверка достижимости должна быть успешной.

Посмотрите таблицу маршрутизации вашего маршрутизатора:

```
R1#show ip route
```

Сейчас в таблице маршрутизации три сети и все они непосредственно подключены к маршрутизатору.

Проверьте связи между узлами сети. В командной строке узла H1 для проверки связи с узлом *H*2 выполните команду:

```
C:\>ping 192.168.2.2
```

Для проверки связи с узлом *H*3 выполните команду:

```
C:\>ping 192.168.3.2
```

Эти проверки связи являются безуспешными, поскольку в таб-

лице маршрутизации нет сведений о пути до сетей этих узлов. Решим эту проблему путем настройки динамической маршрутизации по протоколу *RIP*.

Настройка динамической маршрутизации

Включите протокол динамической маршрутизации *RIP*. Для этого зайдите в режим конфигурирования маршрутизатора и подайте команды:

```
R1(config)#router rip
R1(config)#version 2
R1(config)#no auto-summary
R1(config)#network 10.0.0.0
R1(config)#network 192.168.1.0
```

Подождите 30 секунд. Посмотрите таблицу маршрутизации маршрутизатора:

```
R1#show ip route
```

В таблицу добавились сети по протоколу *RIP* (обозначены буквой *R*).

Проверьте связи между узлами сети. В командной строке узла *H*1 для проверки связи с узлом *H*2 выполните команду:

```
C:\>ping 192.168.2.2
```

Для проверки связи с узлом *H*3 выполните команду:

```
C:\>ping 192.168.3.2
```

Теперь проверки связи между узлами сети являются успешными, потому что маршрутизаторы знаю путь до сетей этих узлов.

Опишите путь пакета от узла *H*1 до узла *H*2. Воспользуйтесь программой *tracert.exe* для определения пути.

ПЛАНИРОВАНИЕ АДРЕСАЦИЯ *IPV4* В КОМПЬЮТЕРНЫХ СЕТЯХ

Двоичные побитовые операции над *IP*-адресами

Планирование сетевой адресации является достаточно простым процессом, однако понимание этого процесса у многих начинающих администраторов вызывает проблемы, в основном, из-за малого опыта выполнения булевых операций над 32-битными двоичными числами. Данная глава содержит множество упражнений, призванных дать опыт, необходимый для усвоения процесса планирования сетевой адресации и начинается с параграфа, посвященному самым базовым операциям над *IP*-адресами.

Заполните таблицу 7, преобразовав десятичное число в 8-битное двоичное значение. Первое число уже преобразовано для примера. Помните, что восемь двоичных битовых значений в октете имеют основание 2 и слева направо выглядят как 128, 64, 32, 16, 8, 4, 2 и 1.

Таблица 7 – Перевод чисел в двоичную систему исчисления

Десятичное	Двоичное
192	11000000
168	
10	
255	
2	

*IPv*4-адрес преобразуются в двоичный формат точно так же, как было проделано выше, но состоит из четырех октетов (байт).

Заполните таблицу 8 двоичными эквивалентами указанных *IP*-адресов.

Для того, чтобы определить сетевой адрес для заданного *IP*-адреса, сначала вам необходимо перевести десятичный *IPv*4-адрес и маску подсети в их двоичный эквивалент. Затем применяется операция двоичного сложения «И» для *IP*-адреса и маски. Полученное 32-битное значение представляет собой сетевой адрес.

Таблица 8 – Перевод *IP*-адресов в двоичную систему исчисления

Десятичное	Двоичное
192.168.10.10	11000000.10101000.00001010.00001010
209.165.200.229	
172.16.18.183	
10.86.252.17	
255.255.255.128	
255.255.192.0	

При использовании операции «И» десятичное значение в каждой битовой позиции 32-битного *IP*-адреса узла сравнивается с соответствующей позицией в 32-битной маске подсети. При наличии двух нулей или 0 и 1 результатом операции «И» будет 0. При наличии двух единиц результатом будет 1, как показано в таблице 9.

Таблица 9 – Определение адреса сети

Описание	Десятичное	Двоичное
IP-адрес	192.168.10.131	11000000.10101000.00001010.10000011
Маска подсети	255.255.255.192	11111111.11111111.11111111.11000000
Сетевой адрес	192.168.10.128	11000000.10101000.00001010.10000000

Заполните таблицы 10, 11, 12, 13, 14.

Для того, чтобы определить, находятся ли два *IP*-адреса в одной сети, необходимо выполнить сравнение сетевых адресов для обоих *IP*-адресов. Сетевые адреса получаются применением операции «И» к *IP*-адресу и маске. Если у обоих *IP*-адресов одинаковый сетевой адрес, то они находятся в одной сети и могут непосредственно взаимодействовать, отправляя пакеты друг другу напрямую. Если же сетевые адреса не совпадают, то компьютеры находятся в разных сетях, для их взаимодействия требуется маршрутизирующее устройство, и пакеты необходимо отправлять на адрес основного шлюза ПК.

Таблица 10 – Задание по определению адреса сети, вариант А

Описание	Десятичное	Двоичное
IP-адрес	172.16.145.29	
Маска подсети	255.255.0.0	
Сетевой адрес		

Таблица 11 – Задание по определению адреса сети, вариант Б

Описание	Десятичное	Двоичное
IP-адрес	192.168.10.10	
Маска подсети	255.255.255.0	
Сетевой адрес		

Таблица 12 – Задание по определению адреса сети, вариант В

Описание	Десятичное	Двоичное
IP-адрес	192.168.68.210	
Маска подсети	255.255.255.128	
Сетевой адрес		

Таблица 13 – Задание по определению адреса сети, вариант Г

Описание	Десятичное	Двоичное
IP-адрес	172.16.188.15	
Маска подсети	255.255.240.0	
Сетевой адрес		

Таблица 14 – Задание по определению адреса сети, вариант Д

Описание	Десятичное	Двоичное
IP-адрес	10.172.2.8	
Маска подсети	255.224.0.0	
Сетевой адрес		

Компьютеру ПК-А присвоен *IP*-адрес 192.168.1.18, а компьютеру ПК-Б — *IP*-адрес 192.168.1.33. Маска подсети обоих компьютеров — 255.255.255.240. Какой сетевой адрес у ПК-А? Какой

сетевой адрес у ПК-Б? Смогут ли эти ПК взаимодействовать друг с другом напрямую?

Компьютеру ПК-А присвоен *IP*-адрес 10.0.0.16, а компьютеру ПК-Б — *IP*-адрес 10.1.14.68. Маска подсети обоих компьютеров — 255.254.0.0. Какой сетевой адрес у ПК-А? Какой сетевой адрес у ПК-Б? Смогут ли эти ПК взаимодействовать друг с другом напрямую? Какой наименьший адрес, присвоенный компьютеру ПК-Б, позволит ему находиться в одной сети с ПК-А?

Вычисление адресов для заданной сети

Маска сети определяет, какие биты *IP*-адреса относятся к адресу сети, а какие – к адресу узла внутри сети. Число битов в узловой части адреса сети определяет количество различных уникальных адресов в этой сети и широковещательный адрес.

В маске сети узловые биты всегда следуют за сетевыми, поэтому очень часто для сокращения вместо полной маски подсети указывают число сетевых бит. Например, для сети 192.168.10.0 с маской 255.255.255.0 можно указать 192.168.10.0/24.

IP-адреса внутри сети распределяются следующим образом, как представлено в таблице 15.

Таблица 15 – Специальные значения *IP*-адресов

Адрес	Биты	Назначение
Сетевой	000…000 (все нулевые)	Адрес сети, идентифицирует саму сеть и не выдается узлам.
Широковещательный	111…111 (все единичные)	Адресует все узлы внутри сети, используется для передачи информации сразу для всей сети.
Уникальные	xxx…xxx (все прочие)	Арес для отдельного узла в сети, должны быть уникальными.

Рассчитать количество узлов для каждой сети можно путём анализа маски подсети. Число отдельных IP-адресов можно определить по формуле:

$$\{\text{количество узлов}\} = 2^{\{\text{количество бит узла}\}} - 2$$

Маска подсети может быть представлена в десятичном формате с точкой-разделителем, например 255.255.192.0, или в формате сетевого префикса, например /18. *IPv*4-адрес всегда содержит 32 бита. Отняв количество битов, используемых сетевой частью (как показано в маске подсети), вы получите количество битов, используемых для узлов.

Например, маска подсети 255.255.192.0 равна /18 в префиксной записи. Вычитание 18 бит сети из 32 бит даст нам 14 бит, оставшихся для узловой части. Исходя из этого, можно выполнить простой расчёт:

$$2^{14} = 16\ 384 - 2 = 16\ 382 \text{ узла}$$

Проанализируйте таблицу 16 и определите сетевую и узловую части указанных *IPv*4-адресов. Первые две строки содержат примеры заполнения таблицы. Сокращения, используемые в таблице:

С = все 8 бит для октета содержатся в сетевой части адреса;

с = бит в сетевой части адреса;

У = все 8 бит для октета содержатся в узловой части адреса;

у = бит в узловой части адреса.

Таблица 16 – Определение сетевой и узловой части

IP-адрес/префикс	**Сеть/узел** С, с = сеть У, у = узел	**Маска подсети**	**Сетевой адрес**
192.168.10.10/24	С.С.С.У	255.255.255.0	192.168.10.0
10.101.99.17/23	С.С.ссссссу.У	255.255.254.0	10.101.98.0
209.165.200.227/27			
172.31.45.252/24			
10.1.8.200/26			
172.16.117.77/20			
10.1.1.101/25			
209.165.202.140/27			
192.168.28.45/28			

Проанализируйте таблицу 17 и укажите диапазон адресов узлов и широковещательных адресов. В первой строке приведён пример завершения таблицы.

Таблица 17 – Определение диапазона IP-адресов

Адрес IPv4/ префикс	Сетевой адрес	Адрес первого узла	Адрес последнего узла	Широковеща-тельный адрес	Nбп	Nбу	N
192.168.10.10/24	192.168.10.0	192.168.10.1	192.168.10.254	192.168.10.255	24	8	254
10.101.99.17/23							
209.165.200.227/27							
172.31.45.252/24							
10.1.8.200/26							
172.16.117.77/20							
10.1.1.101/25							
209.165.202.140/27							
192.168.28.45/28							
192.168.100.25/28							
172.30.10.130/30							
10.1.113.75/19							
198.133.219.250/24							
128.107.14.191/22							
172.16.104.99/27							

Классификация *IPv*4-адресов

Все множество существующих *IPv*4 адресов разбито на несколько классов.

Отдельно выделяются адреса кольцевого интерфейса, которые относятся к сети 127.0.0.0/8. Эти адреса предназначены для различных сервисных функций стека *TCP*/*IP*. Наиболее известен и часто используется адрес 127.0.0.1, который присутствует на каждом компьютере с установленным и правильно работающим стеком *TCP*/*IP*. Все адреса кольцевого интерфейса не адресуются в Интернет и не могут использоваться для передачи данных по сети.

Еще одним важным классом адресов являются частные адреса, которые предназначены для использования в частных локальных сетях. Эти адреса могут присваиваться узлам в локальной сети, но не могут использоваться с сети Интернет. Для обеспечения доступа к узлам из такой локальной сети с частными адресами в Интернет используются механизмы трансляции (преобразования, подмены) адресов – *NAT*. Частные адреса принадлежат диапазонам, представленным в таблице 18.

Определенный диапазон адресов в Интернет предназначен для многоадресной рассылки: 224.0.0.0 – 239.255.255.255. Многоадресной называют рассылку, получателями которой является несколько (от одного и более) узлов в сети Интернет. Любой узел

может подключиться к многоадресной рассылке с определенным адресом и получать ее пакеты, может отключиться в любой момент. Эти механизмы регулируются специальным протоколом *IGMP*. Адрес из многоадресного диапазона не может быть присвоен отдельному компьютеру как его *IP*-адрес.

Таблица 18 – Диапазоны частных *IP*-адресов

Название	**Значения**
Диапазон класса *A*	10.0.0.0 – 10.255.255.255
Диапазон класса *B*	172.16.0.0 – 172.31.255.255
Диапазон класса *C*	192.168.0.0 – 192.168.255.255

В диапазоне *IP*-адресов выделен специальный блок для экспериментальной исследовательской работы и тестирование сетей: 240.0.0.0 – 255.255.255.254. Эти адреса обычно не присваиваются узлам в работающих штатно сетях и могут не маршрутизироваться в сети Интернет, но их можно присвоить любому узлу.

В каждой сети есть отдельный адрес, называемый широковещательным (все узловые биты которого равны 1). Кроме этого, существует глобальный широковещательный адрес – 255.255.255.255. Получателями такого адреса теоретически могли бы быть все узлы в сети Интернет, но в целях безопасности и защиты от перегрузки в глобальной сети, областью действия такого адреса обычно является только сеть, в которой он был отправлен. То есть он мало отличается от широковещательного адреса самой сети. В своих целях администраторы могут организовывать передачу пакетов глобального широковещательного адреса за пределы одного сетевого адреса, но в сети Интернет передача такого адреса запрещена.

Все прочие *IP*-адреса называют общими, они могут работать в сети Интернет без ограничений.

Таким образом, допустимыми для присвоения отдельному узлу являются адреса, которые не являются сетевыми, широковещательными или многоадресными, адресами кольцевого интерфейса.

Таблица 19 – Определение типа *IP*-адреса

IP-адрес	Маска подсети	Тип адреса
10.1.1.1	255.255.255.252	узел
192.168.33.63	255.255.255.192	
239.192.1.100	255.252.0.0	
172.25.12.52	255.255.255.0	
10.255.0.0	255.0.0.0	
172.16.128.48	255.255.255.240	
209.165.202.159	255.255.255.224	
172.16.0.255	255.255.0.0	
224.10.1.11	255.255.255.0	

Таблица 20 – Определение частных *IP*-адресов

IP-адрес/префикс	Общий или частный
209.165.201.30/27	
192.168.255.253/24	
10.100.11.103/16	
172.30.1.100/28	
192.31.7.11/24	
172.20.18.150/22	
128.107.10.1/16	
192.135.250.10/24	
64.104.0.11/16	

Проанализируйте таблицу 19 и определите тип адреса (адрес сети, узла, многоадресной или широковещательной рассылки). В первой строке приведён пример завершения таблицы.

Проанализируйте таблицу 20 и определите тип адреса – общий или частный.

Проанализируйте таблицу 21 и определите, является ли пара адреса и префикса допустимым адресом узла.

Таблица 21 – Определение допустимых *IP*-адресов

IP-адрес/префикс	**Допустимый или нет адрес узла**	**Причина**
127.1.0.10/24		
172.16.255.0/16		
241.19.10.100/24		
192.168.0.254/24		
192.31.7.255/24		
64.102.255.255/14		
224.0.0.5/16		
10.0.255.255/8		
198.133.219.8/24		

Определение количества подсетей

Компьютерную сеть, состоящую из множества узлов, можно разбивать на подсети. Делается это по множеству причин, самыми главными из которых являются:

- из соображений безопасности – для каждого отдела организации можно выделить отдельную подсеть и ограничить доступ этих подсетей друг к другу во избежание утечки конфиденциальной информации или распространения вредоносных программ;
- из соображений производительности – каждая сеть для своей работы использует множество механизмов, основанных на использовании широковещательных пакетов. Чем больше число узлов в сети, тем больше таких пакетов. В сети размером в несколько сотен узлов приличная часть пропускной способности каналов связи занята передачей этих служебных широковещательных пакетов, уменьшая при этом полезную работу сети по передаче пользовательских данных. В масштабах всего Интернет использовать широковещательные пакеты уже попросту невозможно.

Понятие сети и подсети близки друг другу. Когда говорят о крупной сети, то *подсетью* называют какую-либо ее обособленную часть. Упоминается также понятие *суперсети*, которое используют при объединении сетей со смежными сетевыми адресами с уменьшением сетевой маски (для облегчения задачи маршрутизации уменьшением строк в таблице маршрутов устройств).

Отдельной подсетью считается сеть, в которой все узлы имеют одинаковый сетевой адрес и маску. Они могут без ограничений связываться друг с другом напрямую и посылать широковещательные пакеты. Совокупность таких узлов называют *доменом широковещательной рассылки*.

Узлы, которые имеют различные сетевые адреса, уже не являются частью одной подсети и не могут напрямую взаимодействовать друг с другом. Для связи узлов из двух различных подсетей используется специальное сетевой устройство – маршрутизатор, основными задачами которого как раз и являются связь различных подсетей и определение направления передачи пакета данных, исходя из сетевого адреса получателя этого пакета.

В крупной корпоративной сети, состоящей из множества маршрутизаторов, каждая подсеть отделена от другой маршрутизатором. Подсчитать число подсетей можно, определив, сколько подсетей в топологии разделены друг от друга маршрутизатором.

При определении сетевой адресации также важно определить, сколько узлов находится в каждой подсети, чтобы найти для этой подсети оптимальную маску. Количество узлов можно подсчитать, при этом необходимо помнить:

- каждый узел требует один *IP* адрес;
- концентраторы не требуют *IP* адресов, коммутаторы, как правило, их не требуют, но им может быть присвоен один *IP* адрес (*Vlan* 1), так что его тоже необходимо учитывать;
- интерфейс (порт) маршрутизатора, к которому подключена подсеть требует один *IP* адрес, который одновременно является адресом шлюза по умолчанию (*default gateway*) для этой подсети.

Зная количество узлов, можно определить, какая сетевая маска необходима этой подсети. Число *IP*-адресов определяется по формуле:

$$\{\text{количество бит}\} = \lceil \log_2(\{\text{количество узлов}\} + 1) \rceil$$

Операция $\lceil x \rceil$ означает округление числа x до целого вверх.

То есть необходимо найти такое число, чтобы 2 в степени этого числа было не меньше требуемого количества узлов плюс 2.

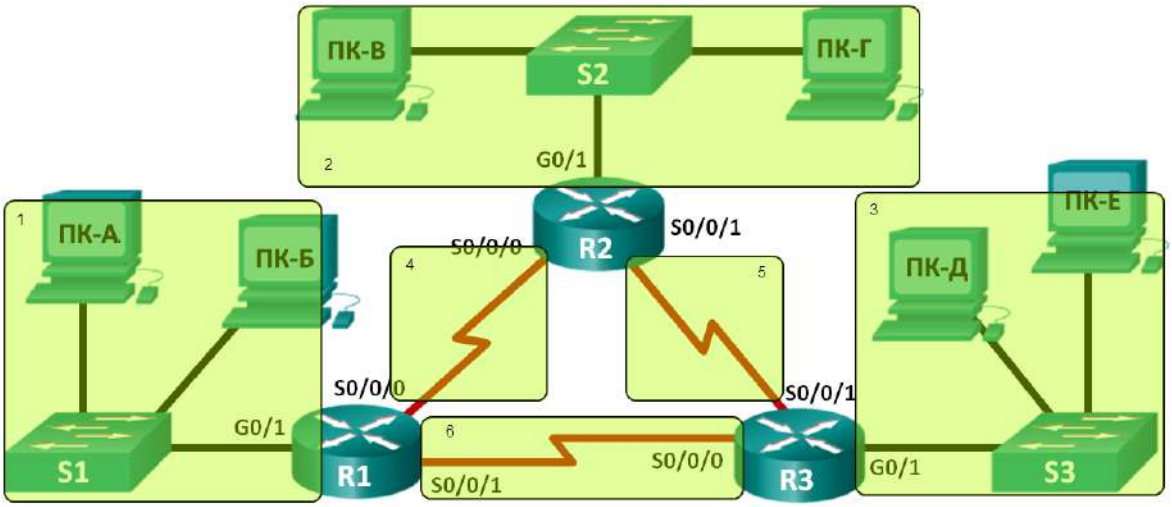


Рисунок 51 – Определение количества подсетей

Таблица 22 – Вычисление количества бит подсети

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0	S1, ПК-А, ПК-Б, R1(G0/1)	4	3
1	S2, ПК-В, ПК-Г, R2(G0/1)	4	3
2	S3, ПК-Д, ПК-Е, R3(G0/1)	4	3
3	R1(S0/0/0), R2(S0/0/0)	2	2
4	R2(S0/0/1), R3(S0/0/1)	2	2
5	R1(S0/0/1), R3 (S0/0/0)	2	2

На рисунке 51 показано разбиение сети на шесть подсетей. Подсети пронумерованы и выделены цветом. В таблице 22 представлен пример расчета количества бит, необходимых для сетевой части каждой подсети.

Определите количество подсетей в топологии на рисунке 52 и заполните таблицу 23.

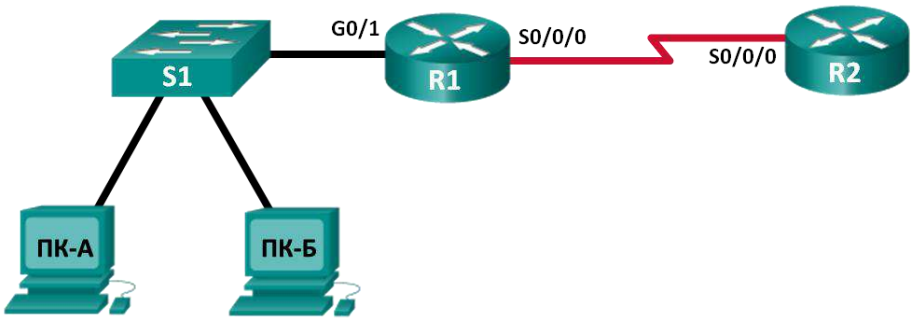


Рисунок 52 – Определение количества подсетей, вариант А

Таблица 23 – Вычисление количества бит подсети, вариант А

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			

Определите количество подсетей в топологии на рисунке 53 и заполните таблицу 24.

Определите количество подсетей в топологии на рисунке 54 и заполните таблицу 25.

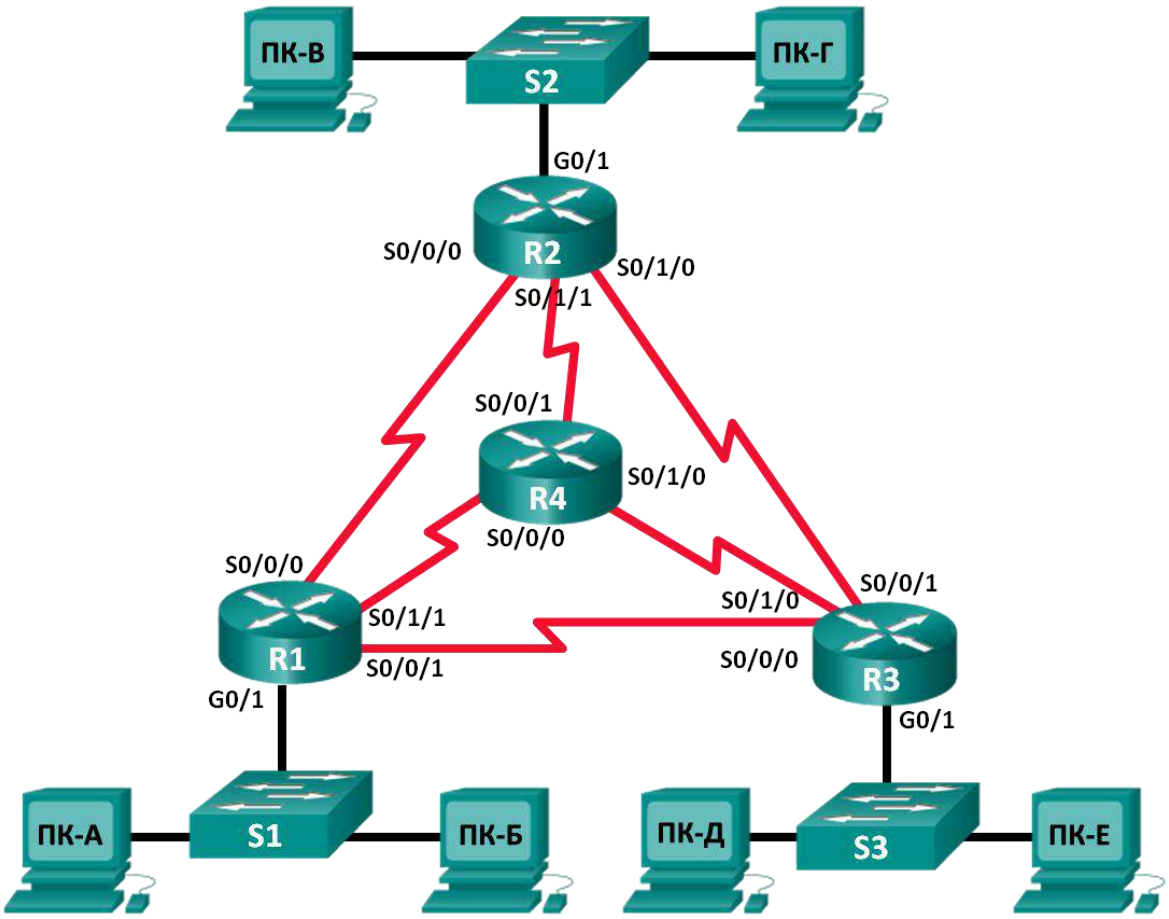


Рисунок 53 – Определение количества подсетей, вариант Б

Таблица 24 – Вычисление количества бит подсети, вариант Б

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

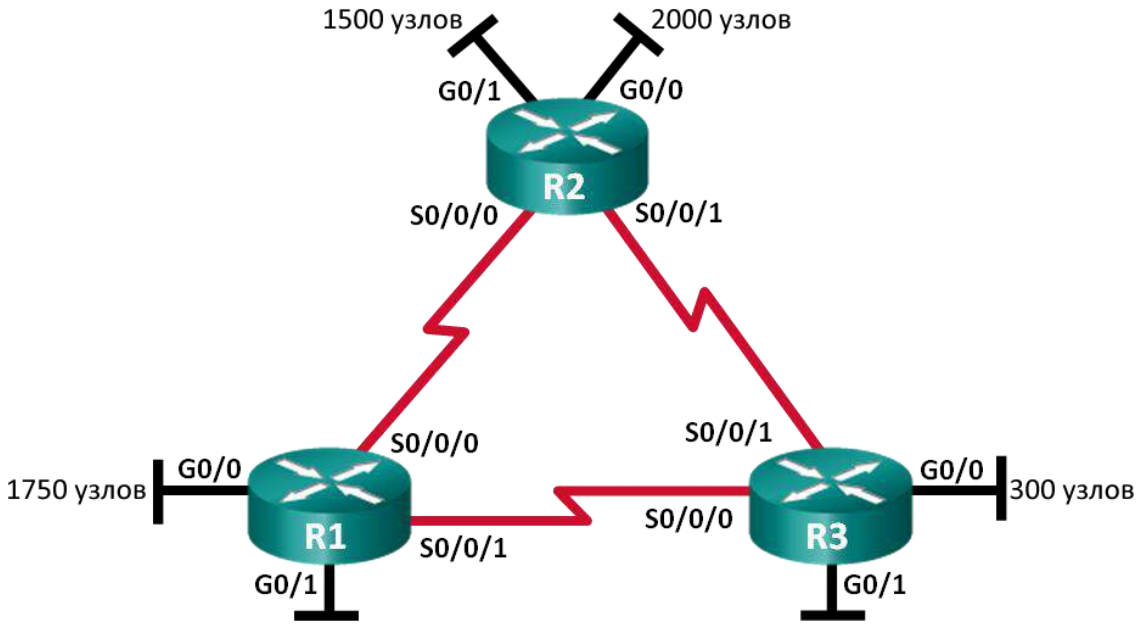


Рисунок 54 – Определение количества подсетей, вариант В

Таблица 25 – Вычисление количества бит подсети, вариант В

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

Разбиение сети на подсети

Одной из часто решаемых задач при проектировании компьютерной сети является разбиение сети на подсети и выделение диапазонов сетевых адресов для соответствующих подсетей.

Обычно проектировщику сети дается один диапазон адресов, либо сеть, заданная в виде адрес/маска, на всю проектируемую сеть. Его задачей является определить количество подсетей в сети, количество узлов в каждой подсети, а затем, исходя из этих данных, спланировать разбиение сети на подсети.

При разбиении маска каждой подсети становится больше, чем исходная маска всей сети. Число бит, на которые увеличивается маска, зависит от числа подсетей, и определяется по формуле:

$$\{\text{количество добавочных бит}\} = \lceil \log_2\{\text{количество подсетей}\} \rceil$$

То есть необходимо найти такое число, чтобы 2 в степени этого числа было не меньше требуемого количества подсетей.

Например, дана сеть 172.16.0.0/16. При анализе топологии сети было определено, что она состоит из 12 подсетей. По формуле для разбиения сети на подсети необходимо 4 добавочных бита.

Добавочные биты увеличивают маску в подсети, при этом уменьшают на то же число количество узловых бит. Из-за этого максимально допустимое число узлов в подсети снижается. Поэтому необходимо убедиться, что получаемое разбиение на подсети позволяет оставить в каждой подсети требуемое число узлов.

Такой подход к разбиению сети на подсети называется *разбиением с фиксированной маской*, потому что во всех подсетях при нем получаются маски одинаковой длины, и, следовательно, одинаковое число максимально допустимых узлов.

Зная *IPv*4-адрес, а также исходную и новую маски подсети, можно определить следующие параметры:

- сетевой адрес этой подсети;
- широковещательный адрес этой подсети;
- диапазон адресов узлов этой подсети;
- количество созданных подсетей;
- количество узлов в подсети.

В таблице 26 показан пример решения одной из задач разбиения сети на подсети с фиксированной длиной маски.

Таблица 26 – Определение разбиения на подсети, пример

Дано:	
IP-адрес узла	172.16.77.120
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.240.0
Найти:	
Количество битов подсети	4
Количество созданных подсетей	16
Количество битов узлов в подсети	12
Количество узлов в подсети	4094
Сетевой адрес этой подсети	172.16.64.0
Адрес IPv4 первого узла в этой подсети	172.16.64.1
Адрес IPv4 последнего узла в этой подсети	172.16.79.254
Широковещательный адрес IPv4 в этой подсети	172.16.79.255

Рассмотрим, как была получена таблица 26.

Исходная маска подсети имела вид 255.255.0.0 или /16. Новая маска подсети — 255.255.240.0 или /20. Полученная разница составляет 4 бита. Так как 4 бита были заимствованы, мы можем определить, что были созданы 16 подсетей, так как 2^4= 16.

В новой маске, равной 255.255.240.0 или /20, остаётся 12 бит для узлов. Воспользуемся следующей формулой:

$$2^{12}= 4096–2=4094$$

Таким образом, 4094 узла в каждой подсети.

Бинарная операция «И» поможет определить сетевой адрес подсети для заданного изначально *IP*-адреса, в результате чего мы получим сеть 172.16.64.0.

В заключение необходимо установить первый узел, последний узел и широковещательный адрес для этой подсети. В нашем примере узловая часть – это последние 12 бит адреса. В первом узле для всех старших битов будет установлено значение 0, а для младшего бита — значение 1. В последнем узле для всех старших битов будет установлено значение 1, а для младшего бита — значение 0.

В этом примере узловая часть адреса находится в третьем и четвёртом октетах. Таблица 27 наглядно демонстрирует расположение бит в сетевой и узловой частях.

Таблица 27 – Определение адресов подсети

1-й октет	**2-й октет**	**3-й октет**	**4-й октет**	**Описание**
cccccccc	**cccccccc**	**ccccуууу**	уууууууу	Маска подсети
10101100 172	**00010000** 16	**0100**0000 64	00000001 1	Первый узел
10101100 172	**00010000** 16	**0100**1111 79	11111110 254	Последний узел
10101100 172	**00010000** 16	**0100**1111 79	11111111 255	Широковещательный

Заполните таблицу 28, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Таблица 28 – Определение разбиения на подсети, вариант А

Дано:	
IP-адрес узла	192.168.200.139
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.224
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 29 – Определение разбиения на подсети, вариант Б

Дано:	
IP-адрес узла	10.101.99.228
Исходная маска подсети	255.0.0.0
Новая маска подсети	255.255.128.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 30 – Определение разбиения на подсети, вариант В

Дано:	
IP-адрес узла	172.22.32.12
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.224.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 31 – Определение разбиения на подсети, вариант Г

Дано:	
IP-адрес узла	192.168.1.245
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.252
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 32 – Определение разбиения на подсети, вариант Д

Дано:	
IP-адрес узла	128.107.0.55
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.255.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 33 – Определение разбиения на подсети, вариант E

Дано:	
IP-адрес узла	192.135.250.180
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.248
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Заполните таблицу 29, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 30, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 31, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 32, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 33, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Планирование сетевой адресации

Окончательным этапом полного планирование сетевой адресации с фиксированной маской для готовой топологии сети является присвоение IP-адресов отдельным важным узлам сети, таким как маршрутизаторы и коммутаторы, серверы и выделенные узлы сети. Таким образом, последовательность действий включает:

- определение числа подсетей и вычисление числа добавочных бит;
- разбиение сети на подсети и составление схемы адресации;
- присвоение адресов узлам сети и интерфейсам сетевых устройств.

Для всех остальных обычных узлов сети отдельные адреса обычно не задаются при планировании, а составляется схема адресации аналогично, как это делалось ранее для одной сети, но теперь это выполняется для всех подсетей.

Например, имея сеть 172.22.0.0 с маской 255.255.0.0 или /16, мы разбиваем ее на 4 подсети. Для такого разбиения нам необходимо 2 добавочных бита, таким образом маска становится /18 или 255.255.192.0.

Адрес каждой подсети будет изменяться в пределах добавочных бит и принимать двоичные значения 00, 01, 10, 11. Сетевая часть адреса, предшествующая добавочным битам, не изменяется.

Таблица 34 – Определение добавочных бит подсетей

№ п/с	Первый октет	Второй октет	Третий октет	Четвертый октет	Сетевой адрес
0	**10101100** 172	**00010110** 22	**00**000000 0	00000000 0	172.22.0.0/18
1	**10101100** 172	**00010110** 22	**01**000000 64	00000000 0	172.22.64.0/18
2	**10101100** 172	**00010110** 22	**10**000000 128	00000000 0	172.22.128.0/18
3	**10101100** 172	**00010110** 22	**11**000000 192	00000000 0	172.22.192.0/18

В таблице 34 приведен пример такого расчета. В двоичном виде добавочные биты подсети выделены подчеркиванием, вся сетевая часть адреса выделена жирным, узловая обычным шрифтом. Ниже двоичного представления дан десятичный эквивалент. У всех подсетей маска одинаковая 255.255.192.0 или /18.

Как видно из таблицы 34, диапазоны всех подсетей не пересекаются друг с другом, а все вместе образуют исходную сеть 172.22.0.0/16.

Для присвоение сетевого адреса узлу или интерфейсу устройства можно выбрать любой допустимый узловой *IP*-адрес из подсети. Обычно интерфейсам маршрутизаторов присваивают первый допустимый адрес в подсети, поскольку он является шлюзом по умолчанию для всех узлов подсети и так его легче запомнить. Адреса узлов при необходимости могут выбираться совершенно произвольно.

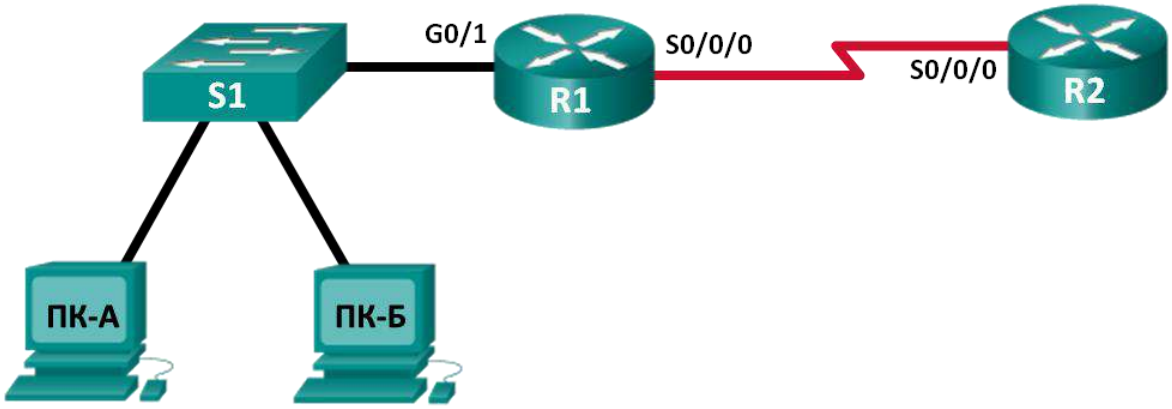


Рисунок 55 – Планирование сетевой адресации, вариант А

Таблица 35 – Планирование подсетей, вариант А

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				

Таблица 36 – Схема адресации, вариант А

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
R2	Serial 0/0/0		
S1	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		

Для сети с топологией на рисунке 55 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 35.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 36.

Для сети с топологией на рисунке 56 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 37.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 38.

Для сети с топологией на рисунке 57 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 39.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 40.

Для сети с топологией на рисунке 58 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 41.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 42.

Для сети с топологией на рисунке 59 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 43.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 44.

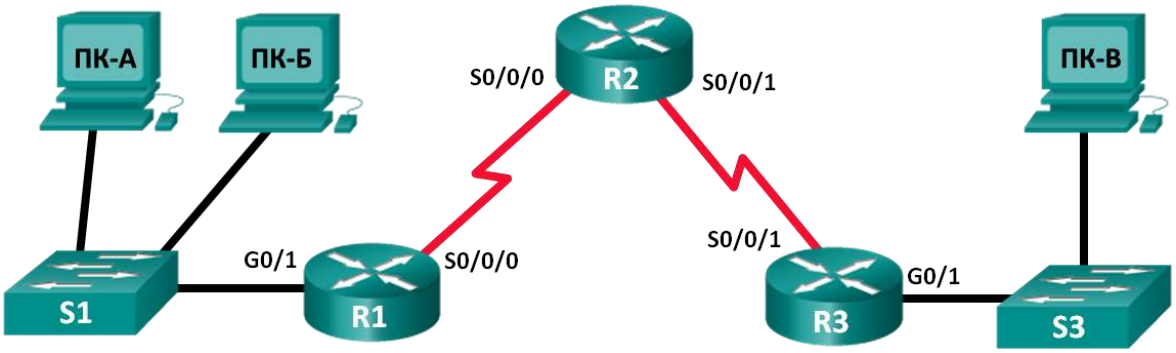


Рисунок 56 – Планирование сетевой адресации, вариант Б

Таблица 37 – Планирование подсетей, вариант Б

Номер подсети	Адрес подсети	Первый используемый адрес узла	Последний используемый адрес узла	Широковещательный адрес
0				
1				
2				
3				

Таблица 38 – Схема адресации, вариант Б

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
R2	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/1		
S1	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		

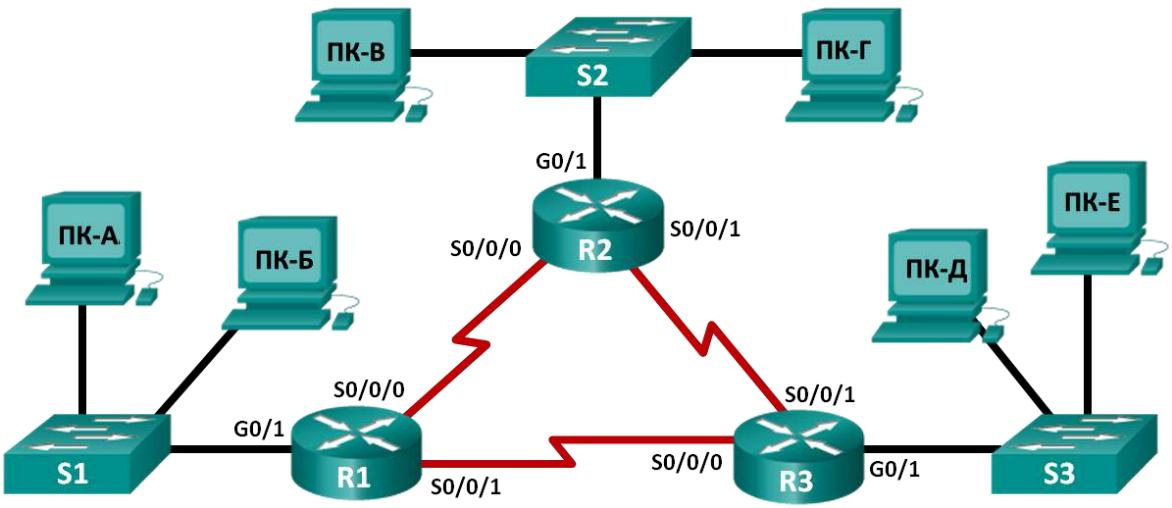


Рисунок 57 – Планирование сетевой адресации, вариант В

Таблица 39 – Планирование подсетей, вариант В

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				
2				
3				
4				
5				

Таблица 40 – Схема адресации, вариант В

Устройство	**Интерфейс**	**IP-адрес**	**Маска подсети**
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R2	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
S1	Vlan 1		
S2	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		
ПК-Г	GigabitEthernet		
ПК-Д	GigabitEthernet		
ПК-Е	GigabitEthernet		

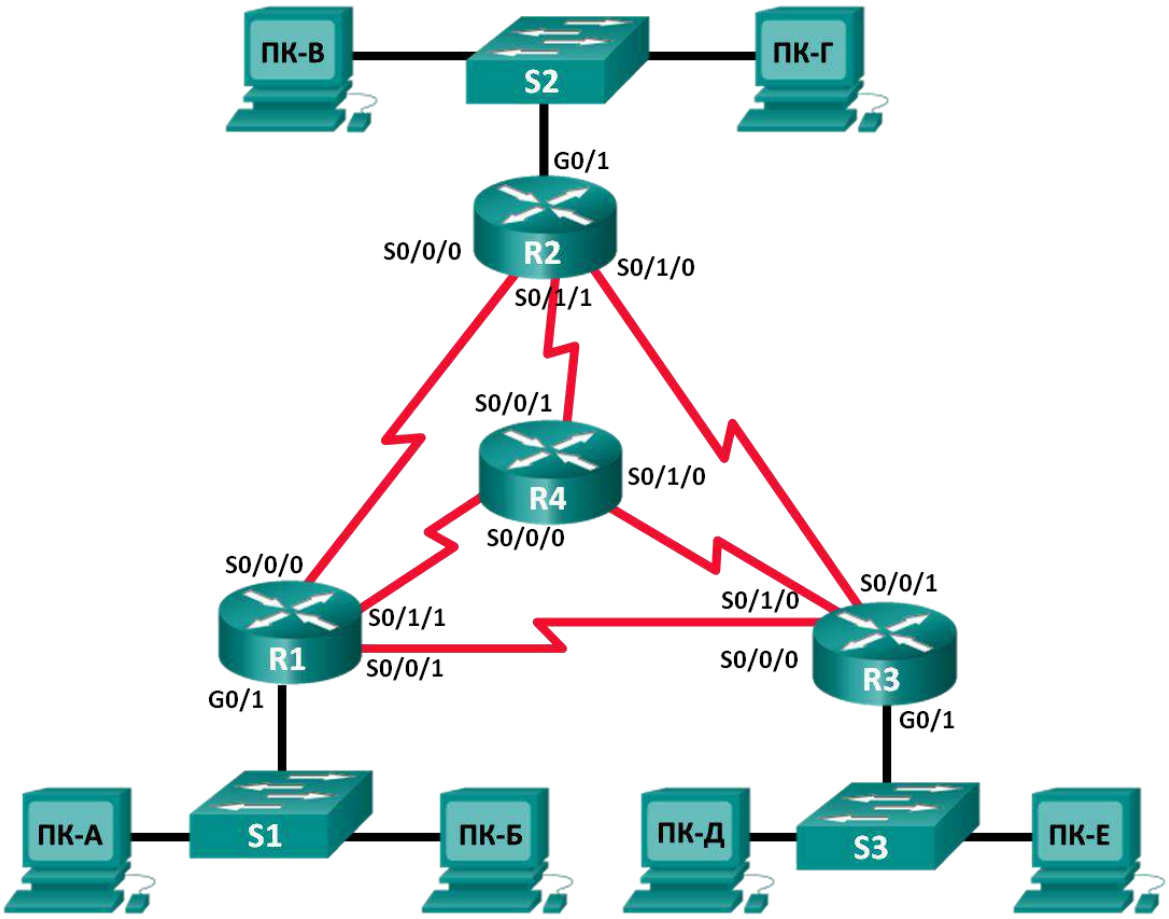


Рисунок 58 – Планирование сетевой адресации, вариант Г

Таблица 41 – Планирование подсетей, вариант Г

Номер подсети	Адрес подсети	Первый используемый адрес узла	Последний используемый адрес узла	Широковещательный адрес
0				
1				
2				
3				
4				
5				
6				
7				
8				

Таблица 42 – Схема адресации, вариант Г

Устрой-ство	**Интерфейс**	**IP-адрес**	**Маска подсети**
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/1		
R2	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/1/0		
	Serial 0/1/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/0		
R4	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/0		
S1	Vlan 1		
S2	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		
ПК-Г	GigabitEthernet		
ПК-Д	GigabitEthernet		
ПК-Е	GigabitEthernet		

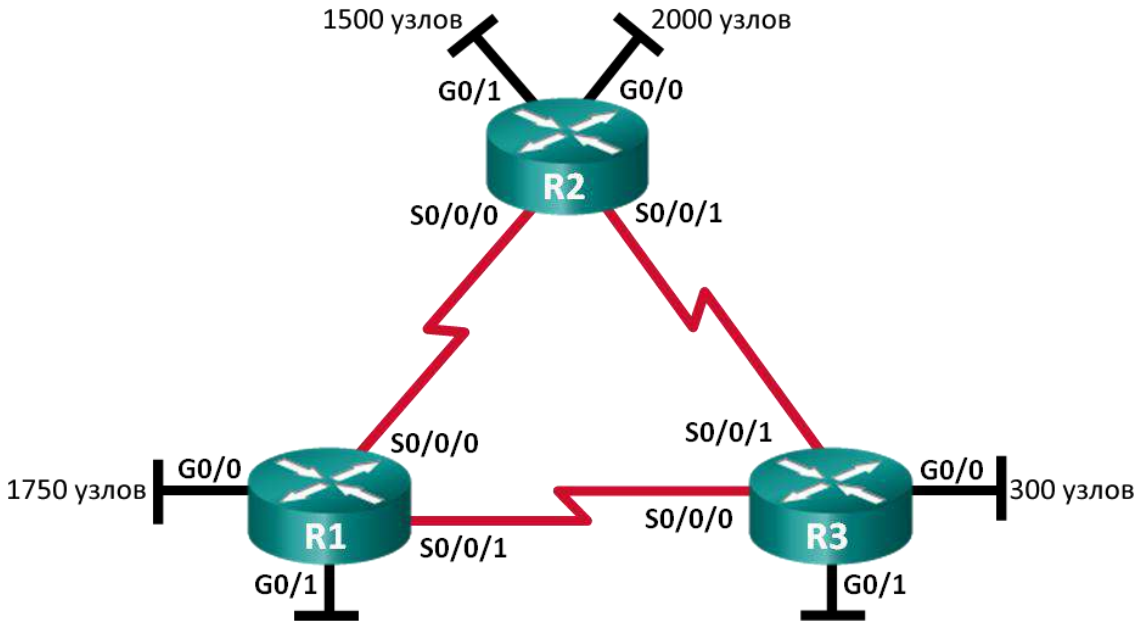


Рисунок 59 – Планирование сетевой адресации, вариант Д

Таблица 43 – Планирование подсетей, вариант Д

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				
2				
3				
4				
5				
6				
7				
8				

Таблица 44 – Схема адресации, вариант Д

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R2	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Cisco Networking Academy. CCNA: Введение в сетевые технологии [Электронный ресурс] – Режим доступа: https://www.netacad.com/ru/courses/networking/ccna-introduction-networks
2. Cisco Networking Academy. CCNA 7: Switching, Routing, and Wireless Essentials [Электронный ресурс] – Режим доступа: https://www.netacad.com/ru/courses/networking/ccna-switching-routing-wireless-essentials
3. Синецкий Р.М. Компьютерные сети [Электронный ресурс]: электрон. образ. ресурс. – ЮРГПУ (НПИ), 2020. – Режим доступа: https://sdo.srspu.ru/enrol/index.php?id=276
4. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: учеб. пособие для вузов. - СПб: Питер, 2010. - 944 с.
5. Таненбаум Э. Компьютерные сети: перевод с английского. - СПб: Питер, 2008. - 992 с.
6. Айвенс К. Компьютерные сети:как победить глюки и заставить домашнюю сеть работать без сбоев. - СПб: Питер, 2006. - 298 с.
7. Куроуз Д.Ф., Росс К. Компьютерные сети: многоуровневая архитектура Интернета. - СПб: Питер, 2004. - 765 с.
8. Закер К. Компьютерные сети. Модернизация и поиск неисправностей: перевод с английского. - СПб: БХВ-Петербург, 2004. - 1008 с.
9. Суворов А.Б. Телекоммуникационные системы, компьютерные сети и интернет: учеб. пособие для вузов. - Ростов-н/Д: Феникс, 2007. - 384 с.
10. Синецкий Р.М. Компьютерные сети: метод. указ. к практ. занятиям, лаб. работам и самостоят. раб. студ. заоч. формы обуч. бакалавр. "Информатика и вычисл. техника". - Новочеркасск: ЮРГПУ(НПИ), 2017. - 20 с.
11. Синецкий Р. М. Компьютерные сети: метод. указания к практ. занятиям, лаб. раб. и самостоят. работе студентов бакалавриата напрвлений подгот. 02.03.03, 09.03.01, 09.03.04 очной формы обучения. - Новочеркасск: ЮРГПУ (НПИ), 2017. - 19 с.

Учебное издание

Синецкий Роман Михайлович

АДМИНИСТРИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Учебное пособие

Редактор *Я.В. Максименко*

Подписано в печать 13.08.2020
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 5,11. Уч.- изд. л. 5,5. Тираж 100 экз. Заказ 46-0380.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ(НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
ipd-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

АЛГОРИТМЫ ОБУЧЕНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

**Учебно-методическое пособие
по изучению дисциплины
«Искусственный интеллект и мягкие вычисления»**

**Новочеркасск
ЮРГПУ (НПИ)
2020**

УДК 004.8 (076.5)
ББК 32.813

Рецензент – кандидат технических наук, доцент, доцент кафедры
«Программное обеспечение вычислительной техники»
Гавриков Михаил Михайлович

Кузнецова А.В., Мохов В.А., Алгазали С.М.М.

Алгоритмы обучения искусственных нейронных сетей: учебно-методическое пособие по изучению дисциплины Искусственный интеллект и мягкие вычисления / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2020. – 72 с.

Теоретический материал и приведенные примеры направлены на изучение принципов обучения искусственных нейронных сетей двух базовых типов архитектур – слоистых и однослойных. Контрольные вопросы к темам и практические задания для ручного расчёта направлены на углубленное изучение алгоритмов обучения с целью их последующей реализации в виде программ для ЭВМ и создания различных модификаций.

Учебно-методическое пособие предназначено для студентов, обучающихся по программам бакалавриата направлений подготовки 09.03.01 «Информатика и вычислительная техника» очной и заочной форм обучения, 09.03.04 «Программная инженерия», 02.03.03 «Математические основы и алгоритмизация информационных систем».

УДК 004.8 (076.5)
ББК 32.813

ВВЕДЕНИЕ

Искусственные нейронные сети (ИНС) работают на тех же принципах, что и их биологический прототип – мозг. С точки зрения реализации – это один из программных механизмов, который позволяет программе обучаться, учитывать имеющийся опыт, самостоятельно находить закономерности в исходных данных, запоминать информацию и восстанавливать её по отдельным фрагментам, учитывать предысторию наблюдаемых процессов и накапливать информацию для выработки правильной стратегии управления, и многое другое. В целом, это семейство технологий для решения трудно формализуемых задач, которые требуют аналитических способностей, присущих человеку.

Нейросети могут иметь различную структуру и в зависимости от задачи, которую необходимо решить, наилучшим образом подходит та или иная архитектура. Например, для классификации используют многослойные персептроны, для анализа временных рядов, распознавания речи и прочих событий, растянутых во времени, – рекуррентные сети, для распознавания изображений – свёрточные сети. Однако есть и универсальные архитектуры, которые предназначены для решения нескольких типов задач. В этом случае всё зависит от способа представления данных для обучения ИНС.

По общему мнению, наиболее подходящим языком для реализации нейронных сетей является язык *python*. Для создания и обучения нейронных сетей можно воспользоваться его готовыми фреймворками типа *tensorflow*, *keras*, *theano* и надстройками над ними. Как правило, это мощные средства, изначально разрабатывавшиеся в рамках крупных исследовательских проектов, переданные программистскому сообществу в бесплатное пользование.

Однако, создавать нейронные сети можно и на *Java*, и на *C++*, и вообще на любом общедоступном процедурном или объектно-ориентированном языке программирования. В этом случае придётся самостоятельно заниматься моделированием системы нейронов и алгоритмом обучения.

Предлагаемое учебное пособие направлено на формирование базовых представлений и объяснение принципов работы наиболее распространённых нейронных сетей – слоистых сетей прямого распространения и самоорганизующихся сетей Кохонена.

Тема 1. МОДЕЛЬ ИСКУССТВЕННОГО НЕЙРОНА И ЕГО МОДИФИКАЦИИ

Искусственный нейрон является основой любой искусственной нейронной сети. Нейроны представляют собой относительно простые, однотипные элементы, имитирующие работу биологических нейронов мозга.

Искусственные нейроны до определенной степени отражают динамику передачи сигналов в реальных биологических нейронах. Биологические нервные клетки (нейроны) состоят из тела клетки, входных отростков – дендритов и выхода – аксона. Очень упрощая, работу нейрона можно описать следующим образом. Дендриты получают сигналы от других клеток через особые биологические образования – синапсы (или синаптические щели). Сигналы поступают в тело клетки, где они суммируются с другими такими же входными сигналами. Если суммарный сигнал в течение короткого промежутка времени является достаточно большим, то клетка возбуждается, вырабатывая в аксоне импульс, который передается на следующие клетки. Не вдаваясь в подробности, подчеркнем, что модели нейронов только очень грубо отражают работу биологических живых нервных клеток.

Искусственный нейрон, так же, как и его естественный прототип, имеет группу входов, которые соединены с выходами других нейронов, и один выход, через который сигнал возбуждения или торможения поступает на входы других нейронов. На рис. 1.1 показана схема искусственного нейрона.

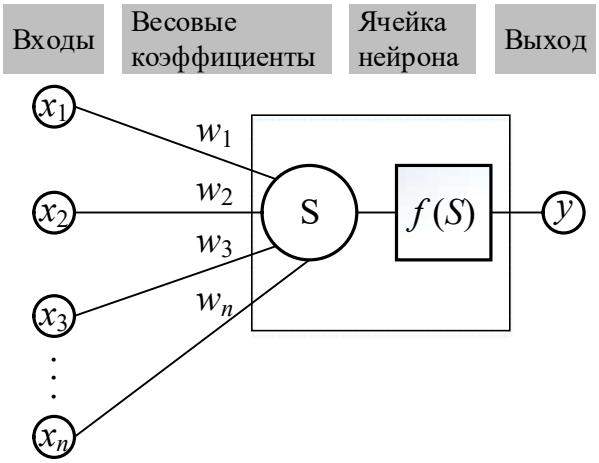


Рис. 1.1. Графическая модель искусственного нейрона

Каждый вход характеризуется весовым коэффициентом или просто весом w_i, который по своему физическому смыслу отражает силу синаптической связи и эквивалентен электрической проводимости. Текущее состояние нейрона определяется как взвешенная сумма его входов:

$$S = \sum_{i=1}^{n} x_i \cdot w_i, \qquad (1.1)$$

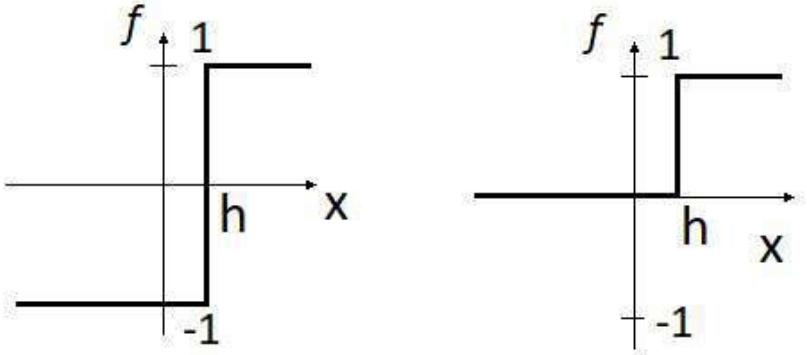
где n – число входов нейрона; x_i – значение i-го входа нейрона; w_i – вес i-го синапса.

Выход нейрона определяется значением некоторой функции от его состояния, т.е.:

$$y = f(S). \qquad (1.2)$$

Эту функцию называют активационной, сжимающей функцией или функцией возбуждения нейрона.

Первой функцией в модели искусственного нейрона была пороговая функция, разновидности которой представлены на рис. 1.2.



а *б*

Рис. 1.2. Пороговые активационные функции
а – пороговая биполярная; *б* – пороговая бинарная

Для рассмотрения работы нейрона неважно, приходит сигнал к нейрону из внешнего мира или с другого нейрона, и неважно, куда отправляется сигнал с нейрона.

В том случае, когда на вход искусственного нейрона подаются двоичные сигналы, а активационная функция – пороговая (функции Хевисайда) с параметром h, мы имеем дело с моделью *формального нейрона*, предложенного Мак-Каллоком и Питтсом в 1943 году. Математическое представление модели с n входами имеет вид:

$$y = \begin{cases} 1, \ \sum_{i=1}^{n} x_i * w_i \geq h; \\ 0, \ \sum_{i=1}^{n} x_i * w_i < h, \end{cases} \qquad (1.3)$$

где x – входной сигнал; y – выходной сигнал; h – значение порога.

Например, для нейрона со следующими значениями весов его трёх входов: $w_1 = 0.30$, $w_2 = 0.40$, $w_3 = 0.25$, значением порога $h = 0.45$ и входным двоичным сигналом 1 0 1 выходное значение будет определяться выражением:

$$0.3*1 + 0.4*0+0.25*1 = 0.55; \ 0.55 > 0.45 \ \Rightarrow \ y=1.$$

Для вычисления выходного сигнала модифицированного формального нейрона использовать правило:

$$\frac{(Сумма \ активных \ весов)}{(Полная \ сумма \ входных \ весов)} \geq (Порог)$$

Для этого нейрона справедливо то, что если есть в наличии два любых активных входа, выход будет всегда активным. Формула расчета значения на выходе модифицированного нейрона будет иметь следующий вид:

$$\frac{0.3*1+0.25*1}{|0.3|+|0.4|+|0.25|} = 0.58; \quad 0.58 > 0.45 \Rightarrow y = 1.$$

Обучим нейрон, функционирующий по формуле (1.3), распознаванию идеального графического образа, представленного битовыми значениями (рис. 1.3). Входной вектор, соответствующий символу «F», можно закодировать, например, построчно:

1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0.

Обучение заключается в подборе параметров нейрона – весовых коэффициентов и значения порога, таких, что только при заданном входном сигнале на выходе нейрона будет единица. Очевидно, что таких наборов параметров может быть сколь угодно много.

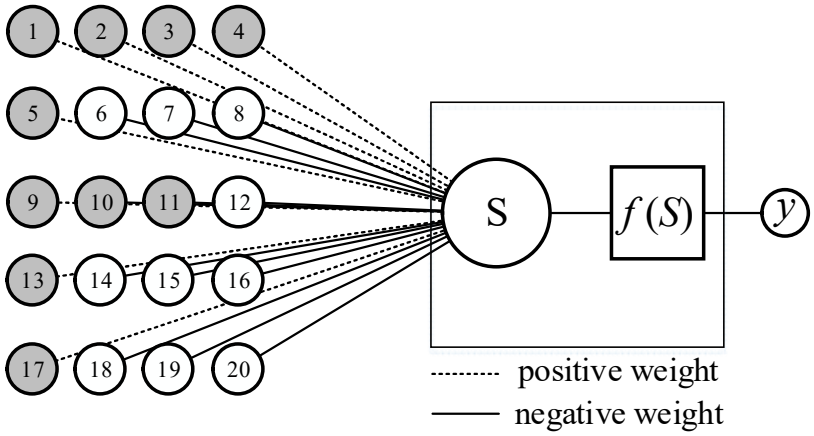


Рис. 1.3. Модель нейрона для распознавания символа «F»

Выберем для активных входов, т.е. входов, на которые подаются 1, малые положительные значения:

w_1=0.11, w_2=0.14, w_3=0.12, w_4=0.10, w_5=0.12,

w_9=0.11, w_{10}=0.13, w_{10}=0.10, w_{13}=0.11, w_{17}=0.14.

Для неактивных входов – малые случайные отрицательные значения в диапазоне [-1.0 .. -0.2].

Подсчитаем взвешенную сумму для входного сигнала, соответствующего символу «F»:

0.11+0.14+0.12+0.10+0.12+0.11+0.13+0.10+0.11+ 0.14=1.18.

Значение порога h следует выбрать таким, чтобы при отсутствии хотя бы одной единицы на входе, взвешенная сумма оказалась меньше порога. Поскольку минимальное значение веса для активного входа равно 0.1, то величина порога должна находиться в диапазоне [1.09 .. 1.18]. При отсутствии хотя бы одной единицы на определенном выше множестве активных входов, взвешенная сумма буде заведомо меньше значения 1.09. В наилучшем случае она может быть равной 1.08, если ноль окажется на входе с номером 4 или с номером 10. Если же на активных входах будут присутствовать единицы, но появятся «лишние» единицы на тех входах, где весовые коэффициенты отрицательные, то они заведомо уменьшат взвешенную сумму 1.18 на величину -0.2. Итоговая сумма снова станет меньше 1.09.

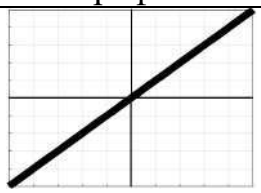
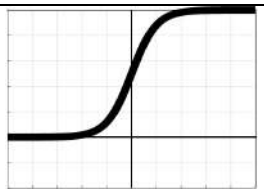
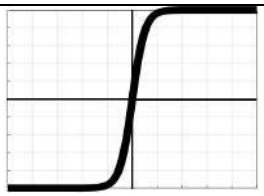
Значения входов x_i и выхода y формального нейрона могут быть как бинарными, т.е. равными 0 или 1, так и бинарными биполярными, равными ±1. В основополагающей работе Мак-Каллока и Питтса [1] входы и выходы нейронов предполагались бинарными,

веса синапсов считались бинарными биполярными, а активационная функция – пороговой. Исследования нейросетей в [1] проводились с точки зрения анализа логических исчислений, которые могут быть построены на базе формальных нейронов. В частности, было показано, что «для всякого логического выражения, удовлетворяющего некоторым условиям, можно найти сеть, имеющую описываемое этим выражением поведение».

Заменив простую пороговую функцию на гладкую ограниченную функцию, можно перейти к модели *градуального нейрона*. В такой модели нейрона информация, передаваемая от нейрона к нейрону, представлена не строгими бинарными, а действительными числовыми значениями.

В табл. 1.1 представлены наиболее часто употребляемые гладкие функции активации. К желательным свойствам функций активации относятся: нелинейность, непрерывная дифференцируемость, бесконечная область значений, монотонность.

Таблица 1.1 Гладкие функции активации искусственного нейрона

Название	График	Уравнение
Линейная		$f(x) = x$
Логистическая или сигмоидальная		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
Гиперболический тангенс		$f(x) = th(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$

Наиболее часто в качестве активационной функции используется так называемая *сигмоидальная функция*, которая в общем случае имеет следующий вид:

$$f(x) = \frac{1}{1+e^{-ax}} \ , \qquad (1.4)$$

При уменьшении параметра α функция становится более пологой, и в пределе, при $\alpha = 0$, вырождается в горизонтальную линию на уровне 0.5. При увеличении α сигмоид приближается по внешнему виду к функции единичного скачка с порогом $h = 0$. Из выражения (1.4) следует, что выходное значение нейрона лежит в диапазоне [0..1]. Одно из ценных свойств сигмоидальной функции – простое выражение для ее производной:

$$f'(x) = af(x)(1 - f(x)) \ , \qquad (1.5)$$

Следует отметить, что сигмоидальная функция дифференцируема на всей оси абсцисс, что используется в большинстве алгоритмов обучения. Вторым замечательным свойством этой функции является её способность усиливать слабые сигналы лучше, чем большие, и предотвращать насыщение от больших сигналов, поскольку они соответствуют областям аргументов, где сигмоида имеет пологий наклон.

Вопросы для самоконтроля

1. Что представляет собой модель формального нейрона?
2. Какой смысл несут в себе весовые коэффициенты нейрона?
3. Что представляет собой взвешенная сумма входного сигнала?
4. Каково назначение пороговой функции нейрона?
5. Чем отличается модифицированная модель формального нейрона от классической модели?
6. Каким образом производится ручная подстройка нейрона сети?
7. Что понимается под активационной функцией нейрона? Какой смысл она в себе заключает?
8. В чем преимущества гладких функций перед пороговой функцией активации?
9. Опишите особенности и достоинства сигмоидальной функции активации.

Тема 2. ВЫЧИСЛЕНИЕ ОШИБКИ В СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ СИГНАЛА. МЕТОДЫ ВЫЧИСЛЕНИЯ ОШИБКИ

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами сети при прямом проходе входного сигнала (образца) по нейронной сети – от входов к выходам.

Различают ошибку для образца (или ошибку за один проход) и ошибку для эпохи. Ошибка, формируемая в каждой эпохе, при верном обучении должна идти на спад. Если этого не происходит, значит, алгоритм обучения не верен.

Ошибку можно вычислить разными методами, но мы рассмотрим лишь три основных: *mean squared error* (далее *MSE*), *root MSE* (*RMSE*) и *Arctan*:

$$E_{MSE} = \frac{(d_1\text{-}y_1)^2 + (d_2\text{-}y_2)^2 + \ldots + (d_n\text{-}y_n)^2}{n}, \quad (2.1)$$

$$E_{RMSE} = \sqrt{\frac{(d_1\text{-}y_1)^2 + (d_2\text{-}y_2)^2 + \ldots + (d_n\text{-}y_n)^2}{n}}, \quad (2.2)$$

$$E_{arctan} = \frac{\arctan^2 (d_1\text{-}y_1) + \ldots + \arctan^2 (d_n\text{-}y_n)}{n}. \quad (2.3)$$

Принцип подсчета ошибки во всех случаях одинаков. За каждый проход вычисляется ошибка путем вычитания от полученного ответа (d_i) от идеального ответа (y_i). Далее, результат либо возводится в квадрат, либо вычисляется квадратный тангенс из этой разности. В случае одного прохода $n = 1$. В случае эпохи n – число образцов в обучающей выборке. По завершении эпохи полученная сумма делится на количество наборов данных в обучающей выборке. При вычислении ошибки на всей эпохе можно определить наихудшую и наилучшую ошибки на определенных образцах.

Для вычисления ошибки сети нет какого-либо ограничения. Стоит лишь учитывать, что каждый метод считает ошибки поразному. У *Arctan*, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У *root MSE* будет наименьшая ошибка, поэтому, чаще всего, используют *MSE*, которая сохраняет баланс в вычислении ошибки.

Пример расчета ошибки для обучения на примере логической функции *XOR*.

Пусть задана нейронная сеть со случайно заданными значениями весовых коэффициентов (рис. 2.1). Функции активации нейронов – сигмоидальные.

Параметры сети:
w_1=0.45, w_2=0.78, w_3= -0.12,w_4=0.13, w_5=1.50, w_6= -2.3.

Обучающая выборка:

x_1	x_2	y
1	0	1
0	1	1
0	0	0
1	1	0

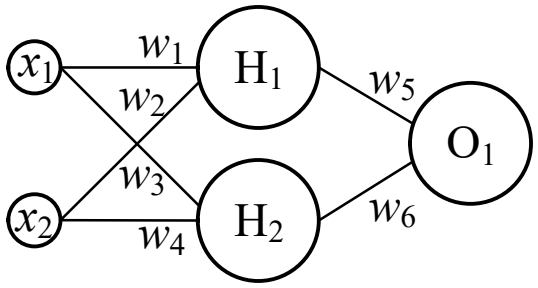


Рис. 2.1. Архитектура сети

Расчет выходных значений нейронов скрытого слоя для первого элемента обучающей выборки (x_1=1, x_2=0, y=1):

H_{1inp} = 1*0.45+0*(-0.12)=0.45 H_{1out} = sigmoid(0.45)=0.61
H_{2inp} = 1*0.78+0*0.13=0.78 H_{2out} = sigmoid(0.78)=0.69

Расчет выходных значений нейронов выходного слоя:

O_{1inp} = 0.61*1.5+0.69*-2.3=-0.67 O_{1out} = sigmoid(-0.67)=0.34

Абсолютная ошибка: Δ_1 = 1-0.34 = 0.66

Для остальных элементов обучающей выборки расчеты аналогичные. Среднеквадратическая ошибка сети на всей выборке:

$$E_{MSE} = ((1 - 0.34)^2+(1 - 0.37)^2+(0 - 0.40)^2+(0 - 0.32)^2)/4 =$$
$$= (0.66^2+0.63^2+(-0.4)^2+(-0.32)^2)/4 = 0.27 = 27\%.$$

Вопросы для самоконтроля

1. Что понимается под ошибкой сети?
2. Какие методы вычисления ошибки являются наиболее распространенными? В чем их отличие?
3. Приведите примеры других способов и методов вычисления ошибки, отличных от изученных на практическом занятии.
4. С какой целью вычисляется ошибка сети?

Тема 3. ОБУЧЕНИЕ СЕТЕЙ ПРЯМОГО РАСПРОСТРАНЕНИЯ СИГНАЛА. АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Нейронные сети «прямого распространения сигнала и обратного распространения ошибки» – это мощнейший инструмент поиска закономерностей, прогнозирования, качественного анализа. Более краткими названиями являются синонимы: *сети прямого распространения сигнала* (*feed forward*) или *сети обратного распространения ошибки (back propagation)*. Такие названия сети получили из-за используемого алгоритма обучения, в котором уменьшение ошибки производится от выходного слоя к входному, т. е. в направлении, противоположном направлению распространения сигнала при нормальном функционировании сети (алгоритм *Back Propagation*).

Нейронная сеть прямого распространения сигнала состоит из нескольких слоев нейронов, причем каждый нейрон слоя i связан с каждым нейроном слоя $i+1$, т. е. речь идет о *полносвязной* НС.

Задача обучения сети сводится к нахождению некой функциональной зависимости $\mathbf{Y}=F(\mathbf{X})$ где $\mathbf{X}$ – входной, а $\mathbf{Y}$ – выходной векторы. В общем случае такая задача, при ограниченном наборе входных данных, имеет бесконечное множество решений. Для ограничения пространства поиска при обучении ставится задача минимизации целевой функции ошибки, которая находится по методу наименьших квадратов (2.1). Обучение нейросети производится методом градиентного спуска, т. е. на каждой итерации изменение веса производится по формуле:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \qquad (3.1)$$

где η – параметр, определяющий скорость обучения.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} * \frac{\partial S_j}{\partial w_{ij}} \qquad (3.2)$$

где y_i – значение выхода j-го нейрона; S_j – взвешенная сумма входных сигналов. При этом множитель

$$\frac{\partial S_j}{\partial w_{ij}} = x_i, \tag{3.3}$$

где x_i – значение i-го входа нейрона.

Рассмотрим определение первого множителя формулы (3.2).

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{\mathrm{d}y_k}{\mathrm{d}S_k} * \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{\mathrm{d}y_k}{\mathrm{d}S_k} * w_{jk}^{(n+1)}, \tag{3.4}$$

где k – число нейронов в слое n+1.

Введем вспомогательную переменную δ (дельта):

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_i} * \frac{\mathrm{d}y_i}{\mathrm{d}S_j}. \tag{3.5}$$

Тогда можно определить рекурсивную формулу для определения δ n-го слоя, если нам известно δ следующего (n+1)-го слоя:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} * w_{jk}^{(n+1)} \right] * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} \tag{3.6}$$

Нахождение же δ для последнего N-го слоя не представляет трудности, так как целевой вектор известен, т. е. вектор тех значений, которые должна выдавать нейронная сеть при данном наборе входных значений.

$$\delta_j^{(N)} = \left(y_i^{(N)} - d_i \right) * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} \tag{3.7}$$

Вернёмся к формуле (3.1) в раскрытом виде:

$$\Delta w_{ij}^{(n)} = -\eta * \delta_j^{(n)} * x_i^n. \tag{3.8}$$

Сомножитель $\delta_j^{(n)} * x_i^n$ называется *градиентом*. В данном случае это – элемент, включающий в себя производную функции ошибки относительно одного весового коэффициента.

Полный алгоритм обучения нейронной сети состоит из следующих этапов:

1. подать на вход сети случайно выбранный входной вектор и определить для него значения выходов (прямой проход сигнала);
2. рассчитать δ для выходного слоя по формуле (3.7) и для скрытых слоев сети по формуле (3.6);
3. рассчитать изменения весов $\Delta w_{ij}^{(n)}$ всех нейронов сети по формуле (3.8);
4. скорректировать веса сети:

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}; \qquad (3.9)$$

5. повторить шаги 1 – 4 для всех векторов обучающей выборки, т.е. для всей *эпохи*;
6. если ошибка E_{MSE} на эпохе существенна, то перейти на шаг 1 (к новой эпохе), иначе – алгоритм завершен.

На рисунке 3.1 проиллюстрирован прямой проход сигнала по нейронной сети с одним скрытым слоем. Начальные значения весовых коэффициентов и входной вектор обучающей выборки те же, что и в предыдущем примере (см. тему 2).

Рассмотрим подробный расчёт новых весовых коэффициентов для одного элемента обучающей выборки.

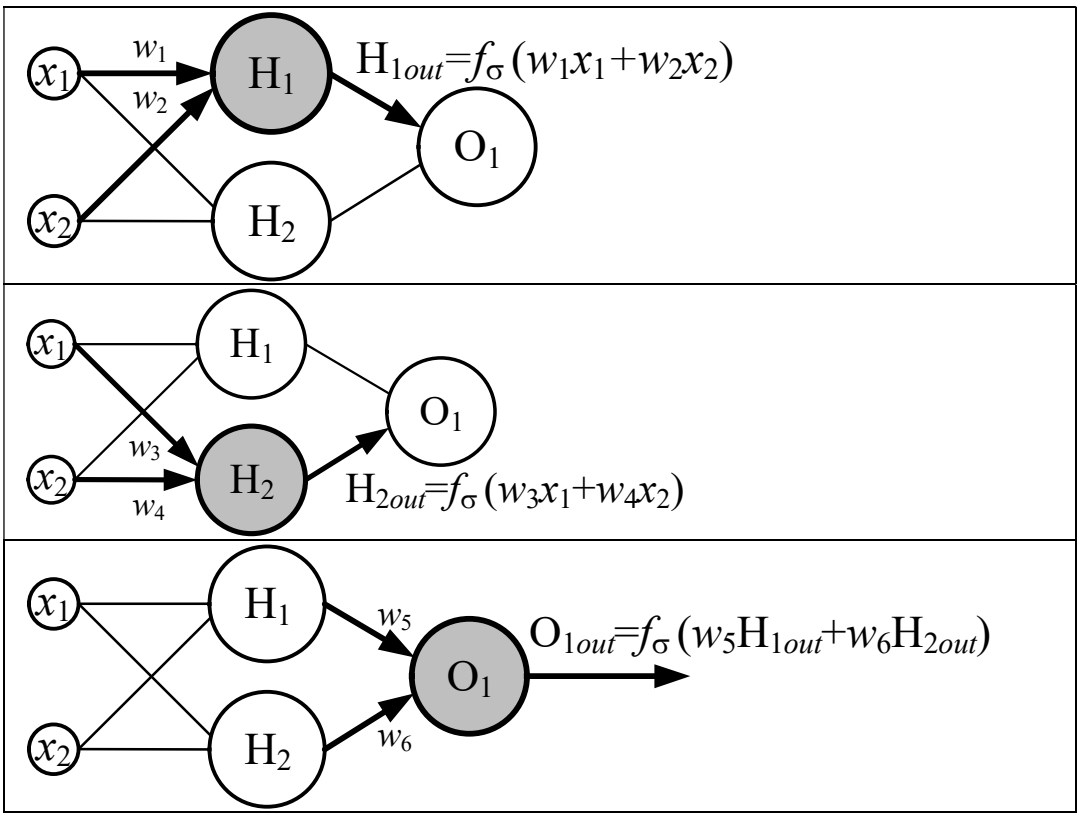


Рис. 3.1. Иллюстрация прямого прохода сигнала по НС

Расчет величины δ для единственного нейрона выходного слоя осуществляется на основе выходного значения O_{1out}, идеального значения выхода y_{ideal}, соответствующее входному вектору в обучающей выборке, и производной от выходного значения:

$$\delta_O = (O_{1out} - y_{ideal}) * f'(O_{1out}) \qquad (3.10)$$

Обратите внимание на производную. Во-первых, с алгоритмом нужно использовать только те функции активации, которые могут быть дифференцированы. Во-вторых, чтобы не делать лишних вычислений, формулу производной можно заменить на более дружелюбную и простую формула вида:

$$f'(OUT) = \begin{vmatrix} f_{sigmoid} = (1 - OUT) * OUT; \\ f_{th} = 1 - OUT^2. \end{vmatrix} \qquad (3.11)$$

Для выходного нейрона имеем:
Данные: $O_{1out} = 0.33$ $O_{1ideal} = 1$ $E_{MSE} = 0.449$
Решение: $\delta_{O_1} = (1 - 0.33) * ((1 - 0.33) * 0.33) = 0.148$

Для расчёта величины дельта в нейронах скрытого слоя вместо сомножителя ($O_{1out} - y_{ideal}$) в формуле (3.10) должна использоваться сумма $\sum(w_i * \delta_{next})$, т.е. сумма произведений всех дельт и весов нейронов последующего слоя, с которыми этот нейрон связан:

$$\delta_H = \sum(w_i * \delta_i) * f'(OUT) \qquad (3.12)$$

Но поскольку в рассматриваемой сети нейрон H_1 связан только с одним выходным нейроном O_1, производная от выхода H_1 умножается на единственны величину, равную $w_5 * \delta_{O1}$:

Данные: $H_{1out} = 0.61 \quad w_5 = 1.5 \quad \delta_{O1} = 0.148$
Решение: $\delta_{H1} = ((1 - 0.61) * 0.61) * (1.5 * 0.148) = 0.053$

На рисунке 3.2 проиллюстрирован процесс вычисления значений δ для нейронов выходного и скрытого слоёв.

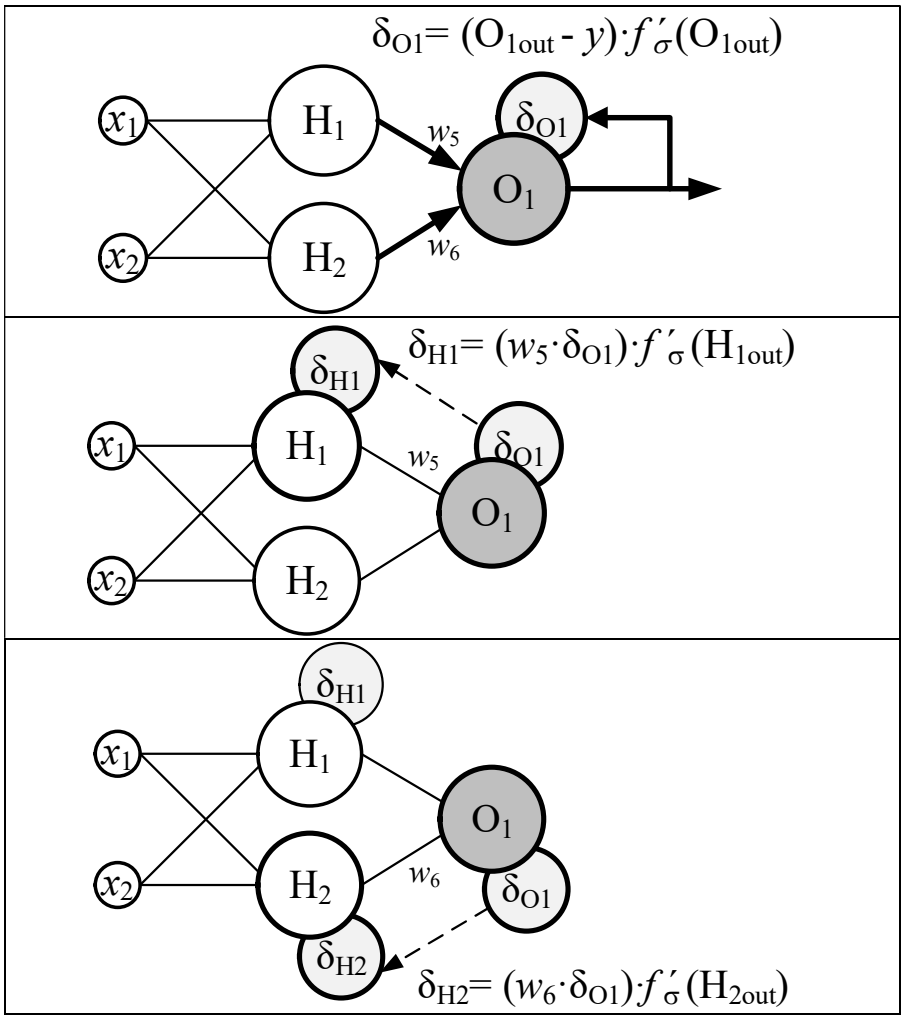


Рис. 3.2. Иллюстрация вычисления величины δ

Вычисление δ_{H2} производится аналогично: $\delta_{H2} = -0.073$.

Для расчета подстроечных значений весов Δw следует найти градиент для каждого входа нейрона. В рассматриваемом методе формула нахождения градиента имеет вид:

$$GRAD_B^A = OUT_A * \delta_B . \qquad (3.13)$$

Здесь точка A – это точка в начале синапса (выход предыдущего нейрона), а точка B – на конце синапса (дельта самого нейрона). Таким образом можно подсчитать градиент w_5 следующим образом:

Данные: $H_{1out} = 0.61$ $\delta_{O1} = 0.148$
Решение: $GRAD_{w5} = 0.61 * 0.148 = 0.09$

Расчет подстроечных значений весовых коэффициентов осуществляется по формуле:

$$\Delta w_i = \eta * GRAD_{w_i} + \alpha * \Delta w_{i(pred)} . \qquad (3.14)$$

В отличие от формулы (3.8) здесь присутствует добавочный член. Переводя формулу в слова получим: изменение веса синапса равно коэффициенту скорости обучения η, умноженному на градиент этого веса, плюс момент α, умноженный на предыдущее изменение этого же веса. На первой итерации этот член равен 0. Настоятельно рекомендуется использовать момент α, так как это позволит избежать проблем с локальными минимумами.

Для весового коэффициента w_5 это будет выглядеть следующим образом.

Данные: $\eta = 0.7$ $\alpha = 0.3$ $w_5 = 1.5$ $GRAD_{w5} = 0.09$
Решение: $\Delta_{w5}(i\text{-}1) = 0$ (предыдущих итераций не было)
$\Delta_{w5} = 0.7 * 0.09 + 0.3 * 0 = 0.063$

Новое значение весового коэффициента w_5 получается путём сложения его значения с величиной подстройки:

$$w_5 = w_5 + \Delta w_5 = 1.5 + 0.063 = 1.563$$

Таким образом, после применения алгоритма весовой коэффициент w_5 изменился с 1.5 до 1.563 на малую величину 0.063.

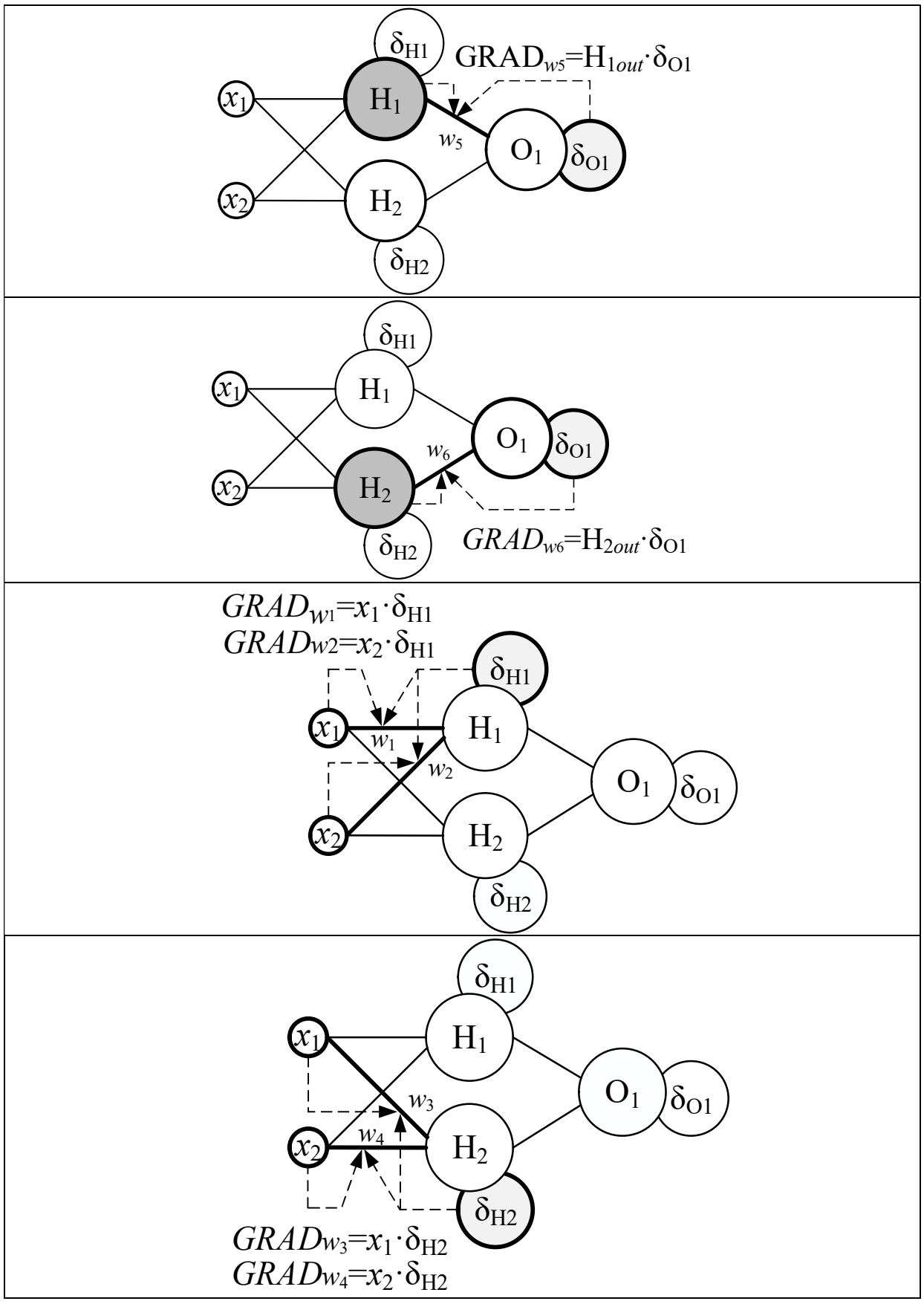


Рис. 3.3. Иллюстрация вычисления градиентов

Иллюстрация вычисления новых весовых коэффициентов представлена на рисунке 3.4.

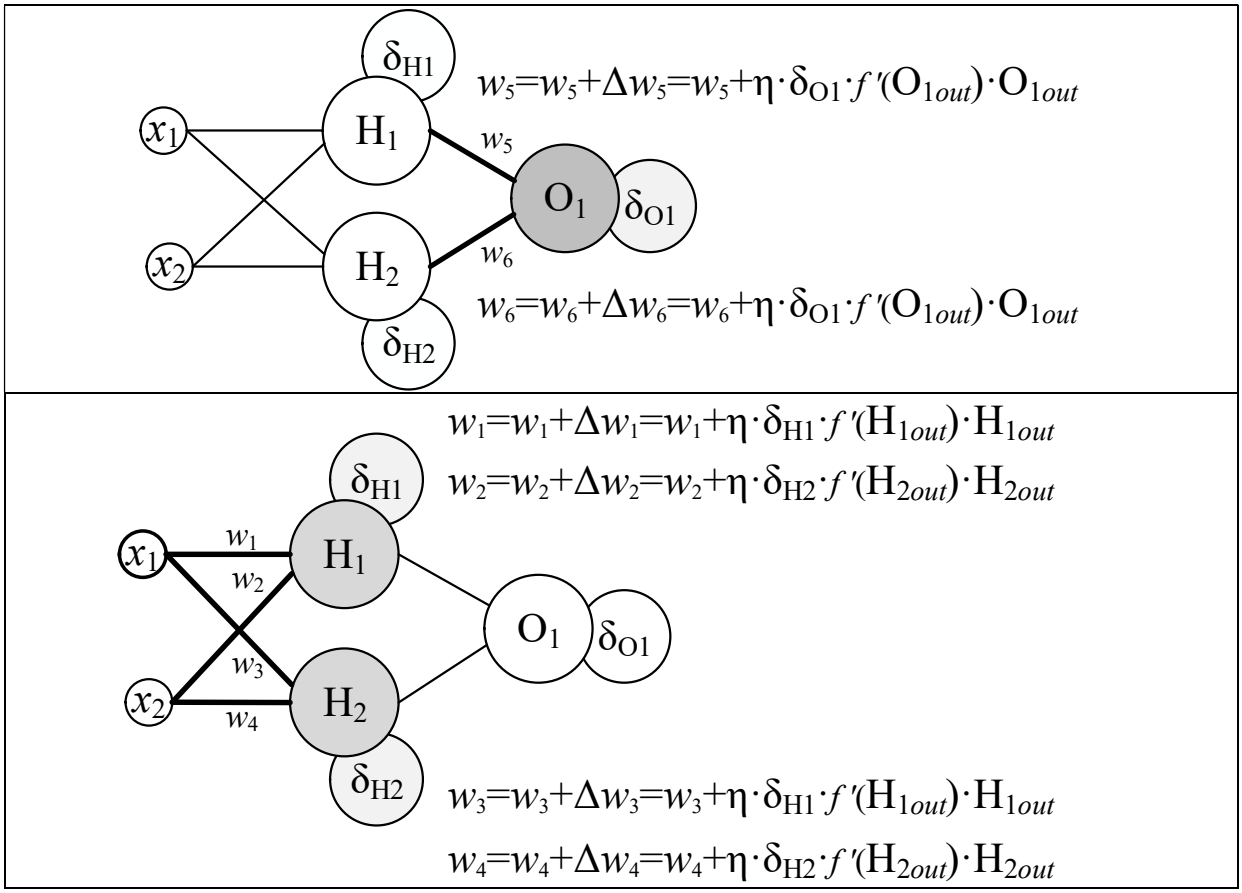


Рис. 3.4. Иллюстрация подстройки весовых коэффициентов

Величины градиентов, подстроечных коэффициентов и собственно новых значений весов для всех остальных входов нейронов вычисляются аналогично вычислениям для w_5. Результаты расчётов представлены в таблице 3.1. Из таблицы видно, как «колеблются» значения весовых коэффициентов после подстройки в сторону векторов обучающей выборки:

Таблица 3.1 Результаты расчётов весовых коэффициентов

	GRAD	Δw	$w^{(t)}$	$w^{(t+1)}$	$w^{(t+2)}$	$w^{(t+3)}$	$w^{(t+4)}$
w_6	0.090	0.071	-2.300	-2.229	-2.153	-2.168	-2.215
w_5	0.102	0.063	1.500	1.563	1.630	1.613	1.570
w_4	0.000	0.000	0.130	0.130	0.074	0.057	0.081
w_3	0.000	0.000	-0.120	-0.120	-0.080	-0.069	-0.089
w_2	-0.073	-0.052	0.780	0.729	0.713	0.709	0.736
w_1	0.053	0.037	0.450	0.487	0.498	0.501	0.479

Последовательная подстройка весовых коэффициентов способствует снижению ошибки сети. Ошибка для всей эпохи, полученная на первоначальных весовых коэффициентах, составила 0.273 (27.3%). После завершения первой эпохи ошибка уменьшилась до величины 0.264 (26.4%).

Вопросы для самоконтроля

1. Опишите суть алгоритма обратного распространения ошибки.
2. Какие задачи решаются на каждом из двух проходов алгоритма?
3. Что понимается под градиентным спуском и как используется градиент при подстройке коэффициентов сети?
4. Что понимается под скоростью обучения? Какие значения скорости рекомендуется выбирать?
5. Какую роль в обучении играет использование момента?
6. в процессе обучения циклически решаются однокритериальные задачи оптимизации.
7. Поясните смысл высказывания: «в процессе обучения многослойного персептрона циклически решаются однокритериальные задачи оптимизации».
8. Какие модификации базового алгоритма распространения Вам известны? Опишите их.

Тема 4. МАТРИЧНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Для вычисления преобразований сигналов между слоями нейронной сети можно использовать произведение матриц. Первая матрица содержит веса нейронов одного слоя. Вторая матрица – вектор – содержит поступившие на этот слой сети сигналы. Результат умножения (×) этих двух матриц – вектор с суммами взвешенных входных сигналов. Затем значения вектора используются в качестве аргумента функции активации (сигмоиды) нейронов слоя.

Рассмотрим прямое прохождение сигнала для нейронной сети с тремя входами и тремя выходами и двумя слоями (рис. 4.1). Первый слой – скрытый с нейронами H1, H2, H3. Второй слой – выходной с нейронами O1, O2, O3.

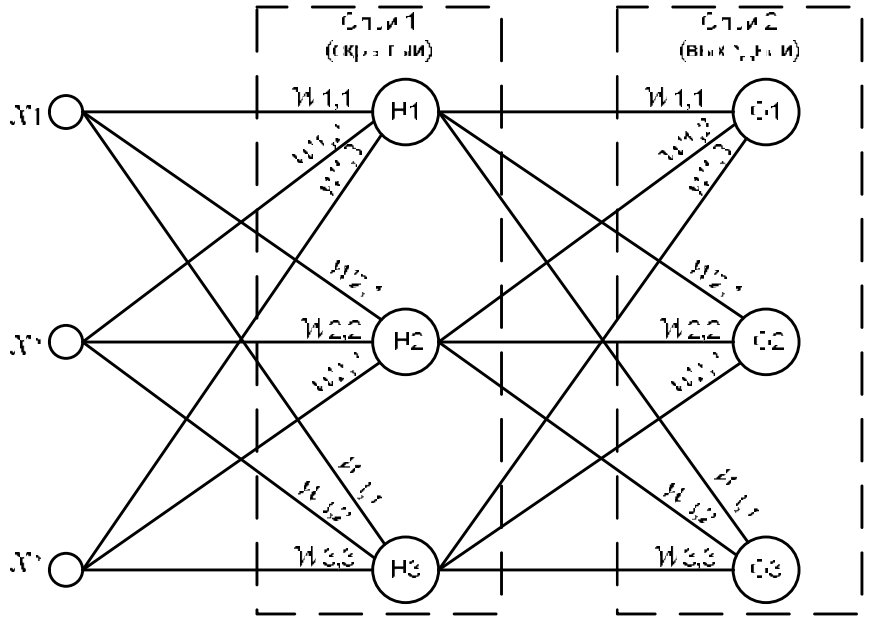


Рис. 4.1. Двухслойная нейронная сеть для реализации матричных операций

Для расчёта выходных значений скрытого слоя используется матрица весовых коэффициентов нейронов скрытого слоя **W1** и вектор входного сигнала **x**:

$$\begin{vmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{vmatrix} \times \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} x_1 \times w_{1,1} + x_2 \times w_{1,2} + x_3 \times w_{1,3} \\ x_1 \times w_{2,1} + x_2 \times w_{2,2} + x_2 \times w_{2,3} \\ x_1 \times w_{3,1} + x_2 \times w_{3,2} + x_3 \times w_{3,3} \end{vmatrix},$$

$$sigmoid\left(\begin{vmatrix} x_1 \times w_{1,1} + x_2 \times w_{1,2} + x_3 \times w_{1,3} \\ x_1 \times w_{2,1} + x_2 \times w_{2,2} + x_2 \times w_{2,3} \\ x_1 \times w_{3,1} + x_2 \times w_{3,2} + x_3 \times w_{3,3} \end{vmatrix} \right) = \begin{vmatrix} d_1^{(H1)} \\ d_2^{(H2)} \\ d_3^{(H3)} \end{vmatrix}$$

Для слоя 2 используется матрица весовых коэффициентов нейронов выходного слоя **W2**, а в качестве вектора входов – вектор, полученный на предыдущем шаге, т.е. вектор выходов нейронов скрытого слоя $\mathbf{d_H}$:

$$\begin{vmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{vmatrix} \times \begin{vmatrix} d_1^{(H1)} \\ d_2^{(H2)} \\ d_3^{(H3)} \end{vmatrix} = \begin{vmatrix} d_1^{(H1)} \times w_{1,1} + d_2^{(H2)} \times w_{1,2} + d_3^{(H3)} \times w_{1,3} \\ d_1^{(H1)} \times w_{2,1} + d_2^{(H2)} \times w_{2,2} + d_3^{(H3)} \times w_{2,3} \\ d_1^{(H1)} \times w_{3,1} + d_2^{(H2)} \times w_{3,2} + d_3^{(H3)} \times w_{3,3} \end{vmatrix},$$

$$sigmoid\left(\begin{vmatrix} d_1^{(H1)} \times w_{1,1} + d_2^{(H2)} \times w_{1,2} + d_3^{(H3)} \times w_{1,3} \\ d_1^{(H1)} \times w_{2,1} + d_2^{(H2)} \times w_{2,2} + d_3^{(H3)} \times w_{2,3} \\ d_1^{(H1)} \times w_{3,1} + d_2^{(H2)} \times w_{3,2} + d_3^{(H3)} \times w_{3,3} \end{vmatrix} \right) = \begin{vmatrix} d_1^{(O1)} \\ d_2^{(O2)} \\ d_3^{(O3)} \end{vmatrix}$$

В матричной форме обе эти операции записываются как:

$$\mathbf{W1} \times \mathbf{x} = \mathbf{d_H},$$
$$\mathbf{W2} \times \mathbf{d_H} = \mathbf{d_O}.$$

Пример. Зададим случайные значения весовым коэффициентам нейронов скрытого и выходного слоёв, т.е. элементам матриц **W1**, **W2**. Зададим входной вектор **x** и соответствующий ему выходной вектор **y**:

$$\mathbf{W1}=\begin{vmatrix}0.90 & 0.30 & 0.40\\ 0.20 & 0.80 & 0.20\\ 0.10 & 0.50 & 0.60\end{vmatrix}\quad \mathbf{W2}=\begin{vmatrix}0.30 & 0.70 & 0.50\\ 0.60 & 0.50 & 0.20\\ 0.80 & 0.10 & 0.90\end{vmatrix},\quad \mathbf{x}=\begin{vmatrix}0.900\\ 0.100\\ 0.800\end{vmatrix}\quad \mathbf{y}=\begin{vmatrix}0.350\\ 0.950\\ 0.480\end{vmatrix}$$

Дважды выполним матричный расчет выходов скрытого и выходного слоя:

$$\mathbf{W1}\times\mathbf{x}=\begin{vmatrix}0.90 & 0.30 & 0.40\\ 0.20 & 0.80 & 0.20\\ 0.10 & 0.50 & 0.60\end{vmatrix}\times\begin{vmatrix}0.900\\ 0.100\\ 0.800\end{vmatrix}=\begin{vmatrix}1.160\\ 0.420\\ 0.620\end{vmatrix}\quad \mathbf{d_H}=sigmoid\left(\begin{vmatrix}1.160\\ 0.420\\ 0.620\end{vmatrix}\right)=\begin{vmatrix}0.761\\ 0.603\\ 0.650\end{vmatrix}$$

$$\mathbf{W2}\times\mathbf{d_H}=\begin{vmatrix}0.30 & 0.70 & 0.50\\ 0.60 & 0.50 & 0.20\\ 0.80 & 0.10 & 0.90\end{vmatrix}\times\begin{vmatrix}0.761\\ 0.603\\ 0.650\end{vmatrix}=\begin{vmatrix}0.976\\ 0.889\\ 1.255\end{vmatrix}\quad \mathbf{d_O}=sigmoid\left(\begin{vmatrix}0.976\\ 0.889\\ 1.255\end{vmatrix}\right)=\begin{vmatrix}0.726\\ 0.709\\ 0.778\end{vmatrix}$$

Разница **e** между желаемыми (**y**) и полученным ($\mathbf{d_O}$) выходом:

$$\mathbf{e}=\mathbf{y}-\mathbf{d_O}=\begin{vmatrix}0.350\\ 0.950\\ 0.480\end{vmatrix}-\begin{vmatrix}0.726\\ 0.709\\ 0.778\end{vmatrix}=\begin{vmatrix}-0.376\\ 0.241\\ -0.298\end{vmatrix}$$

Ошибка сети:

$$\mathbf{E_{MSE}}=\frac{1}{3}\sum_{\mathrm{i}}e_i^2=\frac{(-0.376)^2+0.241^2+(-0.298)^2}{3}=0.096$$

С помощью полученной разности между желаемыми и реальными значениями выходов надо откалибровать весовые коэффициенты для улучшения нейронной сети.

Вычисление дельт для нейронов выходного слоя (см. формулу 3.10):

$$\boldsymbol{\delta}_\mathbf{O}=(\mathbf{y}-\mathbf{d_O})*f'(\mathbf{d_O})=\mathbf{e}*f'(\mathbf{d_O})=\begin{vmatrix}-0.376\\ 0.241\\ -0.298\end{vmatrix}*\begin{vmatrix}(0.726*(1-0.726))\\ (0.709*(1-0.708))\\ (0.778*(1-0.778))\end{vmatrix}=\begin{vmatrix}-0.075\\ 0.050\\ -0.051\end{vmatrix}$$

Вычисление дельт для нейронов скрытого слоя отличается отсутствием разности между выходными значениями этих нейронов и желаемыми значениями, поскольку желаемых значений нет и не может быть в принципе. Вычисление дельт производится по формуле (3.12). В матричных вычислениях первый член формулы представляет собой транспонированную матрицу весовых коэффициентов **W2**, умноженную на вектор дельт выходного слоя. Второй член формулы – по-прежнему производная функции активации от выходного значения скрытого нейрона:

$$\boldsymbol{\delta_H} = \left(\mathbf{W2}^{\mathrm{T}} \times \boldsymbol{\delta}_{\mathrm{O}}\right) * f'(\mathbf{d_H}) =$$

$$= \left(\begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix}^{\mathrm{T}} \times \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \right) * \begin{vmatrix} 0.761*(1-0.761) \\ 0.603*(1-0.603) \\ 0.650*(1-0.650) \end{vmatrix} = \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix}.$$

Вычисление градиента для каждого входа нейронов выходного и скрытого слоев осуществляется единообразно по формуле (3.13). Элементы матрицы градиентов выходного слоя определяются на основе векторов $\boldsymbol{\delta_O}$ и $\mathbf{d_O}$ следующим образом:

$$\mathbf{GRAD}_{\mathbf{W2}} = \boldsymbol{\delta_O} * \mathbf{d}_{\mathrm{H}}^{\mathrm{T}} =$$

$$= \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \times \begin{vmatrix} 0.761 \\ 0.603 \\ 0.650 \end{vmatrix}^{\mathrm{T}} = \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \times \begin{vmatrix} 0.761 & 0.603 & 0.650 \end{vmatrix} = \begin{vmatrix} -0.057 & -0.045 & -0.049 \\ 0.038 & 0.030 & 0.032 \\ -0.039 & -0.031 & -0.033 \end{vmatrix}.$$

Элементы матрицы градиентов скрытого слоя определяются на основе векторов $\boldsymbol{\delta_H}$ и $\mathbf{x}$:

$$\mathbf{GRAD}_{\mathbf{W1}} = \boldsymbol{\delta_H} * \mathbf{x}^{\mathrm{T}} =$$

$$= \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix} \times \begin{vmatrix} 0.900 \\ 0.100 \\ 0.800 \end{vmatrix}^{\mathrm{T}} = \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix} \times \begin{vmatrix} 0.900 & 0.100 & 0.800 \end{vmatrix} = \begin{vmatrix} -0.006 & -0.001 & -0.005 \\ -0.007 & -0.001 & -0.006 \\ -0.015 & -0.002 & -0.013 \end{vmatrix}.$$

Для вычисления корректирующих значений весовых коэффициентов зададим параметр скорости обучения $\eta = 0{,}7$. Воспользуемся формулой:

$$\mathbf{\Delta W} = \mathbf{GRAD_W} * \eta + \mathbf{\Delta W}(t-1) * \alpha . \qquad (4.1)$$

Но поскольку предыдущих итераций не было, о чём свидетельствует аргумент *t*-1, и предыдущие корректирующие значения еще не вычислялись, второе слагаемое с моментом α в формуле (4.1) будет нулевым.

$$\mathbf{\Delta_{W2}} = \begin{vmatrix} -0.057 & -0.045 & -0.049 \\ 0.038 & 0.030 & 0.032 \\ -0.039 & -0.031 & -0.033 \end{vmatrix} * 0{,}7 = \begin{vmatrix} -0.040 & -0.032 & -0.034 \\ 0{,}027 & 0{,}021 & 0.023 \\ -0.027 & -0.022 & -0{,}023 \end{vmatrix}$$

$$\mathbf{\Delta_{W1}} = \begin{vmatrix} -0.006 & -0.001 & -0.005 \\ -0.007 & -0.001 & -0.006 \\ -0.015 & -0.002 & -0.013 \end{vmatrix} * 0{,}7 = \begin{vmatrix} -0.004 & 0.000 & -0.003 \\ -0.005 & -0.001 & -0{,}004 \\ -0.011 & -0.001 & -0.009 \end{vmatrix}$$

Новые значения весовых коэффициентов получаются путем прибавления к старым значениям вычисленных поправочных значений:

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \mathbf{\Delta_W} \qquad (4.2)$$

$$\mathbf{W2} = \begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix} + \begin{vmatrix} -0.040 & -0.032 & -0.034 \\ 0{,}027 & 0{,}021 & 0.023 \\ -0.027 & -0.022 & -0{,}023 \end{vmatrix} = \begin{vmatrix} 0.260 & 0.668 & 0.466 \\ 0.627 & 0.521 & 0.223 \\ 0.773 & 0.078 & 0.877 \end{vmatrix}$$

$$\mathbf{W1} = \begin{vmatrix} 0.90 & 0.30 & 0.40 \\ 0.20 & 0.80 & 0.20 \\ 0.10 & 0.50 & 0.60 \end{vmatrix} + \begin{vmatrix} -0.004 & 0.000 & -0.003 \\ -0.005 & -0.001 & -0{,}004 \\ -0.011 & -0.001 & -0.009 \end{vmatrix} = \begin{vmatrix} 0.896 & 0.000 & 0.397 \\ 0.195 & 0.799 & 0.196 \\ 0.089 & 0.499 & 0{,}591 \end{vmatrix}$$

Полученные значения весовых коэффициентов были использованы для того же самого входного вектора. При этом величина ошибки сети E_{MSE}=0.089 уменьшилась по сравнению с первоначальным значением (0.096).

Матричная запись алгоритма более компактна, так как она не зависит от размера сети. Что еще более важно, некоторые языки программирования могут работать с матрицами. Они могут эффективно и быстро посчитать произведение матриц вместо того, чтобы использовать циклы (*python*, встроенный язык *MatLab* и др.).

Вопросы для самоконтроля

1. Какие преимущества имеет матричный способ реализации алгоритма обратного распространения?
2. Чем отличаются операции «×» и «*» применительно к матрицам?
3. Где и почему в алгоритме используется операция транспонирования матриц?
4. Какие изменения произойдут в матричном алгоритме при изменении числа нейронов во входном слое, в скрытом слое, в выходном слое?
5. Как изменится алгоритм при добавлении второго скрытого слоя?

Тема 5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Библиотека линейной алгебры *numpy* языка *python* является очень удобным инструментом для реализации алгоритмов в области научных вычислений. Её *универсальные функции* предоставляют отличные решения для обучения нейронных сетей. Универсальная функция – это оболочка над обычной функцией, благодаря которой возможно выполнять определенные действия над скалярами, векторами и над целыми массивами. Данные функции выполняют поэлементные действия над массивами, поддерживают механизм транслирования, автоматически преобразуют типы данных и обеспечивают доступ к более тонким настройкам своей работы.

Ниже представлен список функций с кратким описанием в том контексте, в котором они используются в двух примерных программах:

- функция *array*() инициализирует массив входных данных в виде *numpy*-матрицы;
- функция *dot*() реализует скалярное произведение векторов, умножение матрицы на вектор, умножение матрицы на матрицу;
- функция *exp*() вычисляет экспоненту всех элементов массива;
- функция *square*() вычисляет квадрат элементов массива, т.е. каждый элемент массива умножается сам на себя;
- функция *subtract*() вычисляет поэлементную разность значений двух массивов;
- функция *mean*() вычисляет среднее арифметическое значений элементов массива.

Используемые операторы:

- оператор «.T» транспонирует *numpy*-матрицу;
- операторы «+, -, /» выполняют поэлементные операции сложения, вычитания и деления элементов векторов и матриц;
- оператор «*» выполняет поэлементные операции умножения векторов и матриц или умножение элементов векторов и матриц на скаляр.

Первая программа реализует матричный алгоритм обучения нейронной сети прямого распространения сигнала на обучающей выборке, содержащей три элемента:

входной вектор (**x**)	выходной вектор (**y**)
0.90	0.35
0.10	0.95
0.80	0.48

Архитектура сети и весовые коэффициенты матриц скрытого и выходного слоя совпадают с коэффициентами матриц из примера ручного расчёта (см. Тему 4). Каждое действие программы сопровождается выводом промежуточных результатов на экран. Все векторы для лаконичного представления выводятся транспонированными, т.е. в виде горизонтальной строки. Матрицы выводятся в традиционном представлении.

Программа_1.

```
import numpy as np

#================== ИСХОДНЫЕ ДАННЫЕ =================
# произвольные значения для входного и выходного векторов
x = np.array([[0.90, 0.10, 0.80]]).T
y = np.array([[0.35, 0.95, 0.48]]).T
print('====================================')
print('x: ',x.T)     #транспонируем вектор для лаконичного
print('y: ',y.T)     #вывода на экран **

# задание произвольных значений для весовых коэффициентов
# W1-Матрица весовых коэффициентов первого (скрытого) слоя
# W2-Матрица весовых коэффициентов второго (выходного) слоя
W1 = np.array([[0.9,0.3,0.4],[0.2,0.8,0.2],[0.1,0.5,0.6]])
W2 = np.array([[0.3,0.7,0.5],[0.6,0.5, 0.2],[0.8,0.1,0.9]])
print('W1:')
print(W1)
print('W2:')
print(W2)
```

```
# параметр скорости обучения
learn_rate =0.7
print('=============================================')

#=================== ПРЯМОЙ ПРОХОД ==================
#расчёт выходных сигналов
L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
print('L1:',L1.T)                   #Выходы первого слоя
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
print('L2:',L2.T)                   #Выходы второго слоя

#расчёт средневадратической ошибки
MSE1=np.square(np.subtract(y, L2)).mean()
print('\nMSE =', MSE1)
print('=============================================')

#=================== ОБРАТНЫЙ ПРОХОД ===============
#расчет абсолютной ошибки
Eabs=y-L2
print('E_abs:', Eabs.T)             #**

# расчет производные вых. сигналов второго и первого слоя
Deriv2=L2*(1-L2) #здесь 1 – единичная матрица
Deriv1=L1*(1-L1)

#расчет дельт второго и первого слоя
L2delta=Eabs*Deriv2
L1delta= np.dot( W2.T,L2delta,)*Deriv1
print('\nDelta_L2:', L2delta.T)      #**
print('Delta_L1:', L1delta.T)       #**

#расчет градиентов второго и первого слоя
GR2=L2delta.dot(L1.T)
GR1=L1delta.dot(x.T)
print('\nGrad_2:')
print(GR2.T)                         #**
print('Grad_1:')
print(GR1.T)                        #**

#расчет поправочных значений для весовых коэффициентов
deltaW2 = GR2*learn_rate
deltaW1 = GR1*learn_rate
```

```
print('\ndelta_W2:')
print(deltaW2.T)                        #**
print('delta_W1:')
print(deltaW1.T)                        #**

#расчет новых весовых коэффициентов
W1=W1+deltaW1
W2=W2+deltaW2
print('\nW1_new:')
print(W1)
print('W2_new:')
print(W2)

# ======= ПРОВЕРКА ОШИБКИ ПОСЛЕ 1-ГО ПРОХОДА =========
L1 = 1/(1+np.exp(-np.dot( W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE2=np.square(np.subtract(y, L2)).mean()
print('\nMSE_new =', MSE2)
print('=========================================')
```

Результат работы программы

На рис. 5.1 представлены входные и выходные векторы обучающей выборки и матрицы весовых коэффициентов первого (скрытого) и второго (выходного) слоя.

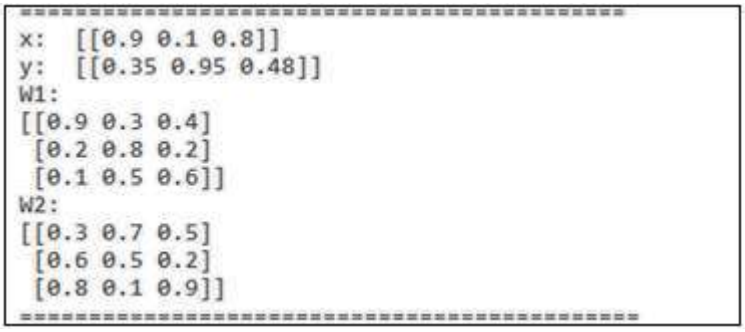


Рис. 5.1. Исходные данные алгоритма

Во время прямого прохода трех входных элементов обучающей выборки вычисляются выходные значения нейронов скрытого слоя L1 и нейронов выходного слоя L2. На основе реальных (L2) и желаемых (y) выходных значений вычисляется среднеквадратическая ошибка *MSE* на всей эпохе. Дополнительно выводятся абсо-

лютные ошибки сети для каждого из трех элементов обучающей выборки E_abs (рис. 5.2).

```
L1: [[0.76133271 0.60348325 0.65021855]]
L2: [[0.72630335 0.70859807 0.77809706]]

MSE = 0.09624698500823996
==========================================
E_abs: [[-0.37630335  0.24140193 -0.29809706]]
```

Рис. 5.2. – Результаты прямого прохода сигналов

На этапе обратного прохода происходит вычисление дельт каждого слоя сети Delta_L2 и Delta_L1. Эти вектора используются для расчёта матриц градиентов каждого синапса Grad_2 и Grad_1. Поскольку в каждом слое сети по три нейрона, а у каждого нейрона три входа, матрицы градиентов, как и матрицы весовых коэффициентов, имею размерности 3×3. Подстроечные значения весовых коэффициентов delta_W2 и delta_W1 получаются из матриц Grad_2 и Grad_1 умножением на константу скорости обучения learn_rate. Результаты промежуточных расчетов представлены на рисунке 5.3.

```
Delta_L2: [[-0.07480414  0.04984632 -0.05147004]]
Delta_L1: [[-0.00612519 -0.00779772 -0.01677461]]

Grad_2:
[[-0.05695084  0.03794964 -0.03918583]
 [-0.04514304  0.03008142 -0.03106131]
 [-0.04863904  0.032411   -0.03346678]]
Grad_1:
[[-0.00551267 -0.00701795 -0.01509715]
 [-0.00061252 -0.00077977 -0.00167746]
 [-0.00490015 -0.00623818 -0.01341969]]

delta_W2:
[[-0.03986559  0.02656475 -0.02743008]
 [-0.03160013  0.02105699 -0.02174292]
 [-0.03404733  0.0226877  -0.02342674]]
delta_W1:
[[-0.00385887 -0.00491256 -0.01056801]
 [-0.00042876 -0.00054584 -0.00117422]
 [-0.00343011 -0.00436672 -0.00939378]]
```

Рис. 5.3. Промежуточные результаты обратного прохода

Окончательные результаты обратного прохода – матрицы пересчитанных значений весов W1_new и W2_new. Их значения представлены на рис. 5.4. Для доказательства эффективности алго-

ритма производится прямой проход и вычисление ошибки сети с новыми весовыми коэффициентами. Уменьшенное значение ошибки MSE_new также выводится на экран.

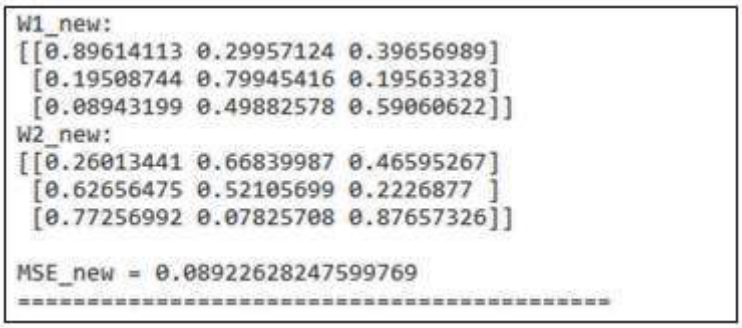


Рис. 5.4. Окончательные результаты обратного прохода и проверка эффективности произведенных вычислений

Вторая программа демонстрирует работу сети для трехмерных входных и выходных векторов. Обучение реализовано за счёт выполнения *N*=5000 циклов обучения. Печать промежуточных вычислений отсутствует. На экран выводятся только значения ошибок до и после обучения.

Для сокращения объёма кода во второй программе удалено вычисление ряда промежуточных переменных (векторов Eabs, Deriv1,2). Их расчёт перенесен в расчеты для основных переменных. Добавлено использование момента (momentum=0.3) при вычислении подстроечных значений коэффициентов. С этой цель введены массивы deltaW1_old и deltaW2_old. В них сохраняются значения delta_W1 и delta_W2 после каждой подстройки весовых коэффициентов сохраняются в массивах для использования на следующем шаге.

Программа выводит значения входного и выходного векторов обучающей выборки Input и Output, два значения ошибки – до и после обучения сети MSE и MSE_new, и значения, выдаваемые сетью на последнем шаге обучения Neural_Net_Output (рис. 5.5).

Программа_2.

```
import numpy as np
#================== ИСХОДНЫЕ ДАННЫЕ =================
# задание произвольных значений для входного (x)
# и выходного (y) векторов
x = np.array([[1, 0, 1], [0, 1, 1], [1, 1, 1]])
y = np.array([[1, 1, 1], [0, 0, 1], [1, 1, 0]])
# задание случайных значений для весовых коэффициентов
W1 = 2*np.random.random((3, 3))-1
W2 = 2*np.random.random((3, 3))-1
# параметры: скорость обучения и момент
learn_rate=0.7
momentum=0.3
# число эпох
N=5000

#расчёт выходных сигналов для вычисления начальной ошибки
L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE1=np.square(np.subtract(y, L2)).mean()
print('\nMSE =', MSE1)

deltaW2_old = 0
deltaW1_old=0

for j in range(N):
#=================== ПРЯМОЙ ПРОХОД ====================
#расчёт выходных сигналов
    L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
    L2 = 1/(1+np.exp(-np.dot( W2,L1)))

#=================== ОБРАТНЫЙ ПРОХОД ====================
#расчет дельт второго и первого слоя
    L2delta=(y-L2)*(L2*(1-L2))
    L1delta= np.dot( W2.T,L2delta,)*L1*(1-L1)
#расчет градиентов второго и первого слоя
    GR2=L2delta.dot(L1.T)
    GR1=L1delta.dot(x.T)
#расчет поправочных значений для весовых коэффициентов
    deltaW2 = GR2*learn_rate + deltaW2_old*momentum
    deltaW1 = GR1*learn_rate + deltaW1_old*momentum
```

```
#расчет новых весовых коэффициентов
    W1=W1+deltaW1
    W2=W2+deltaW2
#сохранение поправочных коэффициентов текущего шага
    deltaW2_old=deltaW2
    deltaW1_old=deltaW1

# ======= ПРОВЕРКА ОШИБКИ ПОСЛЕ ОБУЧЕНИЯ  =========
L1 = 1/(1+np.exp(-np.dot( W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE2=np.square(np.subtract(y, L2)).mean()

print('\nInput:')
print(x)
print('\nOutput:')
print(y)
print('\nNeural_Net_Output:')
print(L2)
print('\nCycles = ',N)
print('MSE_new = ',MSE2)
```

```
MSE = 0.2017459125377166
Input:
[[1 0 1]
 [0 1 1]
 [1 1 1]]
Output:
[[1 1 1]
 [0 0 1]
 [1 1 0]]
Neural_Net_Output:
[[0.99890246 0.99601532 0.99007858]
 [0.00897447 0.00681352 0.98491968]
 [0.99091445 0.99313959 0.01521282]]
Cycles =  5000
MSE_new = 9.232668049558024e-05
```

Рис. 5.5. Результаты работы программы_2

Анализ результатов показывает, что сеть научилась распознавать данные с высокой степенью точности. Среднеквадратическая ошибка обученной сети составляет 9.23e-5, а максимальная абсо-

лютная ошибка (разница между элементами матриц Output и Neural_Net_Output) составляет 1.5%.

Вопросы для самоконтроля

1. Дайте краткую характеристику языку *python*. В каких областях рекомендуется его применение?
2. Каково назначение библиотеки *numpy*? Чем оправдано её применение для обучения нейронных сетей?
3. Что представляют собой массивы *numpy*?
4. Из каких переменных программы_1 можно получить представление об архитектуре нейронной сети?
5. Поясните, как в программном коде вычисляется ошибке сети на эпохе.
6. Чем обусловлено добавление момента при обучении сети в программе_2?
7. Проведите эксперименты с программой, закомментировав участок кода, осуществляющий использование момента, и поясните разницу в полученных результатах.
8. Проведите сравнение результатов программы_2 при одном и том же числе циклов, но с разными значениями скорости обучения. Какие выводы Вы можете сделать?

Тема 6. СЕТИ КОХОНЕНА

Термин «сети Кохонена» (англ.: *Kohonen network, KCN*) был введен в 1982 г. финским ученым Тойво Кохоненом. Хотя изначально они применялись для обработки изображений и звука, в настоящее время сети Кохонена являются эффективным средством кластерного анализа, когда требуется обнаружить скрытые закономерности в больших массивах данных.

Сеть Кохонена состоит из узлов-нейронов, которые в результате обучения сдвигаются в области, где сосредоточены исследуемые объекты. На рисунке 6.1 представлена архитектура сети Кохонена, каждый нейрон которой имеет два входа – по числу признаков или свойств кластеризуемых объектов.

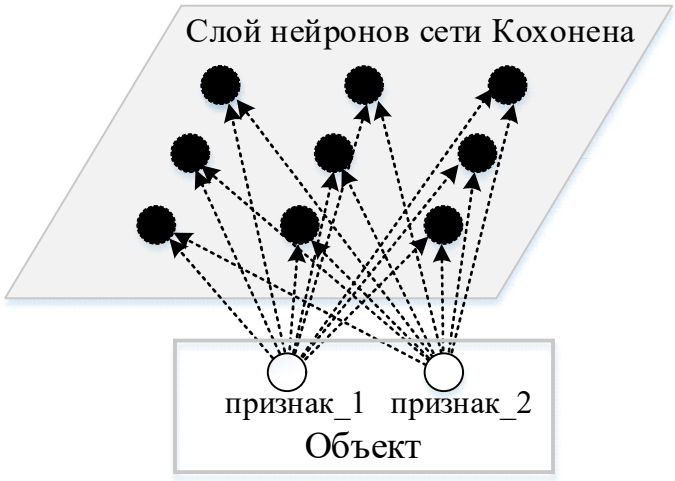


Рис. 6.1. Архитектура сети Кохонена

В отличие от большинства нейронных сетей, сеть Кохонена не имеет скрытых слоев: данные с входного слоя передаются непосредственно на единственный и он же выходной слой сети. Как и в обычной нейронной сети, элементы входного слоя не являются нейронами. Их задача просто передать значения входных полей исходной выборки на нейроны выходного слоя. Нейроны выходного слоя называются *кластерными элементами*, их количество определят максимальное количество групп, на которые сеть может разделить входные данные. Увеличивая количество нейронов, можно увеличивать детализацию результатов кластеризации. Исследователю следует знать, что могут иметь место ситуации, когда часть нейронов окажется незадействованной, либо несколько нейронов

диагностируют объекты одного и того же реально существующего кластера.

Каждая связь между входными элементами и нейронами сети имеет некоторый вес, который устанавливается в процессе инициализации. В большинстве случаев эти значения устанавливаются случайным образом, хотя существуют и другие способы инициализации. Значение весов, как правило, небольшие и лежат в интервале от 0 до 1.

Процесс обучения сети Кохонена, как и сетей прямой передачи сигнала, заключается в подстройке весов. В основе построения сети Кохонена лежит *конкурентное обучение*, при котором выходные узлы (нейроны) конкурируют между собой за право стать «победителем». Говорят, что нейроны *избирательно настраиваются* для различных входных примеров или классов входных примеров в ходе соревновательного процесса обучения. Побеждает тот нейрон, чей вектор весов ближе всего к входному вектору сигналов. Весовые коэффициенты нейрона-победителя подстраиваются таким образом, чтобы он стал ещё ближе к текущему объекту.

Исходные значения признаков объектов должны быть предварительно пронормированы. Это следует делать для того, чтобы признаки с большими абсолютными значениями не преобладали над признаками, величины которых малы.

Алгоритм обучения Кохонена

Пусть имеется набор из n экземпляров m-мерных векторов $\mathbf{x}$ исходной выборки. Каждый вектор $\mathbf{x}_i$ содержит оцифрованный набор признаков исследуемых объектов $\mathbf{x}_i=\{x_{i1}, x_{i2}, \ldots, x_{im}\}$, $i=1..n$. Векторы весов $\mathbf{w}_j$ каждого из q нейронов сети Кохонена тоже имеют размерность m: $\mathbf{w}_j=\{w_{1j}, w_{2j}, \ldots, w_{mj}\}$, $j=1..q$. За меру близости двух векторов можно взять квадрат евклидова расстояния:

$$\rho(\mathbf{x1}, \mathbf{x2}) = \sum_{j=1}^{m} (x1_j - x2_j)^2 . \qquad (6.1)$$

На каждом шаге обучения из исходного набора данных случайно выбирается один вектор. Затем производится поиск нейрона выходного слоя, для которого расстояние между его вектором весов и входным вектором минимально.

В обучении по Кохонену сам нейрон-победитель и нейроны, которые являются соседями нейрона-победителя, подстраивают свои веса, используя линейную комбинацию входных векторов и текущих векторов весов:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_{ni} - w_{ij}(t)), \qquad (6.2)$$

где $0<\eta\leq 1$ – коэффициент скорости обучения; индексы t, t+1 обозначают текущее и новое значение весового коэффициента.

Кохонен показал, что скорость обучения должна быть уменьшающейся функцией от числа итераций обучения. Согласно его рекомендациям, процесс обучения сети можно разделить на две фазы – фазу точной и фазу грубой подстройки. На первой фазе скорость обучения велика, и веса нейронов корректируются на большие величины, что позволяет приблизительно настроить их в соответствии с распределением значений признаков исследуемых объектов. Это так называемая грубая кластеризация. В процессе точной настройки скорость обучения резко уменьшается, что позволяет подстраивать веса на каждом шаге более точно.

По завершении эпохи вычисляется значение критерия останова алгоритма обучения. Если критерий не достигнут, то осуществляется переход к новой эпохе. В качестве критериев останова процесса обучения можно использовать следующие:

– количество полных циклов обучения ограничено константой, например, количество циклов равно количеству элементов во входном множестве;

– выход сети стабилизируется, т.е. входные вектора не переходят между кластерными элементами;

– изменения весов становятся незначительными.

В целях повышения эффективности алгоритма нейроны, которые слабо участвуют в конкуренции (не получают достаточного количества побед), могут быть сокращены.

Рассмотрим процесс обучения на примере сети Кохонена размером 2×2, предназначенной для кластеризации объектов с двумя признаками. Случайные начальные значения весовых коэффициентов:

$w_{1,1} = 0.9$ $w_{2,1} = 0.8$ (нейрон 1)
$w_{1,2} = 0.9$ $w_{2,2} = 0.2$ (нейрон 2)
$w_{1,3} = 0.1$ $w_{2,3} = 0.8$ (нейрон 3)
$w_{1,4} = 0.1$ $w_{2,4} = 0.2$ (нейрон 4)

В качестве объектов выступают абоненты, потребляющие трафик сети Интернет. Первый признак – возраст, второй – месячный объем трафика. Значения признаков пронормированы:

$x_{1,1} = 0.8$ $x_{1,2} = 0.8$ (пожилой человек, высокое потребление)
$x_{2,1} = 0.8$ $x_{2,2} = 0.1$ (пожилой человек, низкое потребление)
$x_{3,1} = 0.2$ $x_{3,2} = 0.9$ (молодой человек, высокое потребление)
$x_{4,1} = 0.1$ $x_{4,2} = 0.1$ (молодой человек, низкое потребление)

Для упрощения расчётов радиус обучения примем равным нулю ($\sigma = 0$), поэтому возможность подстройки весов будет предоставлена только нейрону-победителю. Коэффициент скорости обучения установим $h = 0{,}5$.

Вычисляем евклидово расстояние между входным вектором $\mathbf{x}_1 = (0.8; 0.8)$ и векторами весов всех четырех нейронов выходного слоя.

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_1) = \sqrt{(w_{11} - x_{11})^2 + (w_{21} - x_{12})^2} = \sqrt{(0.9 - 0.8)^2 + (0.8 - 0.8)^2} = 0.10 \quad \text{(min)}$$

$$D(\mathbf{w}_2, \mathbf{x}_1) = \sqrt{(w_{12} - x_{11})^2 + (w_{22} - x_{12})^2} = \sqrt{(0.9 - 0.8)^2 + (0.2 - 0.8)^2} = 0.61$$

$$D(\mathbf{w}_3, \mathbf{x}_1) = \sqrt{(w_{13} - x_{11})^2 + (w_{23} - x_{12})^2} = \sqrt{(0.1 - 0.8)^2 + (0.8 - 0.8)^2} = 0.70$$

$$D(\mathbf{w}_4, \mathbf{x}_1) = \sqrt{(w_{14} - x_{11})^2 + (w_{24} - x_{12})^2} = \sqrt{(0.1 - 0.8)^2 + (0.2 - 0.8)^2} = 0.92$$

Для первого объекта победителем стал нейрон с номером 1, для которого расстояние между вектором его весов $\mathbf{w}_1 = (0.9; 0.8)$ и входным вектором $\mathbf{x}_1 = (0.8; 0.8)$ оказалось минимальным. Видно, что вектор весов нейрона-победителя более «похож» на входной вектор $\mathbf{x}_1$, чем векторы весов других нейронов. Можно предположить, что нейрон 1 «открывает» кластер, в котором будут содержаться пожилые люди с высоким объёмом потреблённого трафика.

Подстройка:

Согласно формуле (6.2) для первого нейрона имеем:

$$w_{11}(t+1) = w_{11}(t) + 0.5 \times (x_{11} - w_{11}(t)) = 0.9 + 0.5 \times (0.8 - 0.9) = 0.85$$

$$w_{21}(t+1) = w_{21}(t) + 0.5 \times (x_{12} - w_{21}(t)) = 0.8 + 0.5 \times (0.8 - 0.8) = 0.8$$

Видим, что весовые коэффициенты нейронов «подталкиваются» в направлении значений входного вектора, т.е. w_{11} нейрона-победителя, соответствующий признаку «возраст», изначально был равен 0,9, но в результате подстройки был изменен в сторону нормированного значения «возраст» первого входного вектора, равного 0,8. Поскольку коэффициент скорости обучения $h = 0{,}5$, то величина подстройки равна половине расстояния между текущим весом и значением поля. Данная подстройка позволит нейрону 1 стать более «успешным» и в «захвате» других возможных объектов, содержащих информацию о пожилых людях с высокой интернет-активностью.

Второй входной вектор $\mathbf{x}_2 = (0.8; 0.1)$.

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_2) = \sqrt{(w_{11} - x_{21})^2 + (w_{21} - x_{22})^2} = \sqrt{(0.85 - 0.8)^2 + (0.8 - 0.1)^2} = 0.70$$

$$D(\mathbf{w}_2, \mathbf{x}_2) = \sqrt{(w_{12} - x_{21})^2 + (w_{22} - x_{22})^2} = \sqrt{(0.9 - 0.8)^2 + (0.2 - 0.1)^2} = 0.14 \text{ (min)}$$

$$D(\mathbf{w}_3, \mathbf{x}_2) = \sqrt{(w_{13} - x_{21})^2 + (w_{23} - x_{22})^2} = \sqrt{(0.1 - 0.8)^2 + (0.8 - 0.1)^2} = 0.99$$

$$D(\mathbf{w}_4, \mathbf{x}_2) = \sqrt{(w_{14} - x_{21})^2 + (w_{24} - x_{22})^2} = \sqrt{(0.1 - 0.8)^2 + (0.2 - 0.1)^2} = 0.71$$

Подстройка:

$$w_{12}(t+1) = w_{12}(t) + 0.5 \times (x_{21} - w_{12}(t)) = 0.9 + 0.5 \times (0.8 - 0.9) = 0.85$$

$$w_{22}(t+1) = w_{22}(t) + 0.5 \times (x_{22} - w_{22}(t)) = 0.2 + 0.5 \times (0.1 - 0.1) = 0.15$$

Третий входной вектор $\mathbf{x}_3 = (0.2; 0.9)$.

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_3) = \sqrt{(w_{11} - x_{31})^2 + (w_{21} - x_{32})^2} = \sqrt{(0.85 - 0.2)^2 + (0.8 - 0.9)^2} = 0.66$$

$$D(\mathbf{w}_2, \mathbf{x}_3) = \sqrt{(w_{12} - x_{31})^2 + (w_{22} - x_{32})^2} = \sqrt{(0.85 - 0.2)^2 + (0.15 - 0.9)^2} = 0.99$$

$$D(\mathbf{w}_3, \mathbf{x}_3) = \sqrt{(w_{13} - x_{31})^2 + (w_{23} - x_{32})^2} = \sqrt{(0.1 - 0.2)^2 + (0.8 - 0.9)^2} = 0.14 \quad \text{(min)}$$

$$D(\mathbf{w}_4, \mathbf{x}_3) = \sqrt{(w_{14} - x_{31})^2 + (w_{24} - x_{32})^2} = \sqrt{(0.1 - 0.2)^2 + (0.2 - 0.9)^2} = 0.71$$

Подстройка:

$$w_{13}(t+1) = w_{13}(t) + 0.5 \times (x_{31} - w_{13}(t)) = 0.1 + 0.5 \times (0.2 - 0.1) = 0.15$$
$$w_{23}(t+1) = w_{23}(t) + 0.5 \times (x_{32} - w_{23}(t)) = 0.8 + 0.5 \times (0.9 - 0.8) = 0.85$$

Четвёртый входной вектор $\mathbf{x}_4 = (0.1; 0.1)$.

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_4) = \sqrt{(w_{11} - x_{41})^2 + (w_{21} - x_{42})^2} = \sqrt{(0.85 - 0.1)^2 + (0.8 - 0.1)^2} = 1.03$$
$$D(\mathbf{w}_2, \mathbf{x}_4) = \sqrt{(w_{12} - x_{41})^2 + (w_{22} - x_{42})^2} = \sqrt{(0.85 - 0.1)^2 + (0.15 - 0.1)^2} = 0.75$$
$$D(\mathbf{w}_3, \mathbf{x}_4) = \sqrt{(w_{13} - x_{41})^2 + (w_{23} - x_{42})^2} = \sqrt{(0.15 - 0.1)^2 + (0.85 - 0.1)^2} = 0.75$$
$$D(\mathbf{w}_4, \mathbf{x}_4) = \sqrt{(w_{14} - x_{41})^2 + (w_{24} - x_{42})^2} = \sqrt{(0.1 - 0.1)^2 + (0.2 - 0.1)^2} = 0.10 \text{ (min)}$$

Подстройка:

$$w_{14}(t+1) = w_{14}(t) + 0.5 \times (x_{41} - w_{14}(t)) = 0.1 + 0.5 \times (0.1 - 0.1) = 0.1$$
$$w_{24}(t) = w_{24}(t) + 0.5 \times (x_{42} - w_{24}(t)) = 0.2 + 0.5 \times (0.1 - 0.2) = 0.15$$

В таблице 6.1 представлены значения весов нейронов до и после первой подстройки. Конечно, движение нейронов к объектам достаточно очевидно. Однако, пример хорошо иллюстрирует основы работы алгоритма обучения сети Кохонена с использованием конкуренции между нейронами (рис. 6.2).

Таблица 6.1. Значения весовых коэффициентов до и после одной эпохи обучения

	Начальные значения		Значения после 1-й эпохи	
	w_1	w_2	w_1	w_2
Нейрон 1	0.9	0.8	0,85	0,80
Нейрон 2	0.9	0.2	0,85	0,15
Нейрон 3	0.1	0.8	0,15	0,85
Нейрон 4	0.1	0.2	0,10	0,15

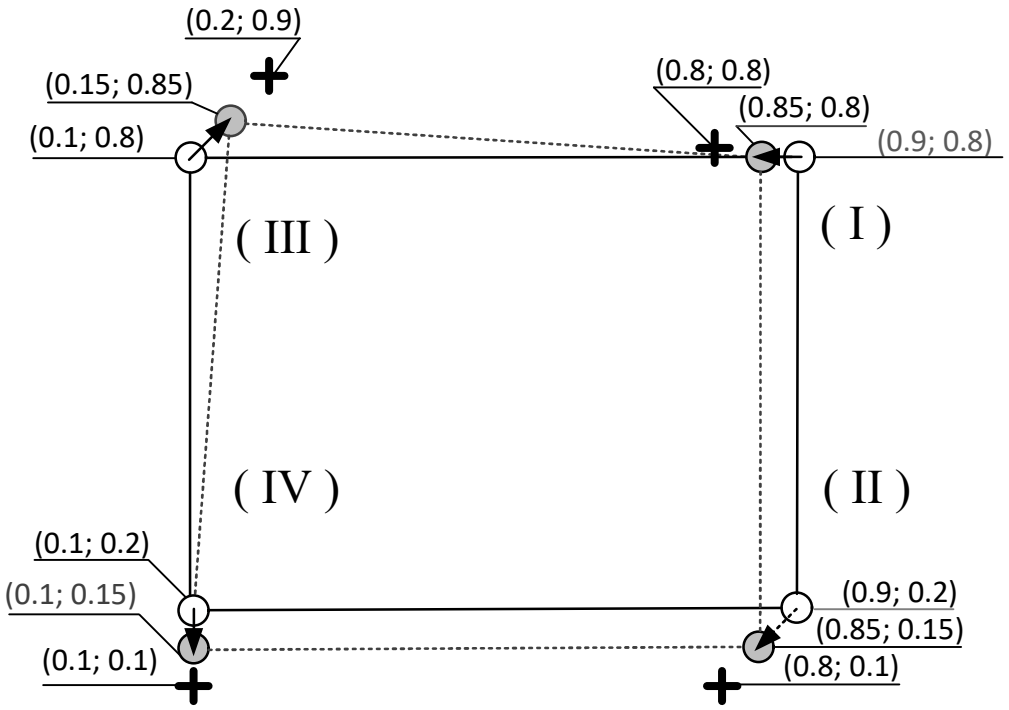


Рис. 6.2. Иллюстрация обучения сети Кохонена

Вопросы для самоконтроля

1. Каково принципиальное отличие в обучении сетей Кохонена от обучения сетей прямого распространения сигнала?
2. Для каких типов задач сети Кохонена являются наилучшим решением?
3. Какова архитектура сети Кохонена?
4. Что понимается под термином «нейрон-победитель»?
5. Опишите формально алгоритм обучения сети Кохонена.
6. Каковы параметры алгоритма? Каким образом они выбираются?
7. Какие фазы можно выделить в процессе обучения сети Кохонена?
8. С какой целью производится нормирование входных данных? Какие виды нормирования Вам известны?
9. Какие критерии останова могут использоваться для завершения алгоритма? Предложите свой критерий.

Тема 7. САМООРГАНИЗУЮЩИЕСЯ КАРТЫ КОХОНЕНА

Самоорганизующиеся нейронные карты (англ. s*elf organizing map*, *SOM*) или карты Кохонена – это разновидность нейронных сетей Кохонена. Они представляют собой не только эффективный инструмент кластеризации, но и не менее эффективный инструмент визуализации. В отличии от сети Кохонена, в которой число нейронов соответствует количеству формируемых кластеров, в *SOM* число нейронов заведомо больше предполагаемого числа кластеров.

Карты позволяют представлять результаты группировки объектов в виде двумерных карт, где расстояния между объектами соответствуют расстояниям между их векторами в многомерном пространстве, а сами значения признаков отображаются различными цветовыми оттенками.

Карта Кохонена состоит из сегментов прямоугольной или шестиугольной формы, называемых ячейками. Каждая ячейка связана с определенным нейроном и представляет собой «сферу влияния» или «область притяжения» данного нейрона. Шестиугольные ячейки позволяют более корректно отражать расстояния между объектами на карте, поскольку в этом случае расстояния между центрами смежных ячеек одинаковы, т.е. расстояние не зависит от направления (рис. 7.1). Чем больше число сегментов карты или её размер, тем более подробно можно представить распределение признаков объектов. Можно вернуться к обычной сети Кохонена, уменьшив число сегментов карты до нужного количества кластеров.

Подстройка весовых коэффициентов нейронов *SOM* получается так же, как и в обычной сети Кохонена, т.е. на основе конкурирующего обучения. Объекты, векторы признаков которых оказываются ближе к вектору весов данного нейрона, попадают в связанную с ним ячейку. Распределение объектов на карте в целом соответствует распределению векторов весов нейронов. И если объекты на карте расположены близко друг к другу, то и векторы признаков этих объектов близки и наоборот, если ячейки с объектами находятся далеко друг от друга, то и векторы их признаков различаются сильно.

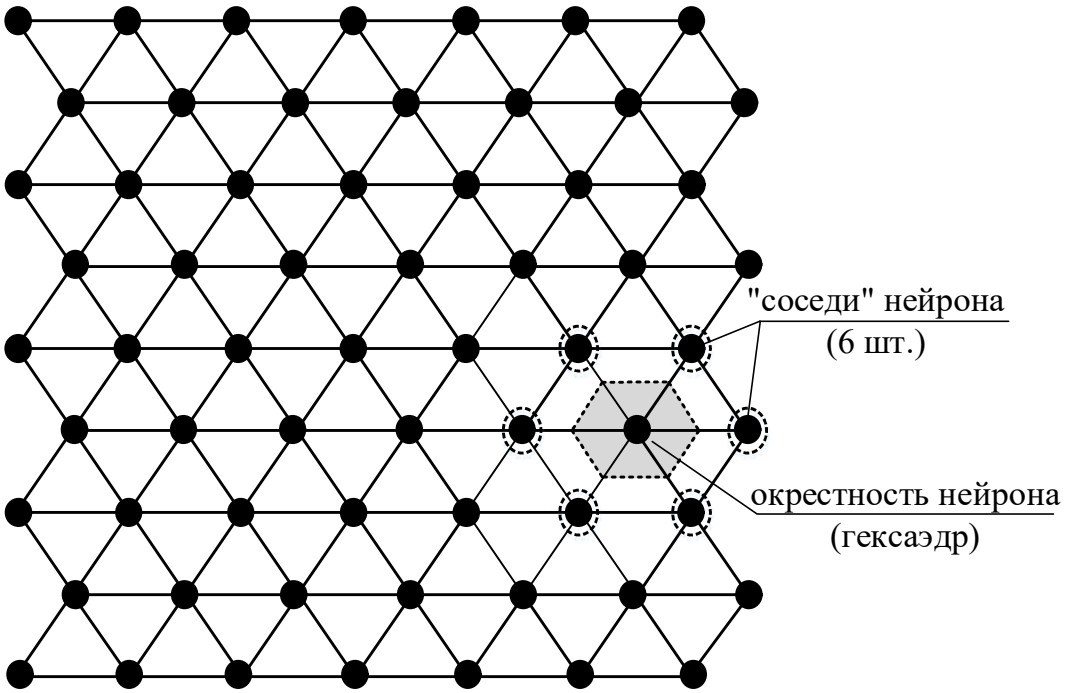


Рис. 7.1. Самоорганизующаяся карт размером 9×7

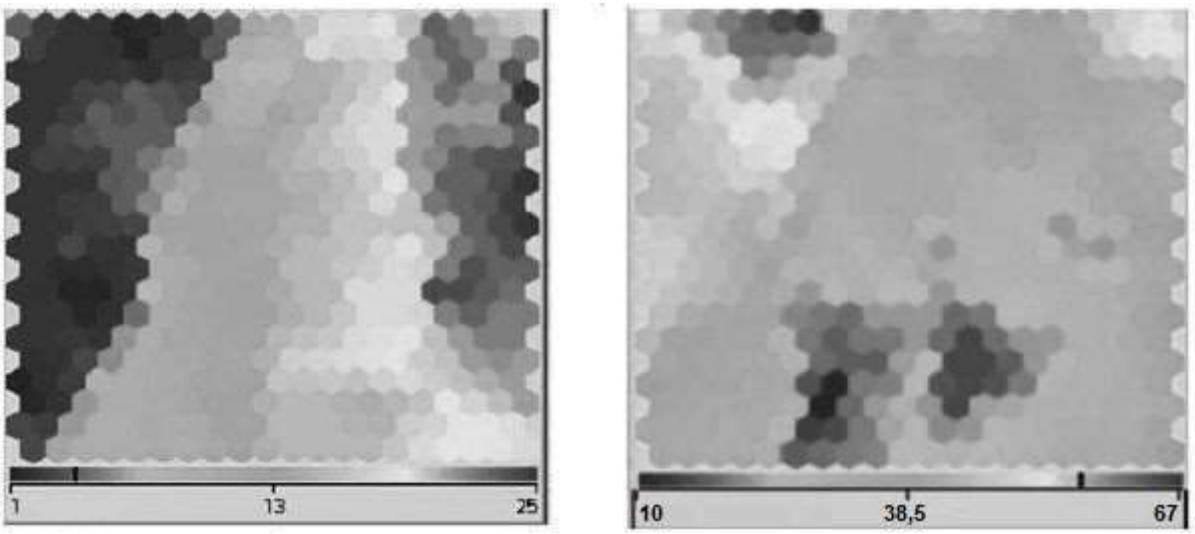
Отображение результатов обучения

Отобразить *SOM* целиком нельзя, поскольку её узлы-нейроны представляют собой точки в *n*-мерном признаковом пространстве. Но можно построить отдельные двумерные раскрашенные карты для каждого признака. Совокупность таких карт даёт интегральную картину, характеризующую набор исследуемых объектов. Специальная раскраска *SOM* выполняет функции третьего измерения и помогает понять, как распределены признаки объектов. Каждой ячейке (нейрону) на раскрашенной карте назначается цвет в соответствии со значениями признаков, попавших в неё объектов.

Раскраску карт можно осуществить разными способами. Например, если в ячейку попал только один объект, то её цвет будет определяться по одному значению признака. Если таких объектов несколько, цвет ячейки будет пропорционален среднему значению признака. Если центр ячейки – «мертвый» нейрон, для вектора весов которого не нашлось ни одного достаточно близкого вектора признаков объекта при обучении карты, то в ячейку не попадет ни одного объекта, и ее расцветка может формироваться на основе соответствующего веса нейрона, поскольку в процессе обучения нейрон наверняка «подтягивался» в ту или иную сторону и резкого

различия между цветом пустой ячейки и цветом соседних ячеек может и не быть. Либо у пустых ячеек выделяются границы, а сами ячейки заполняются какой-либо штриховкой (резко выделяются).

Раскраска карт позволяет оценивать и сами результаты кластеризации. Если ячейки с примерно одинаковой расцветкой образуют обособленные области, то результаты кластеризации хорошие, т.е. алгоритму удалось выделить группы объектов с похожими признаками. Если ячейки с разными цветами разбросаны вперемешку по всей карте, то результаты плохие. Хотя возможна ситуация, когда по одним признакам объекты различаются хорошо, а по другим хуже. Это может привести к тому, что карты, построенные по одним признакам, которые для разных групп объектов различаются хорошо, покажут хорошую кластеризацию, а по другим, которые различаются хуже, – плохую (рис. 7.2).



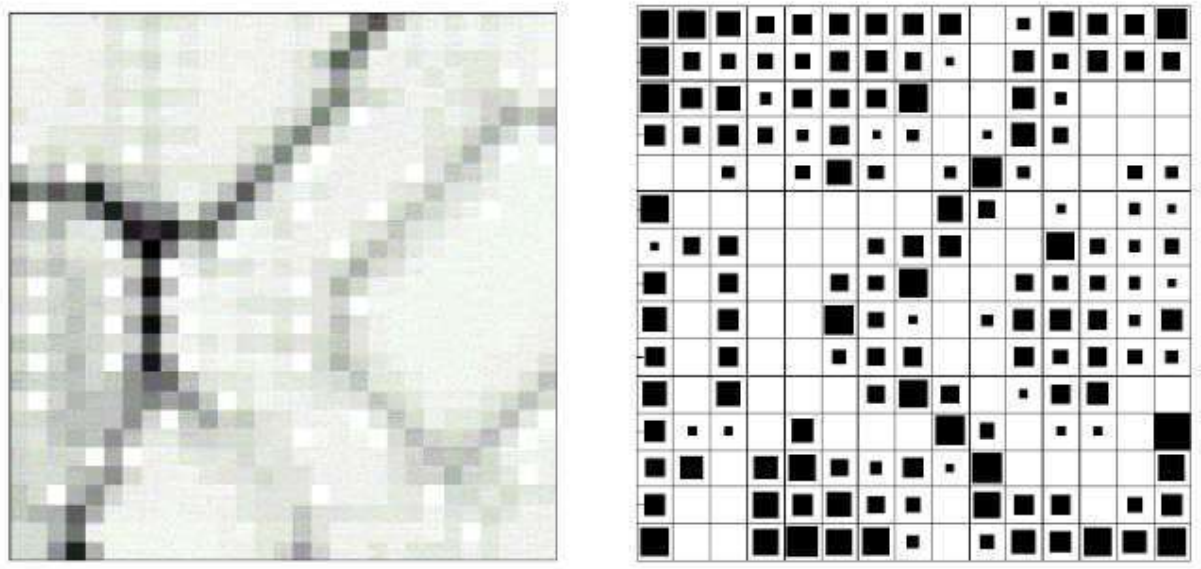
а *б*

Рис. 7.2. Примеры удачной (*а*) и неудачной (*б*) раскрасок

Автоматическое выделение кластеров на обученной карте может производиться на основе построения *унифицированной матрицы расстояний* (*U*-матрица), которая строится в оттенках серого и отображает расстояния между соседними узлами карты. В этом методе рассчитанное по евклидовой метрике расстояние между соседними нейронами представляется в виде различной раскраски между соседними вершинами. Темная раскраска соответствует большему расстоянию между нейронами, а светлая свидетельствует о близком соседстве (рис. 7.3, *а*). Светлые области могут считаться

кластерами, а темные – разделителями. Это может быть полезным представлением при попытке найти кластеры во входных данных без наличия априорной информации о кластерах.

Ещё одним популярным способом визуализации является *диаграмма Хинтона*. На каждом узле сетки изображается квадрат, размер которого пропорционален числу точек, ближайших к данному узлу, а оттенок соответствует значению соответствующего отображаемого признака (рис. 7.3 *б*). Чем больше точек попадает в узел, тем больше соответствующий размер квадрата. Оттенок квадрата соответствует значению цвета на матрице расстояний для данного нейрона. Таким образом диаграмма Хинтона соответствует матрице плотности попадания объектов в узлы карты.



а *б*

Рис. 7.3. Пример *U*-матрицы (*а*) и диаграммы Хинтона (*б*) для карт с прямоугольными ячейками.

Частота попадания объектов в область определённого нейрона карты может отображаться и на каждой эпохе обучения. Такие карты по своему виду схожи с диаграммами Хинтона, однако они отображают число реагирований нейронов-победителей на входные данные на каждой эпохе и в совокупности позволяют отслеживать динамику активности нейронов.

Выбор числа нейронов карты

Выбор числа ячеек в *SOM* зависит от особенностей решаемой задачи. Если нужна детальная информация по каждому объекту и его признакам, то количество нейронов карты должно приближаться к числу объектов выборки. Но увеличение числа нейронов карты приводит к росту вычислительных и временных затрат на ее обучение. В большинстве случаев для анализа представляют интерес обобщенные данные, когда сделанные на их основе выводы и заключения можно распространить на группы объектов. В этом случае нужно, чтобы число нейронов было в несколько раз меньше числа кластеризуемых объектов. Строгих правил для определения числа нейронов *SOM* не существует. Пользователь должен сам определить (эмпирически) оптимальное количество нейронов исходя из решаемой задачи и особенности анализируемых данных. Например, если разброс значений признаков в обучающей выборке сильный (т.е. векторы объектов в пространстве признаков разрежены), то, возможно, следует уменьшить число нейронов карты, чтобы избежать большого количества пустых ячеек. И наоборот, если векторы объектов расположены в пространстве признаков плотно, то для получения лучших результатов можно увеличить число нейронов.

Недостатки SOM

Одним основных недостатков является эвристический характер метода. В большинстве случаев начальная инициализация карты, т.е. задание начальных векторов весов нейронов, является произвольной и тогда возможна потеря однозначности результата. Если обучать карту несколько раз, то итоги будут различаться из-за того, что устанавливаемые случайным образом начальные веса нейронов будут различны и обучение пойдет несколько по-иному.

Второй недостаток – «мертвые» нейроны, веса которых были проинициализированы таким образом, что их векторы оказались в той части пространства признаков, где отсутствуют или почти отсутствуют векторы признаков объектов. Большое число мёртвых нейронов может привести к выполнению обработки входных данных меньшим числом нейронов, что ухудшит результаты кластеризации. В карте может не хватить активных нейронов для достоверной интерпретации исследуемых объектов.

Для устранения указанных недостатков, можно, во-первых, инициализировать карту не случайными весами, а векторами признаков объектов обучающей выборки, во-вторых, подсчитывать количество «побед» каждого нейрона с целью учета его активности. Если это число превышает некоторое значение (т.е. нейрон слишком активен), то чтобы дать возможность другим тоже стать победителями, он штрафуется искусственным завышением расстояния между вектором его весов и вектором весов объектов. При этом величина «штрафа» пропорциональна числу побед нейрона. После того как в процесс обучения вовлекается достаточное количество нейронов, можно продолжить конкуренцию без регуляризации, т.е. без штрафов.

Обучение SOM

В отличие от сети Кохонена, при настройке *SOM* активизируется не только нейрон-победитель, но и его соседи. Победивший нейрон становится центром некоторого множества «нейронов-соседей». Все нейроны такого соседства тоже обладают правом подстройки весов в сторону текущего объекта (рис. 7.4).

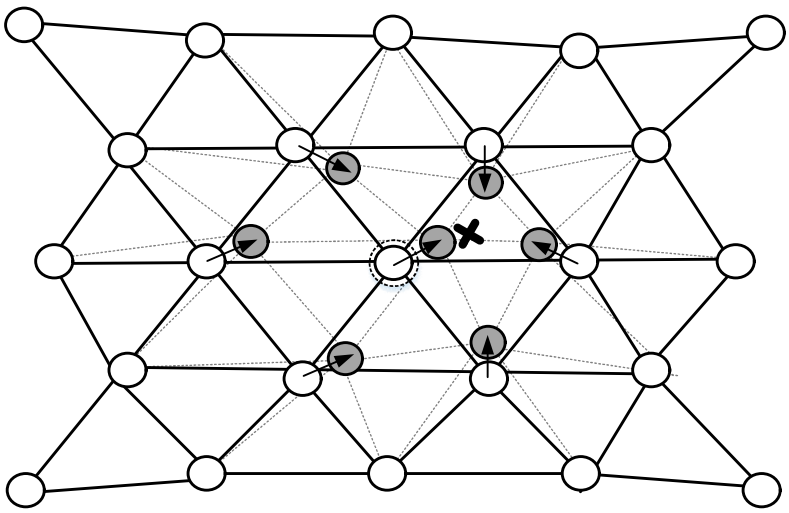


Рис. 7.4. Иллюстрация процесса подстройки сети Кохонена с учётом топологической окрестности

Их веса подстраиваются таким образом, чтобы в дальнейшем нейроны смогли увеличить свой шанс стать победителем для похожих наборов входных объектов. Иногда нейроны-соседи подстраиваются на меньшие величины по сравнению с нейроном-

победителем. В ряде случаев («совершенная» конкуренция) ситуация может быть прямо противоположной: нейрон-победитель получает минимальные или даже отрицательные значения для подстройки, а его соседи – существенно приближаются к входному объекту.

Наиболее употребимой функцией для определения радиуса топологической окрестности (или гиперсферы), в которую попадут нейроны-соседи, является функция Гаусса:

$$h(u,c,t) = \exp\left(-\frac{\rho(c,u)}{\sigma(t)}\right) \tag{7.1}$$

где u – номер нейрона в двумерной решетке слоя нейронов, для которого вычисляется значение h; c – номер нейрона-победителя; t – параметр времени.

Величина топологической окрестности σ нейрона-победителя может быть задана константой или функцией, которая линейно или экспоненциально стремится к нулю с увеличением параметра времени:

$$\sigma(t) = \frac{1}{\exp(t^{-2})} \tag{7.2}$$

Оптимальные значения параметров алгоритма обучения сети Кохонена – начальная скорость обучения η, радиус окрестности σ или закон его изменения также определяются исследователем для каждой предметной области на основе серии экспериментов.

Программная реализация

Ещё одна библиотека языка *python* – *tensorflow* – предназначена для реализации различных алгоритмов машинного обучения. Основной её целью является предоставление исследователю гибких, простых в использовании, но в то же время мощных инструментов для реализации задач из области построения и обучения нейронных сетей. Использование возможностей *tensorflow* позволяет решать задачи автоматического нахождения и классификации

образов, достигая качества человеческого восприятия. Функции библиотеки объединяют вычислительную алгебру и методы оптимизации для легкого вычисления многих математических выражений. Вычисления *tensorflow* выражаются в виде потоков данных через граф состояний. Название *tensorflow* происходит от операций с многомерными массивами данных, которые также называются «тензорами». Это объекты линейной алгебры, линейно преобразующие элементы одного линейного пространства в элементы другого. Частными случаями тензоров являются скаляры и векторы. Ранги таких тензоров d равны соответственно нулю или единице. В большинстве случаев тензор представляют собой многомерную таблицу размерности $d \times d \times ... \times d$, заполненную числами – компонентами тензора [10-12].

В приложении представлена программная система, состоящая из трёх модулей, для реализации карты Кохонена:

1. Модуль *model.py* содержит разработанный класс «*SOM*», в котором реализованы методы инициализации и обучения нейросети на основе технологии тензорных вычислений.
2. Модуль *grid.py* предназначен для создания и отображения раскрашенных карт.
3. Модуль *main.py* предназначен для обработки входных данных, определения оптимального значения параметра топологической окрестности нейронов-соседей и последующего старта процесса обучения карты.

Программная система обеспечивает выполнение следующих функций: подготовку данных к виду, пригодному для обработки нейронной сетью; инициализацию нейронной сети на основе технологии тензорных вычислений; вычисление значения параметра топологической окрестности для минимизации ошибки обобщения; последующее обучение сети с использованием оптимального σ. В результате проведённой серии экспериментов производится группировка входных объектов по совокупности признаков; наглядное представление результатов группировки; тестирование обученной нейронной сети для последующего соотнесения неизвестных ей объектов к сформированным кластерам.

В качестве данных для обучения используются реальные объекты – *SQL*-запросы, описываемые набором из 32-х параметров

производительности. При подготовке данных к виду, пригодному для обучения сети, и с целью сокращения размерности входного пространства используется метод главных компонент. Это позволяет перейти от 32-х мерного к 4-х мерному пространству.

Помимо библиотеки *tensorflow* для работы программы требуется установка библиотечных модулей *numpy* и *matplotlib*.

Вопросы для самоконтроля

1. Что понимается под картой Кохонена и чем она отличается от нейронной сети Кохонена?
2. Какие возможности реализуют нейронные карты?
3. Что понимается под раскраской карт? Как выполняется раскраска карт признаков кластеризуемых объектов?
4. Какие типы раскрасок отображают топологические особенности множества входных объектов? Как они создаются?
5. Какими факторами следует руководствоваться при определении числа узлов карты?
6. Что понимается под окрестностью нейрона? Какова её роль в обучении?
7. Формализуйте алгоритм обучения *SOM*.
8. Каковы параметры алгоритма обучения и каким образом они определяются?
9. Опишите достоинства и недостатки карты Кохонена.
10. Изучите программный код приложения для реализации самоорганизующейся карты и поясните каким образом тензорные вычисления оптимизируют процесс создания и обучения *SOM*?
11. Как в программной системе осуществляется подготовка данных для создания и обучения карты?
12. Опишите процесс определения оптимального значения топологической окрестности.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Варианты заданий по теме 1

1. Битовые образы – буквы русского алфавита размером 3×4 в количестве 5 первых букв фамилии студента.
2. Битовые образы – четные арабские цифры размером 4×4: 0,2,4,6,8.
3. Битовые образы – нечетные арабские цифры размером 4×4: 01,3,5,7,9.
4. Битовые образы – буквы латинского алфавита размером 3×4 в количестве 5 первых букв имени студента.
5. Битовые образы – графические изображения символов верхнего ряда клавиатуры размером 3×4 в количестве (!, #, *, и т.п.).
6. Битовые образы – буквы греческого алфавита размером 4×5 в количестве 5 шт.

Варианты заданий по теме 2

Рассчитать ошибку нейронной сети (рис. 2.1) на обучающей выборке, представляющей собой таблицу истинности логической функции двух аргументов. Определить минимальную и максимальную ошибку за эпоху. Весовые коэффициенты выбрать самостоятельно, основываясь на личных данных студента. Например, w_1=0,10 w_2=–0,30 w_3= 0,20 w_4=–0, 01 w_5=0,23 w_5=0,49 (дата рождения студента число, месяц, год – 10.03.2001, последние цифры зачетки - 2349).

1. Ошибка MSE, логическая функция – XOR.
2. Ошибка RMSE, логическая функция – XOR.
3. Ошибка Arctan, логическая функция – XOR.
4. Ошибка MSE, логическая функция – OR.
5. Ошибка RMSE, логическая функция – OR.
6. Ошибка Arctan, логическая функция – OR.
7. Ошибка MSE, логическая функция – штрих Шефера.
8. Ошибка RMSE, логическая функция - штрих Шефера.
9. Ошибка Arctan, логическая функция - штрих Шефера.
10. Ошибка MSE, логическая функция – стрелка Пирса.
11. Ошибка RMSE, логическая функция – стрелка Пирса.
12. Ошибка Arctan, логическая функция – стрелка Пирса.

Варианты заданий по теме 3

1. Рассчитать величину дельта на основе посчитанной ранее ошибки для одного (любого) элемента обучающей выборки.
2. Подстроить весовые коэффициенты сети.
3. Провести расчет ошибки для этого же элемента обучающей выборки на сети с новыми весовыми коэффициентами.
4. Выполнить пункты 1-3 для всей эпохи.
5. Обосновать полученный результат.

Сеть, её начальные значения весовых коэффициентов и обучающая выборка те же, что и в предыдущем задании. Значение скорости обучения задать самостоятельно.

Варианты заданий по теме 4

Провести ручной расчет первоначальной ошибки, подстройку коэффициентов сети и новое значение ошибки с применением матричных операций на собственном наборе входных данных. Архитектура сети: 3 входа, 3 выхода, 3 нейрона в скрытом слое.

Варианты заданий по теме 5

Провести программный расчет ошибки и коррекцию весовых коэффициентов нейронной сети для собственного набора входных данных. Вычислить ошибку новой сети на тех же входных данных.

Архитектура сети:

1. 2 входа, 1 выход, 1 скрытый слой с 2-мя нейронами.
2. 2 входа, 1 выход, 1 скрытый слой с 3-мя нейронами.
3. 2 входа, 1 выход, 1 скрытый слой с 4-мя нейронами.
4. 2 входа, 2 выхода, 1 скрытый слой с 2-мя нейронами.
5. 2 входа, 2 выхода, 1 скрытый слой с 3-мя нейронами.
6. 3 входа, 1 выход, 1 скрытый слой с 2-мя нейронами.
7. 3 входа, 1 выхода, 1 скрытый слой с 3-мя нейронами.
8. 3 входа, 2 выхода, 1 скрытый слой с 2-мя нейронами.
9. 3 входа, 2 выхода, 1 скрытый слой с 3-мя нейронами.
10. 3 входа, 3 выхода, 1 скрытый слой с 2-мя нейронами.
11. 3 входа, 3 выхода, 1 скрытый слой с 2-мя нейронами.
12. 3 входа, 3 выхода, 1 скрытый слой с 3-мя нейронами.

Начальные значения весовых коэффициентов сети задать случайным образом. Векторы входных/выходных значений, скорость

обучения и момент сформировать на основе личных данных студента.

Варианты заданий по теме 6

Произвести расчет изменения весовых коэффициентов нейронной сети Кохонена на самостоятельно созданном обучающем наборе. Значения признаков пронормировать. Для ускорения расчётов следует воспользоваться возможностями табличного процессора *Excel*. Параметры для построения сети Кохонена – размер сети; количество признаков объекта и предполагаемое количество кластеров для каждого варианта заданы в таблице.

Таблица Варианты заданий

№	Параметры задачи	№	Параметры задачи
1.	2×2, 2 признака; 4 кластера	7.	3×2; 2 признака; 6 кластеров
2.	2×2; 2 признака; 3 кластера	8.	3×2; 2 признака; 4 кластера
3.	2×2; 2 признака; 2 кластера	9.	3×2; 2 признака; 2 кластера
4.	2×2; 3 признака; 4 кластера	10.	3×2; 3 признака; 6 кластера
5.	2×2; 3 признака; 3 кластера	11.	3×2; 3 признака; 4 кластера
6.	2×2; 3 признака; 2 кластера	12.	3×2; 2 признака; 2 кластера

Примеры предметных областей для создания обучающих наборов: абоненты ресурсоснабжающей организации; объекты растительного и животного мира; учащиеся учебных заведений; объекты купли-продажи; клиенты финансовых организаций; пользователи социальных сетей; объекты социально-политической деятельности и др.

Варианты заданий по теме 7

1. Подготовьте данные для обучения карты Кохонена на основе примера, приведённого в программном коде. В качестве входных данных используйте объекты из предыдущего задания.

2. Запустите программу для получения оптимального значения ширины топологической окрестности.

3. Обучите новую карту с использованием оптимального параметра. Проанализируйте сформированные программой раскрашенные карты. Сделайте выводы о распределении объектов.

ЛИТЕРАТУРА

1. Мак-Каллок У.С., Питтс У. Логическое исчисление идей, относящихся к нервной активности // «Автоматы» под ред. К.Э.Шеннона и Дж. М.Маккарти: 1956. - С. 362 - 384. [Электронный ресурс]: http://www.raai.org/library/books/mcculloch/mcculloch.pdf

2. Кузнецова А.В., Мохов В.А. Нечеткая логика и нейронные сети в наукоемких производствах: учеб. Пособие: Новочеркасск: ЮРГТУ (НПИ), 2011. -142 с.

3. Нейронные сети. Statistica Neural Networks: Методология и технологии современного анализа данных / под редакцией В.П. Боровикова. - 2-е изд., перераб. и доп. – М.: Горячая линия-Телеком, 2008. - 392 с.

4. Яхъяева Г.Э. Нечеткие множества и нейронные сети : учеб. пособие для вузов. – М.: БИНОМ. Лаборатория знаний : Интернет-Ун-т Информ. технологий, 2006. - 316 с.

5. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы: пер. с польск. / Д. Рутковская, М. Пилиньский, Л. Рутковский. – М.: Горячая линия-Телеком, 2006. - 452 с.

6. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика / 2-е изд. – М.: Горячая линия-Телеком, 2002. - 382 с.

7. Нейронные сети: STATISTICA Neural Networks : пер. с английского. – М.: Горячая линия-Телеком, 2000. - 182 с.

8. Сузи Р. А. Язык программирования PYTHON: учебное пособие – М.: БИНОМ. Лаборатория знаний: Интернет-Ун-т Информ. технологий, 2006. - 326 с

9. Уроки по программированию, DevOps и другим IT-технологиям: Учебники/Машинное обучение. [Электронный ресурс]: https://coderlessons.com/tutorials

10. Официальный сайт TensorFlow. [Электронный ресурс]: https://www.tensorflow.org/

11. Справочник Википедии по TensorFlow. [Электронный ресурс]: https://en.m.wikipedia.org/wiki/TensorFlow

12. Ramsundar B., Zadeh R.B. TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning // «O'Reilly Media», 1 edition, March 23, 2018. – 256 p.

ПРИЛОЖЕНИЕ

МОДУЛЬ model.py

```
#!/usr/bin/python
import tensorflow as tf
import numpy as np
import grid

class SOM:
  def __init__(self, Rows, Cols, Inputs, seed=None):
    # значение для инициалиации генератора псевдослучайных чисел
    self.Seed = seed

    # вычисление топологии SOM заданного размера
    self.grid = grid.Grid(Rows, Cols)

    # вычисление топологических расстояний между каждыми двумя элементами карты
    topodists = np.linalg.norm(np.expand_dims(self.grid.Centers, axis=2) -
                               np.expand_dims(self.grid.Centers, axis=1),axis = 0)

    # задание константы топологических расстояний в вычислительном графе
    self._TopoDists = tf.constant(topodists, name='Distances')

    # матрица ближайших соседей для каждого элемента
    # (0 - сам элемент или любой не соседний, 1 - ближайший соседний)
    self._NNN  = tf.constant((np.absolute(topodists - 1) < 0.001).astype(np.float32))
```

```
# задание константы координат элементов карты в вычислительном графе
self._Coords = tf.constant(self.grid.Centers.T,  dtype=tf.float32, name='Grid')

# входные узлы вычислительного графа:
#   Alpha  - скорость обучения,
#   Sigma  - параметр ширины функции соседства,
#   Input  - поэлементный вход самоорганизующейся карты
#   BatchedInput - пакетный вход самоорганизующейся карты
self._Alpha= tf.placeholder(shape=(None), dtype=tf.float32, name='Alpha')
self._Sigma= tf.placeholder(shape=(None), dtype=tf.float32, name='Sigma')
self._Input= tf.placeholder(shape=(Inputs), dtype=tf.float32, name='Input')
self._BatchedInput = tf.placeholder(shape=(None, Inputs), dtype=tf.float32,
                                    name='BatchedInput')

# переменная для хранения весовых коэффициентов(координат)
# элементов карты в пространстве входного множества
self._Weights= tf.Variable(tf.random_normal([Cols*Rows, Inputs], seed=self.Seed),
                           dtype=tf.float32, name='Weights')

# вычисление расстояние между текущим входным элементом
# и всеми элементами карты пространстве входных данных
self._DeltaW = self._Input - self._Weights
self._Dist = tf.norm(self._DeltaW, axis=1)
```

```
# получение индекса ближайшего элемента карты
self._BMU  = tf.argmin(self._Dist)

# то же, но для пакетного входа
self._BatchedDeltas = (tf.expand_dims(self._BatchedInput, axis = 1) -
                       tf.expand_dims(self._Weights,      axis = 0))

# индекс кластера для каждого входного элемента, предоставленного на пакетный вход
self._BatchedBMU = tf.argmin(tf.norm(self._BatchedDeltas, axis = 2), axis = 1)

# значение индекса кластера к которому принадлежит
# элемент входного мнодества в унитарном коде
self._OneHotBBMU = tf.one_hot(self._BatchedBMU,    axis=1, depth=Cols*Rows)

# количество элементов входного множества в каждом кластере
self._ClustHits = tf.reduce_sum(self._OneHotBBMU, axis=0, keepdims=True)

# соотнесения элементов входного множества с сответствующими им
# элементами самоорганизующейся карты
self._ClustBatchInp = tf.multiply(
                        tf.expand_dims(self._BatchedInput, axis = 1),
                        tf.expand_dims(self._OneHotBBMU,   axis = 2))

# вычислене ошибки квантования для каждого кластера
self._Qmap = tf.reshape(
```

```
                            tf.divide(
                              tf.reduce_sum(
                                tf.norm(
                                  tf.multiply(
                                    self._ClustBatchInp -
                                    tf.expand_dims(self._Weights,    axis = 0),
                                    tf.expand_dims(self._OneHotBBMU, axis = 2)),
                                  axis = 2),
                                axis = 0),
                              self._ClustHits),
                            shape=[-1])

    # вычисление U-map для карты
    self._Umap = tf.reduce_sum(
                            tf.multiply(
                              tf.norm(
                                tf.expand_dims(self._Weights, axis=1) -
                                tf.expand_dims(self._Weights, axis=0),
                                axis=2),
                              self._NNN),
                            axis=1) / tf.reduce_sum(self._NNN, axis=1)

    # вычисление маски интерполяции
    self._InterpolationMask = tf.multiply(
                                tf.cast(
```

```
                                  tf.abs(
                                    self._TopoDists -
                                    tf.reduce_min(
                                      tf.boolean_mask(
                                        self._TopoDists,
                                        tf.reshape(self._ClustHits, shape=[-1]) > 0),
                                      axis = 0,
                                      keepdims = True )) < 0.001,
                                  dtype=tf.float32),
                                tf.cast(
                                  tf.transpose(self._ClustHits) > 0,
                                  dtype=tf.float32))

# вычисление среднего для каждого кластера
self._ClusterMean   = tf.divide(
                                tf.reduce_sum(self._ClustBatchInp, axis = 0),
                                tf.transpose(self._ClustHits))

# вычисление дисперсии для каждого кластера
self._ClusterVariance   = tf.divide(
                                tf.reduce_sum(
                                  tf.square(
                                    tf.multiply(
                                      self._ClustBatchInp - self._ClusterMean,
                                      tf.expand_dims(self._OneHotBBMU, axis=2))),
```

```
                                axis = 0), tf.transpose(self._ClustHits))

# интерполяция средних кластеров с ненулевы количеством элементов
# входного множества на кластеры с нулевым количеством элементов
self._ClustMeanInerpol  = tf.divide(
                              tf.reduce_sum(
                                tf.multiply(
                                  tf.expand_dims(self._InterpolationMask, axis=2),
                                  tf.expand_dims(
                                    tf.where(
                                      tf.is_nan(self._ClusterMean),
                                      tf.zeros_like(self._ClusterMean),
                                      self._ClusterMean),
                                    axis=1)), axis=0),
                              tf.transpose(
                                tf.reduce_sum(
                                  self._InterpolationMask, axis=0, keepdims=True)))

# интерполяция дисперсий кластеров с ненулевы количеством элементов
# входного множества на кластеры с нулевым количеством элементов
self._ClustVariInerpol  = tf.divide(
                              tf.reduce_sum(
                                tf.multiply(
                                  tf.expand_dims(self._InterpolationMask, axis=2),
```

```
                                    tf.expand_dims(
                                      tf.where(
                                        tf.is_nan(self._ClusterVariance),
                                        tf.zeros_like(self._ClusterVariance),
                                        self._ClusterVariance),
                                      axis=1 )), axis=0),
                                  tf.transpose(
                                    tf.reduce_sum(
                                      self._InterpolationMask, axis=0, keepdims=True)))

# вычиление значений функции соседства для всех элементов карты
self._Neighbourhood = tf.multiply(
                          self._Alpha,
                          tf.exp(
                            tf.negative(
                              tf.square(
                                tf.norm( tf.subtract(self._Coords,
                                                     self._Coords[self._BMU, :]),
                                  axis=1, ord='euclidean', keepdims=True) ) /
                                (2 * (self._Sigma ** 2)) )),
                            name='Neighbourhood')

# вычисление значений для обновления весов карты
self._WUpdDelta = tf.multiply(self._DeltaW, self._Neighbourhood)
```

```
  # обновление весов карты
  self._Train = tf.assign_add(self._Weights,self._WUpdDelta, name='Train')

  # создание сесси для исполнения вычислительного графа и инициализация
  # переменных (весов карты)
  self._Session = tf.Session()
  tf.global_variables_initializer().run(session=self._Session)

# метод для обучения карты
def train( self, EpochsNum, Alpha, train_ds, validation_ds=None,
           sigma_func  = lambda x, x_max, a, b: a-(a-b)*(x/x_max),
           sigma_start = 50, sigma_end   = 0 ):

# обнуление векторов ошибок
  GenErr = []
  Qerr = []

  # цикл обучения карты
  for epoch in range(0, EpochsNum):
    for i in train_ds:
      Ws = self._Session.run( self._Train,
             feed_dict={self._Input: i, self._Alpha: Alpha,
                                self._Sigma: sigma_func(epoch, EpochsNum,
                                                        sigma_start, sigma_end) })
```

```
    # вычисление ошибки обобщения для карты на текущей эпохе обучения
    if validation_ds is not None:
      TrMeans, TrVars, Qmap = self._Session.run(
                          [self._ClustMeanInerpol, self._ClustVariInerpol, self._Qmap],
                            feed_dict={self._BatchedInput: train_ds} )

      VlMeans, VlVars = self._Session.run([self._ClustMeanInerpol,
                self._ClustVariInerpol], feed_dict={self._BatchedInput: validation_ds} )

      GenErrByWeight     np.square(TrMeans - VlMeans) + (TrVars + VlVars)/2
      GenErr.append(GenErrByWeight.mean())
      Qerr.append(np.nanmean(Qmap))
  return Ws, np.array(GenErr), np.array(Qerr)

# метод для получения значений индексов ближайших элементов карты
# для каждого элемента входного множества
def classify(self, DataSet):
  return self._Session.run(
           self._BatchedBMU, feed_dict = {self._BatchedInput: np.array(DataSet)})

# метод для получения значений индексов ближайших элементов карты
# для каждого элемента входного множества в унитарном коде
def classify_onehot(self, DataSet):
  return self._Session.run(
           self._OneHotBBMU, feed_dict = {self._BatchedInput: np.array(DataSet)})
```

```
  # метод вычисления ошибки квантования для каждого элемента карты с учётом входных данных
  def calculate_qmap(self, DataSet):
    return self._Session.run(
             self._Qmap,       feed_dict = {self._BatchedInput: np.array(DataSet)})
```

МОДУЛЬ GRID.PY

```
import numpy as np
class Grid():
  def __init__(self, Rows, Cols, Stride=1, Type='hex'):
    seq   np.array(range(0, Rows*Cols))     # порядковый номер каждого элемента решётки

    # вычисление координат для каждого элемента решётки
    self.Centers = np.stack((Stride / 2 * (seq // Cols % 2) + seq % Cols,
                             Stride * (seq // Cols) * np.sin(np.pi/3)), axis=0 )

    # вычисление координат вершин шестиугольника для визуализации карты
    radius    = Stride / (2 * np.cos(np.pi/6))
    self.Vertices = np.stack((np.cos(np.pi*2*np.array(range(0,6))/6+np.pi/6),
                    np.sin(np.pi*2*np.array(range(0,6))/6+np.pi/6)),axis=0 )*radius

    # координаты вершин шестиугольника для каждого элемента решётки
    self.Cells = np.expand_dims(self.Vertices.T, axis=0) + np.expand_dims(self.Centers.T,
axis=1)
```

МОДУЛЬ main.py

```
from matplotlib.collections import PatchCollection, PolyCollection
from matplotlib.patches import Polygon
import grid
import matplotlib.pyplot as plt
import model
import numpy as np
import tensorflow as tf

# имена компонент входных данных – признаки кластеризуемых объектов
var_names = ['executions', 'sharable_mem', 'loaded_versions', 'version_count',
  'io_interconnect_bytes', 'io_offload_elig_bytes', 'io_offload_return_bytes',
  'physical_read_bytes', 'cell_uncompressed_bytes', 'physical_write_bytes',
  'parse_calls', 'px_servers_execs', 'end_of_fetch_count', 'fetches', 'buffer_gets',
  'invalidations', 'loads', disk_reads', 'optimized_physical_reads', '
   physical_read_requests', 'rows_processed', 'sorts', 'physical_write_requests',
  'direct_writes', 'cpu_time', 'elapsed_time', 'javexec_time', 'plsexec_time',
  'iowait', 'apwait', 'ccwait', 'clwait']

# загрузка данных
data   = np.loadtxt('data4.txt').tolist()
labels = np.loadtxt('labels4.txt', dtype=np.str).tolist()
       = np.matrix(data, dtype=np.float)
```

```
# вычисление основных моментов для входных данных
mx_d = np.nanmean(d, axis=0)
std_d = np.nanstd(d, axis=0)

# центрирование и стандартизация входных данных
d0     = (d - mx_d) / std_d

# вычисление ковариационной\корреляционной матрицы входных данных
cov_d0 = np.cov(d0.T)

# вычисление собственных векторов и чисел корреляционной матрицы
d0_eigval, d0_eigvec = np.linalg.eig(cov_d0)   # первичные компоненты

# сортировка компонент по убыванию объясняемой каждым компонентом дисперсии
index     = np.argsort(d0_eigval)[::-1]
d0_eigvec = d0_eigvec[:, index]
d0_eigval = d0_eigval[index]

# проекция данных на первичные компоненты
InputData      = np.matmul(d0, d0_eigvec[:, 0:11])

# разделение входных данных на обучающую и проверочную выборки для SOM
TrainData      = InputData[:InputData.shape[0]//2, :]
ValidationData = InputData[InputData.shape[0]//2:, :]
```

```
# размеры карты
Rows = 70
Cols = 70

# создание объекта карты
MySOM1 = model.SOM(Rows, Cols, InputData.shape[1], seed=12345)

# обучениие карты
weights, GenErr, Qerr = MySOM1.train(25, 0.05, train_ds      = np.array(TrainData),
                                     validation_ds = np.array(ValidationData) )

# вычисление границ для визуализации
borders = np.stack((MySOM1.grid.Cells.min((0,1)),MySOM1.grid.Cells.max((0,1))), axis=0).T

# получение результатов кластеризации входных данных
BMUs = MySOM1.classify(InputData)

# вычисление количества элементов входных данных в каждом кластере
HIST = MySOM1.classify_onehot(InputData).sum( axis=0 )

# вычисление U-map для самоорганизующейся карты
NNN = (np.abs(np.linalg.norm(
           np.expand_dims(MySOM1.grid.Centers, axis=2) -
           np.expand_dims(MySOM1.grid.Centers, axis=1),
           axis=0)) - 1 < 0.01).astype(int)
```

```
Umap = np.multiply(
         np.linalg.norm(np.expand_dims(weights.T, axis=2) -
           np.expand_dims(weights.T, axis=1),
           axis=0), NNN ).sum(axis=1) / NNN.sum(axis=1)

# получение карты ошибки квантования для каждого кластера
Qmap = MySOM1.calculate_qmap(InputData)

# перевод значений весов карты к компонентам входного сигнала
# из пространства первичных компонент для визуализации
restored_weights = np.matmul(weights, d0_eigvec.T[0:11])

# вычисление карты удалённостей центроидов кластеров от
# точки отсчёта координат (мат. ожидания входного множества)
magnitude = np.linalg.norm(weights, axis=1)

# отрисовка визуализации карты и сохранение визуализации в файлы
for i, j in zip( np.concatenate(
                   ( np.stack(
                     ( np.array(HIST).reshape(-1,),
                       Qmap,
                       np.nan_to_num(Qmap),
                       Umap,
                       magnitude ), axis=1),
                     weights,
```

```
                    restored_weights ),
                  axis=1).T,
                  ['HIST', 'QMAP_W_NAN', 'QMAP_WO_NAN', 'UMAP', 'MAGNITUDE'] +
                  ['COMP_%02d' % num for num in range(0, InputData.shape[1])] +
                  ['%02d_%s' % (index, value) for index, value in enumerate(var_names) ]
):

  fig, ax = plt.subplots()
  fig.set_size_inches( fig.get_size_inches() * 2 )
  plt.tight_layout()
  plt.axis('scaled')
  ax.set_title(j)
  ax.set_facecolor('grey')

  p = PolyCollection(MySOM1.grid.Cells, array=i, cmap='jet', antialiased=False,
snap=False, linewidths = 0)
  p.autoscale()
  ax.add_collection(p)
  plt.xlim(borders[0])
  plt.ylim(borders[1])
  plt.colorbar( mappable = ax.collections[0] )
  plt.savefig(j + '.png')
  plt.close()
  del p
```

Учебное издание

Кузнецова Алла Витальевна
Мохов Василий Александрович
Алгазали Салах Махди Мадлол

АЛГОРИТМЫ ОБУЧЕНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Учебно-методическое пособие
по изучению дисциплины
«Искусственный интеллект и мягкие вычисления»

Редактор Я.В. Максименко.

Подписано в печать 30.06.2020
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 4,18. Уч.-изд.л. 4,25. Тираж 50 экз. Заказ 46-0242.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

В.Н. Кубил

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ДОРОЖНОГО ДВИЖЕНИЯ В СРЕДЕ AnyLogic

Учебное пособие по дисциплине «Цифровые технологии в инженерии»

Новочеркасск
ЮРГПУ (НПИ)
2020

УДК 004.94:656.11

Рецензент – заведующий кафедрой «Программное обеспечение вычислительной техники», доцент, кандидат технических наук
Гринченков Дмитрий Валерьевич

Кубил В.Н.

Имитационное моделирование дорожного движения в среде AnyLogic: учеб. пособие по дисциплине «Цифровые технологии в инженерии» / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова.– Новочеркасск: ЮРГПУ (НПИ), 2020. – 56 с.

Приведены программы практических занятий, посвященных имитационному моделированию дорожного движения в среде AnyLogic. Даны темы для самостоятельной работы.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата по направлениям подготовки 23.03.01 «Технология транспортных процессов» и 23.03.03 «Эксплуатация транспортно-технологических машин и комплексов».

УДК 004.94:656.11

ВВЕДЕНИЕ

AnyLogic — программное обеспечение для имитационного моделирования, разработанное российской компанией *The AnyLogic Company*. Инструмент обладает современным графическим интерфейсом и позволяет использовать язык *Java* для разработки моделей в самых разных областях: цепи поставок, склады, железные дороги, горное дело, нефть и газ, склады, пассажиропотоки и т.д. В рамках данного практического курса вы освоите моделирование в одной из таких областей — дорожное движение.

Проектирование дорожной инфраструктуры — всегда непростая задача. Необходимо предотвращать заторы, предусматривать рост траффика, и при этом учитывать возможности бюджета и особенности конкретных населенных пунктов.

Прежде чем вносить изменения в дорожную сеть, необходимо принять в расчёт все факторы, которые могут оказать влияние на дорожную ситуацию. Особенно это важно при строительстве крупных общественных зданий, таких как аэропорты, ж/д вокзалы, станции метро и стадионы, при подведении к ним дорог и организации парковок. Размещение и синхронизация светофоров, расположение объектов дорожной сети (парковок, остановок общественного транспорта, выделенных полос) — всё это оказывает непосредственное влияние на траффик и пропускную способность дорог, а значит тоже должно быть принято во внимание. Внесение изменений в дорожную сеть всегда имеет экономические эффекты, и они могут быть крайне тяжелыми, если проводить эксперименты над реальной системой. Поэтому так важно находить оптимальные решения до того, как вносить изменения.

AnyLogic позволяет моделировать дорожные сети, используя Библиотеку дорожного движения — гибкий и мощный инструмент для создания реалистичных имитационных моделей и принятия наиболее эффективных решений при проектировании и оснащении дорог. Визуализация помогает быстро построить модель и оценить её работу: карты плотности показывают загруженность дорог, а анимация демонстрирует поток машин и узкие места. *AnyLogic* даёт полную свободу в экспериментах и позволяет оптимизировать модель в виртуальной среде для последующей успешной реализации проекта в реальном мире.

Практическое занятие №1

ЗНАКОМСТВО СО СРЕДОЙ ДЛЯ МОДЕЛИРОВАНИЯ *AnyLogic*

Цель занятия: ознакомиться с интерфейсом среды для моделирования *AnyLogic* и научиться создавать с ее помощью новые модели.

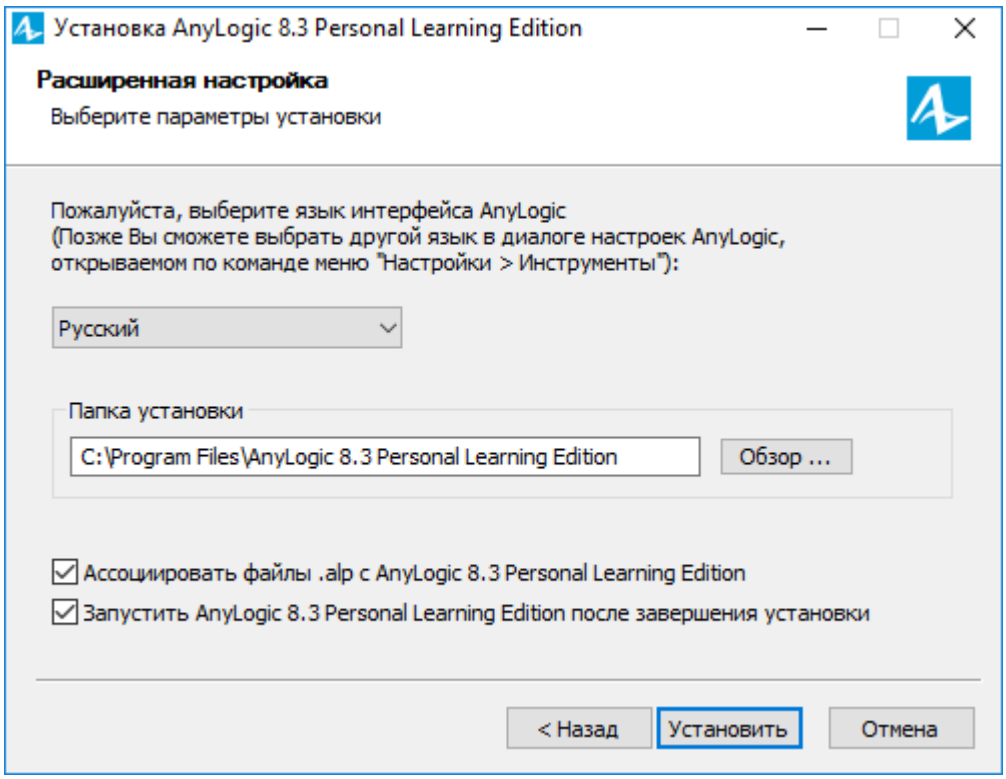
Программа занятия

1. Установить *AnyLogic* на персональную ЭВМ.
2. Создать новую модель.
3. Ознакомиться с пользовательским интерфейсом.
4. Добавить элементы палитр на диаграмму.
5. Построить и запустить модель.
6. Подготовить и оформить отчёт о проделанной работе.

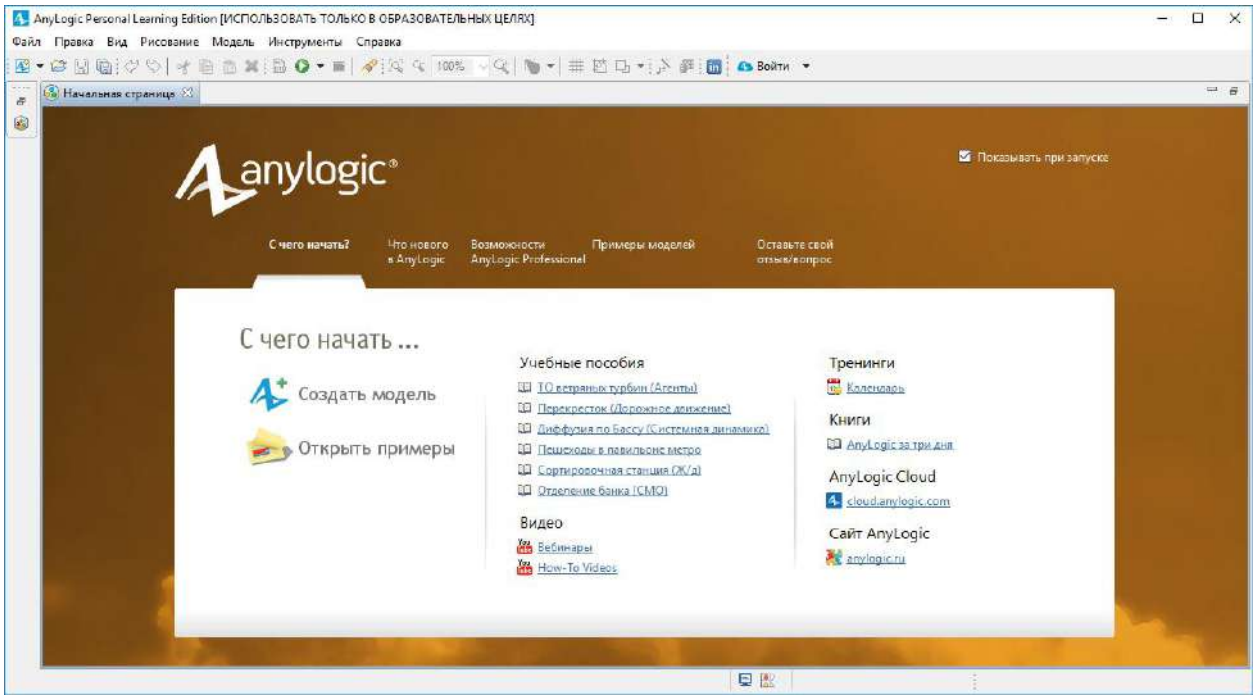
Пояснения к занятию

Получить бесплатную версию *AnyLogic Personal Learning Edition (PLE)* для начинающих и студентов можно на официальном сайте компании-разработчика. Для этого перейдите по адресу www.anylogic.ru и сверху в навигационной панели выберите раздел «скачать». На открывшейся странице приведена сравнительная таблица различных вариантов издания, а также системные требования среды разработки. В колонке «*Personal Learning Edition*» дана ссылка на бесплатную версию программы, для загрузки которой потребуется только заполнить краткую форму-анкету.

1. Установить *AnyLogic* на персональную ЭВМ. Запустите полученную программу установки *AnyLogic* и следуйте инструкциям инсталлятора. Версия для самостоятельного обучения *AnyLogic PLE* устанавливается без активации. В настройках установки при необходимости измените язык интерфейса и путь для устанавливаемых файлов программы. После установки на Рабочем столе появится ярлык для запуска *AnyLogic*.



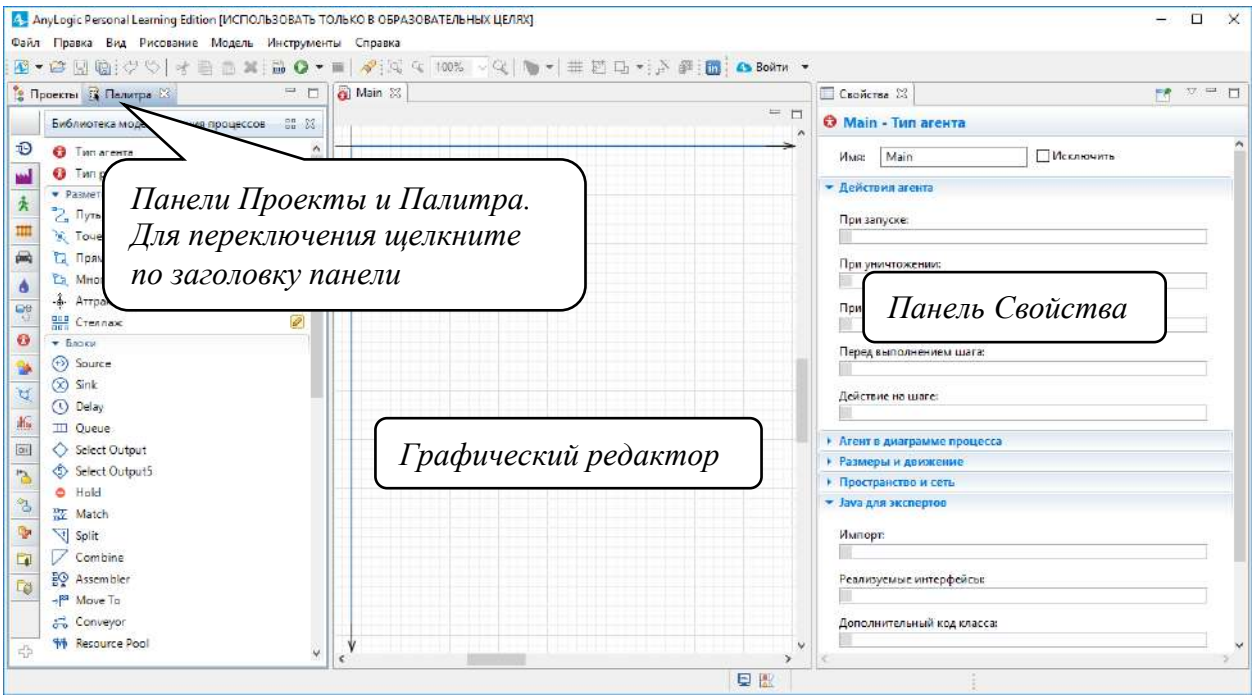
2. Создать новую модель. Запустите *AnyLogic*. Откроется *Начальная страница*. *Начальная страница* предлагает обзор программы *AnyLogic* и ее функционала, а также позволяет открывать различные примеры моделей. Попробуем создать новую модель.



1.1. Закройте Начальную страницу и создайте новую модель. Для этого выберите **Файл** > **Создать** > **Модель** из главного меню *AnyLogic*. Откроется диалоговое окно **Новая модель.**

1.2. Введите любое **Имя модели**, которое потом будет отображаться в списке проектов.
1.3. В поле **Местоположение** выберите каталог, в котором вы хотите сохранить файлы модели.
1.4. Щелкните по кнопке **Готово**. Будет создана новая модель.

3. Ознакомиться с пользовательским интерфейсом. Теперь рассмотрим основные элементы пользовательского интерфейса *AnyLogic*.



- *Графический редактор* позволяет редактировать диаграмму агента. Агенты – это главные строительные блоки модели *AnyLogic*. Вы можете добавлять элементы на диаграмму, перетаскивая их из **Палитры** на холст редактора. Синяя прямоугольная рамка ограничивает ту область холста, которая будет отображаться в окне модели при ее запуске.
- Панель **Проекты** отображает содержимое моделей *AnyLogic*, открытых в рабочем пространстве в текущий момент. Элементы каждой модели отображаются в виде иерархического дерева, для облегчения навигации.
- Панель **Палитра** содержит все графические элементы *AnyLogic*, которые сгруппированы в отдельные палитры, отображаемые в виде значков слева. Чтобы добавить тот

или иной элемент в модель, перетащите соответствующий элемент из палитры в графический редактор.

- Панель **Свойства** позволяет вам просматривать и изменять свойства выделенных в текущий момент элементов модели.
- Чтобы открыть или закрыть панель, выберите в меню **Вид** соответствующий пункт с именем панели.
- Чтобы изменить размер панели, перетащите мышью ее границу.
- Вы всегда можете воспользоваться опцией **Восстановить расположение панелей** в меню **Инструменты**, чтобы вернуть расположение панелей по умолчанию.

Вы можете переключиться между панелями **Палитра** и **Проекты**, щелкая по заголовкам в левой части рабочего пространства. Откройте панель **Проекты**, чтобы изучить структуру новой модели.

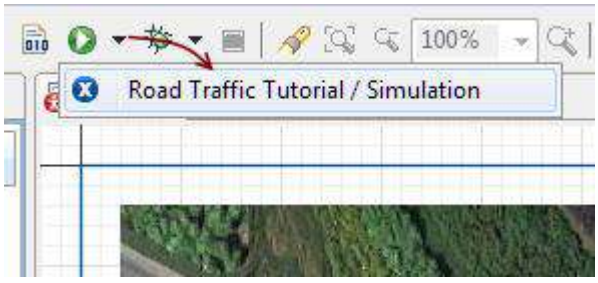
- Панель **Проекты** предоставляет простой и быстрый доступ к содержимому моделей, открытых в рабочем пространстве *AnyLogic*. *AnyLogic* отображает структуру каждой модели в виде иерархического дерева.
- По умолчанию в каждой модели создается один тип агента – `Main` – и один эксперимент `Simulation`, хранящий настройки запуска этой модели.
- Двойной щелчок по типу агента или эксперименту открывает его диаграмму в графическом редакторе.
- Также у каждой модели есть своя встроенная **База данных**. База данных изначально пуста, но при необходимости вы можете импортировать в нее данные из внешней базы данных (например, *MS Excel*), а также собрать информацию о выполнении модели в специальные логи (для этого нужно выбрать в свойствах базы данных опцию **Записывать в лог информацию о выполнении модели**).
- Щелчок по элементу модели в дереве выделяет этот элемент и показывает его в центре графического редактора. Если вы не можете найти какой-либо элемент на графиче-

ской диаграмме, воспользуйтесь таким способом нахождения элемента.

- Над областью графического редактора отображаются вкладки диаграмм. Если вкладка с нужной диаграммой закрыта (например, *Main*), кликните дважды по нужному пункту панели **Проекты** для ее открытия.

4. Добавить элементы палитр на диаграмму. Переключитесь на панель **Палитры**, выберите один или несколько произвольных графических элементов из любой палитры (например, иллюстрацию из палитры **Картинки**) и перетяните на пустую диаграмму агента *Main* в область графического редактора, ограниченную голубой рамкой. Затем откройте палитру **Презентация** и добавьте таким же образом элемент ***Текст***. В появившемся поле введите ваши ФИО, факультет и группу. Выделите добавленный элемент, кликнув по нему в графическом редакторе, и на панели **Свойства** измените его внешний вид по своему усмотрению, выбрав другие значения для полей *Цвет* и *Шрифт*.

5. Построить и запустить модель. Сначала постройте вашу модель с помощью кнопки панели инструментов **Построить модель.** (клавиша F7 на клавиатуре), а затем запустите эксперимент с помощью копки **Запустить** (клавиша F5 на клавиатуре). На момент запуска этого конкретного эксперимента наша модель – единственная открытая модель в рабочем пространстве. В дальнейшем будет запускаться тот эксперимент, который запускался вами в последний раз. Чтобы выбрать какой-то другой эксперимент, вам будет нужно щелкнуть правой кнопкой мыши по этому эксперименту в панели **Проекты** и выбрать **Запустить** из контекстного меню.



В появившемся окне кликните по кнопке запуска ►, после чего должна появиться ваша статичная модель с добавленными объектами. Сохраните модель, выбрав **Файл** > **Сохранить.**

6. Подготовить и оформить отчёт о проделанной работе. Составьте, распечатайте и защитите у преподавателя отчет о проделанной работе. В отчет обязательно включаются: заголовок (номер и название практического занятия), цель и описание проделанной работы со скриншотом конечного результата, ответы на контрольные вопросы, выводы, место для подписей студента и преподавателя.

Контрольные вопросы

1. Что такое имитационное моделирование?
2. В каких случаях целесообразно прибегать к имитационному моделированию? Приведите пример.
3. Какие подходы (виды) имитационного моделирования вам известны? Чем они отличаются? Какие из них поддерживаются в *AnyLogic*?
4. На разработку моделей в каких областях ориентированы инструменты *AnyLogic*?
5. Какие элементы по умолчанию содержит любая новая модель *AnyLogic*?

Практическое занятие №2

СОЗДАНИЕ ДОРОГ И ПЕРЕКРЕСТКОВ

Цель занятия: научиться создавать дороги и перекрестки в среде для моделирования *AnyLogic*, изучить их свойства.

Программа занятия

1. Создать в *AnyLogic* новую модель.
2. Добавить спутниковый снимок на графическую диаграмму.
3. Отключить привязку к сетке и установить масштаб.
4. Создать дороги и перекрестки.
5. Настроить атрибуты дорог.
6. Скорректировать форму дорог и перекрестков.
7. Подготовить и оформить отчёт о проделанной работе.

Примерные варианты заданий

№	Город	Пересекающиеся улицы
1	Москва	Харьковская ул., Медынская ул.
2	Ростов-на-Дону	Красноармейская ул., Будённовский просп.
3	Новочеркасск	Баклановский просп., Магнитный пер.
4	Москва	Садовое кольцо, Садовая-Спасская ул., просп. Академика Сахорова
5	Ростов-на-Дону	просп. Космонавтов, бульв. Комарова
6	Новочеркасск	Баклановский просп., Троицкая пл.
7	Москва	ул. Хачатуряна, Каргопольская ул.
8	Ростов-на-Дону	ул. Текучёва, Будённовский просп.
9	Новочеркасск	Платовский просп., пл. Ермака
10	Москва	ул. Новинки, Коломенская ул.
11	Ростов-на-Дону	Пушкинская ул., Будённовский просп.
12	Новочеркасск	Московская ул., ул. Просвещения
13	Москва	ул. Зацепский Вал, Кожевническая ул., Татарская ул.
14	Ростов-на-Дону	ул. Вавилова, Таганрогская ул.
15	Новочеркасск	Баклановский просп., ул. Крылова., пер. Галины Петровой
16	Москва	Садовая-Черногрязская ул., ул. Земляной Вал, Старая Басманная ул.
17	Ростов-на-Дону	Коммунистический просп., ул. Зорге
18	Новочеркасск	Баклановский просп., Пушкинская ул., ул. Ленгника
19	Москва	ул. Пресненский Вал, Ходынская ул., Большой Тишинский пер.
20	Ростов-на-Дону	Театральный просп., Красноармейская ул.

№	Город	Пересекающиеся улицы
21	Новочеркасск	Харьковское шоссе, ул. Мацоты
22	Москва	ул. Островитянова, ул. Академика Опарина
23	Ростов-на-Дону	Красноармейская ул., Кировский просп.
24	Новочеркасск	Баклановский просп., Сарматская ул.
25	Москва	ул. Вавилова, ул. Панфёрова
26	Ростов-на-Дону	Пушкинская ул., Будённовский просп.
27	Новочеркасск	Платовский просп., Московская ул.
28	Москва	Тверская ул., ул. Охотный Ряд., Моховая ул.
29	Ростов-на-Дону	просп. Михаила Нагибина, просп. Ленина
30	Новочеркасск	Будённовская ул., ул. 26 Бакинских Комиссаров

Пояснения к занятию

Для данного практического занятия Вам понадобится спутниковый снимок местности. Получите его у преподавателя либо воспользуйтесь любой известной вам картографической системой (например, *Google Maps* или Яндекс.Карты). Снимок должен отчетливо отображать форму и разметку участка дорожной сети с указанием масштаба (обычно в нижнем правом углу). Ваша задача – воссоздать дороги и перекрестки в модели *AnyLogic*, разместив их строго поверх тех, что на снимке. Варианты заданий из таблицы выдаются на усмотрение преподавателя. Рассмотрим порядок создания дорог и перекрестков в *AnyLogic* на примере развязки в микрорайоне Хотунок г. Новочеркасска (рис. 2.1).

Рис. 2.1. Пересечение улиц Трамвайная, Харьковская, Макаренко, Гагарина и Петрова на двух перекрестках в г. Новочеркасске.

С помощью картографического сервиса Яндекс.Карты был получен спутниковый снимок данного участка (рис. 2.2).

Рис 2.2. Спутниковый снимок перекрестков

1. Создать в *AnyLogic* новую модель. В диалоговом окне создания модели введите ее имя `Road Traffic 00`, указав вместо `00` номер вашего варианта. Единицы модельного времени – секунды. Создаваемый вместе с моделью агент `Main` послужит местом, в котором мы выстроим логику модели: здесь мы создадим дорожную сеть и в дальнейшем зададим движение транспорта на диаграмме процесса. В центре рабочей области должен отображаться графический редактор диаграммы типа агента `Main`.

2. Добавить спутниковый снимок на графическую диаграмму. Откройте палитру **Презентация** и перетащите элемент ***Изображение*** на графическую диаграмму `Main`.

Теперь нам необходимо выбрать изображение, которое мы будем использовать. Диалог для выбора файла появится автоматиче-

ски. Откройте папку, в которой находится ваш файл изображения, выберите его и нажмите **Открыть.** Разместите изображение таким образом, чтобы перекрестки оказались внутри рамки, ограничивающей область для отображения в окне модели при ее запуске. Если рамку не видно за изображением временно нажмите **Спрятать фигуры** на нижней панели инструментов.

Чтобы ваш участок дорожной сети наилучшим образом уместился в пределах рамки можете увеличить или уменьшить ее размер, например, указав в свойствах высоту и ширину, соответствующие размеру изображения в пикселях. Если снимок небольшой относительно размера экрана желательно, чтобы его границы совпали с рамкой. В этом случае при запуске модели спутниковый снимок полностью займет окно. Для элемента изображения на панели **Свойства** поставьте галочку *Блокировать*, чтобы при дальнейших действиях случайно не сместить его. При необходимости разблокировки любой элемент всегда можно найти в одной из веток панели **Проекты.**

3. Отключить привязку к сетке и установить масштаб. Прежде чем начать добавлять такие элементы разметки, как дороги или перекрестки, отожмите кнопку **Включить/Отключить сетку** на панели инструментов. Это позволит размещать элементы модели более точно относительно снимка. Чтобы вам было легче редактировать отдельные графические элементы, увеличьте масштаб графической диаграммы с помощью кнопки **Приблизить** на панели инструментов (либо прокручивайте колесо мыши при зажатой клавише *Ctrl*). Перемещаться по графическому окну удобно двигая мышь с зажатой правой кнопкой.

Теперь нужно установить правильный масштаб. Для этого перетяните элемент **Масштаб** на снимок и измените его длину так, чтобы она совпала с длиной отрезка масштаба на снимке (рис 2.3). Затем укажите в его свойствах нужную длину линейки. Таким образом размер добавляемых в модель элементов будет соответствовать размерам объектов на снимке.

Рис 2.3. Наложение масштаба модели на снимок

4. Создать дороги и перекрестки. Откройте палитру **Библиотека дорожного движения** и сделайте двойной щелчок по элементу ***Дорога***. Иконка элемента изменится и активируется режим рисования. Теперь можно рисовать дорогу в графическом редакторе. Определитесь с тем, какую улицу будете считать основной. В рассматриваемом примере – это улица Трамвайная, идущая с северо-запада на юго-восток. Кликните мышью по разделительной полосе дороги на снимке там, где начинается основная улица, чтобы поставить первую точку. Чтобы нарисовать прямой сегмент, щелкните мышью в том месте, где хотите разместить конечную точку сегмента. Если сегмент дороги имеет форму дуги, вместо клика зажмите левую кнопку мыши в точке окончания дугового сегмента и перемещайте курсор с нажатой левой кнопкой мыши до тех пор, пока сегмент не приобретет необходимую форму. Продолжайте рисовать дорогу сегмент за сегментом. В случае необходимости отмены последнего действия нажмите на кнопку панели инструментов **Отменить** (либо используйте сочетание клавиш *Ctrl+Z*). При внезапной необходимости выйти из какого-либо режима, в том числе режима рисования, можно нажать клавишу *Esc*. Чтобы завершить рисование дороги, добавьте ее последнюю точку двойным щелчком мыши. Если это первая дорога в модели, вы, скорее всего, увидите сообщение с предложением сменить масштаб. Поскольку правильный масштаб уже был установлен заранее, следует отказаться.

Перекрестки в *AnyLogic* создаются автоматически при соединении дорог, поэтому нужно нарисовать новую дорогу, образующую перекресток с основной. Как и прежде начните рисование дороги с двойного щелчка по элементу ***Дорога*** в палитре. Добавьте начальную точку, щелкнув по разделительной полосе там, где дорога должна начаться. Продолжайте рисовать, а затем добавьте конечную точку в месте пересечения с основной дорогой для создания перекрестка. Чтобы правильно соединить дороги в Т-образный перекресток, сделайте двойной щелчок мыши, когда увидите зеленую точку на разделительной полосе основной дороги. Появившийся перекресток разобьет основную дорогу на две, каждую из которых теперь можно настраивать по отдельности. Повторите описанные действия для создания других дорог и перекрестков вашего участка дорожной сети, если таковые присутствуют на снимке. В рассматриваемом примере имеется два перекрестка: на одном

пересекаются три дороги (Т-образный), на другом – четыре (Х-образный).

5. Настроить атрибуты дорог. Прежде чем начать настраивать моделируемые дороги вам нужно увидеть их на снимке. Для этого кликните один раз по дороге, чтобы выделить ее, а затем еще раз, чтобы выделилась вся дорожная сеть. На панели **Свойства** в секции **Внешний вид** измените *Цвет дороги* на «**Нет заливки**». В результате видимой останется только белая разметка нарисованных дорог.

По умолчанию дороги создаются с двусторонним движением, при этом каждое направление дороги (основное и встречное) содержит по две полосы движения. Кликните по дороге и в ее свойствах задайте *Кол-во полос основного движения* и *Кол-во полос встречного движения* для каждой нарисованной дороги в соответствии со снимком. Заметьте, крайняя пунктирная линия разметки отделяет дорогу не от обочины, а от еще одной полосы движения. Если движение по дороге возможно только в одном направлении, отметьте галочкой *Одностороннее движение*. Если дорога имеет нестандартную разделительную полосу, подберите значение *Ширины разделительной полосы*. При необходимости можете задать в свойствах нужный цвет разделительной полосы. Таким же образом настройте все дороги в модели.

6. Скорректировать форму дорог и перекрестков. Расположение сегментов выделенной дороги можно корректировать, перемещая точки, расположенные на разделительной полосе. Вы можете добавлять либо удалить точки двойным щелчком по разделительной полосе в нужном месте. Нажатием правой кнопки мыши вызывается контекстное меню, через которое можно выбрать следующие опции:

- **Изменить направление.** При этом основное направление изменится на противоположное. Основное направление указывает фиолетовая стрелка по центру выделенной дороги.
- **Сделать сегмент прямым/дуговым.** При наведении мыши на сегмент вы увидите, какую форму этот сегмент примет после преобразования. Щелкните мышью по сегменту, форму которого вы хотите изменить.
- **Редактировать направляющие (изменение изгибов).** Вы увидите множество маркеров. Если квадратные маркеры

отображают конечные точки сегментов дороги, то прямые линии с круглыми маркерами на концах - это и есть направляющие линии. У каждой конечной точки сегмента есть своя направляющая линия. Направляющие линии появятся для каждой точки изгиба дороги. Перетащите мышью конечную точку направляющей линии (рис 2.4). Форма соответствующего дугового сегмента при этом будет меняться. Отпустите кнопку мыши, когда вы получите сегмент нужной вам формы. Возможно, для этого потребуется также изменить длину и/или положение другой направляющей линии этого сегмента дороги. По завершению редактирования снова вызовите контекстное меню щелчком правой кнопки и отключите данную опцию.

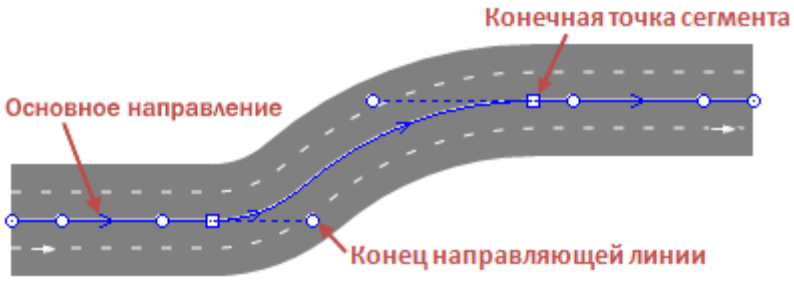
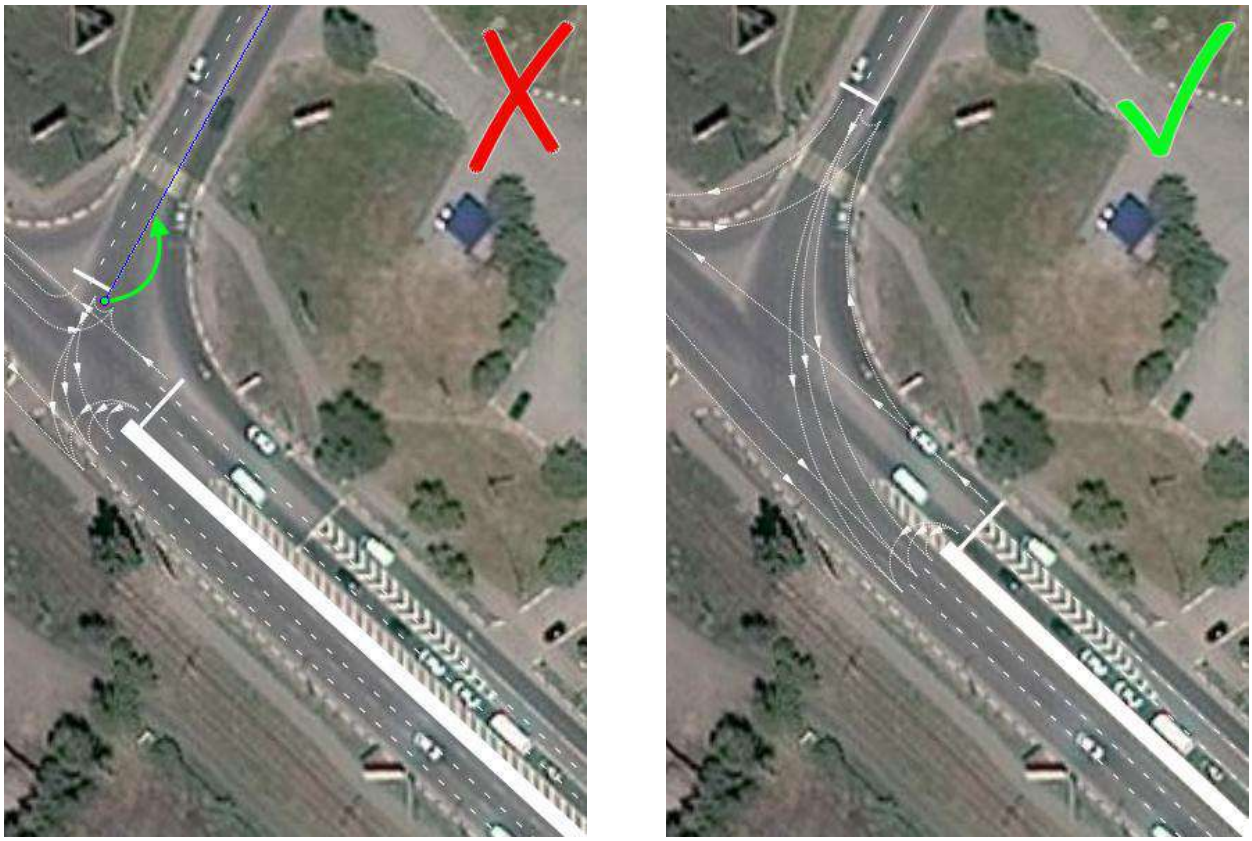


Рис. 2.4. Редактирование сегмента дороги

Чтобы увеличить форму и длину въездов на перекрестке, нужно увеличить площадь самого перекрестка. Это можно сделать, отдалив конечные точки примыкающих к перекрестку дорог. Для этого сначала выделите в графическом редакторе дорогу. Вы увидите, что ее конечная точка у перекрестка выделена зеленым цветом. Аккуратно перетащите эту точку в нужную сторону от перекрестка. При этом отпустить кнопку мыши следует только если точка все еще подсвечивается зеленым цветом – это означает сохранение соединения с перекрестком. Если же точка станет белой, то соединение дороги с перекрестком будет потеряно – в таком случае отмените последнее действие, и затем выполните его корректно. Если необходимо переместить точку далеко, делайте это в несколько подходов, по чуть-чуть. Продолжайте до тех пор, пока стоп-линия модели, находящаяся при въезде на перекресток, не совпадет со стоп-линией снимка, как на рисунке справа. Если стоп-линии на снимке не видно, добейтесь наилучшего совпадения скруглений перекрестка, предварительно вернув дорогам серый цвет.

В итоге вся дорожная разметка (разделительные полосы, пунктирные границы полос, стоп-линии) должна правильно накладываться, как показано на рис. 2.5, б.



а) б)

Рис 2.5. Наложение дорог на снимок:
а – плохое соответствие; б – приемлемое соответствие

Крайние точки дорог у границ снимка целесообразно вынести за его пределы таким образом, чтобы до ближайшего перекрестка оставалось не менее двухсот метров, как показано на рис. 2.6. Тем самым вы обеспечите место для скопления поступающих потоков транспортных средств при дальнейшем моделировании.

Придумайте сокращенные обозначения для каждой дороги и добавьте их в модель с помощью элементов ***Текст***. В рассматриваемом примере подпишем дороги в соответствии с названиями: *Tram*, *Khar*, *Gaga*, *Petr*, *Maka*.

7. Подготовить и оформить отчёт о проделанной работе. Укажите вариант вашего задания. Опишите ход выполнения своими словами, избегая любых заимствований. Обязательно дайте пояснения относительно конкретно вашего индивидуального варианта (перекрестка), а не абстрактного. Попытайтесь обосновать принимаемые вами решения (например, задаваемые свойства элементов

модели). Добавьте снимок с конечным результатом, не забыв вернуть дорожной сети изначальный цвет «Grey». В остальном структура отчета аналогично предыдущему.

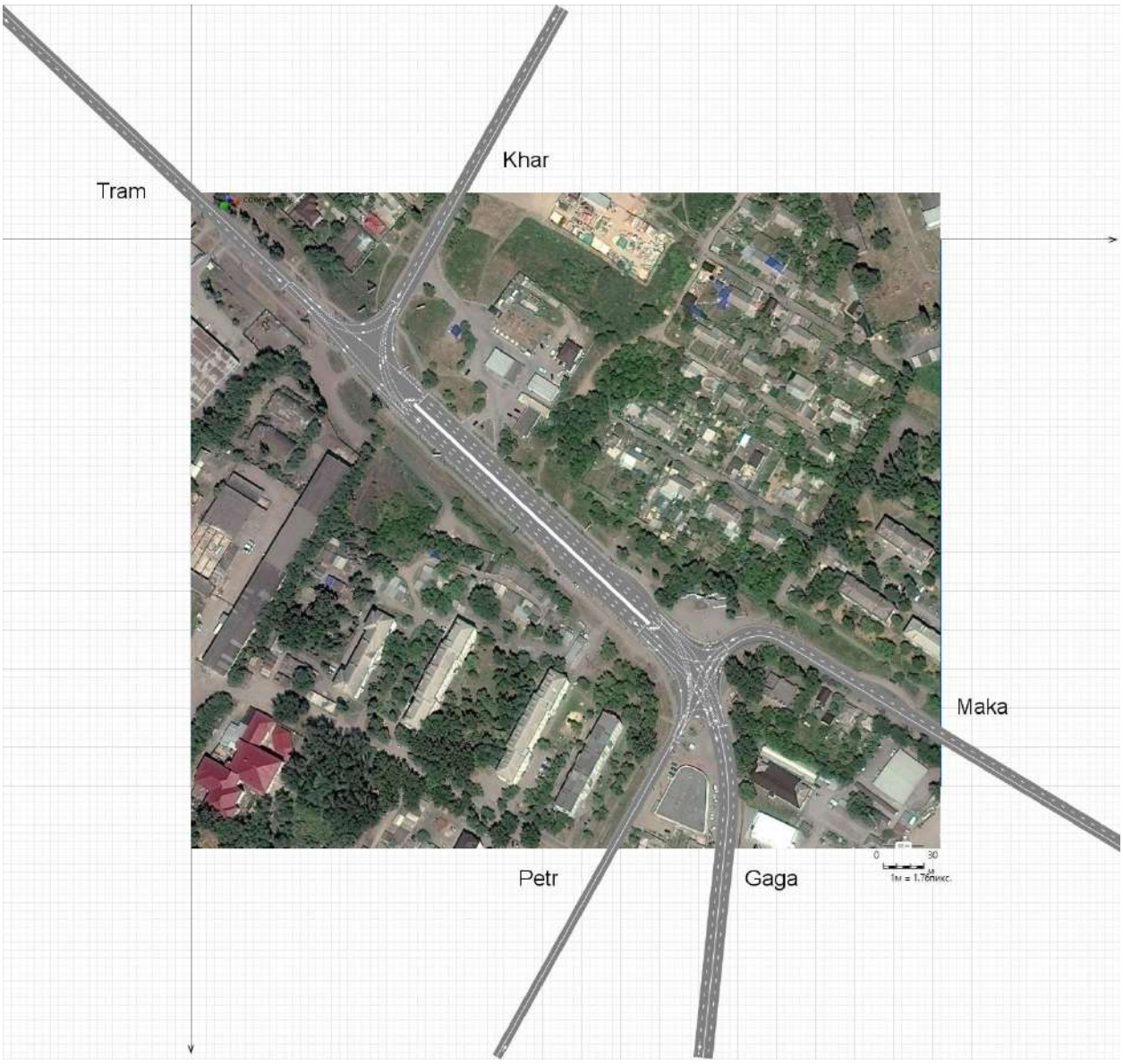


Рис. 2.6. Результат мостроения модели дорог и перекрестков

Контрольные вопросы

1. Какие свойства дорог в *AnyLogic* вам известны? Какие из них вы редактировали при создании вашего участка дорожной сети?
2. Каким образом можно изменить форму и кривизну дугового сегмента дороги?
3. Что такое масштаб? Как его изменение влияет на модель?
4. Изучите раскрывающийся список на панели **Проекты**: какие элементы модели добавились в группе **Main** > **Презентация** в результате создания вашего участка дорожной сети?

Практическое занятие №3

УСТАНОВКА ПРАВИЛ ДОРОЖНОГО ДВИЖЕНИЯ

Цель занятия: научиться устанавливать правила дорожного движения с учетом знаков и разметки в моделях *AnyLogic*.

Программа занятия

1. Задать разрешенные направления движения автомобилей.
2. Скорректировать форму соединителей полос.
3. Установить приоритеты и ограничения движения.
4. Создать диаграмму процесса движения автомобилей.
5. Построить и запустить модель.
6. Подготовить и оформить отчет о проделанной работе.

Пояснения к занятию

Для этого практического занятия вам нужно уточнить некоторые детали дорожной сети. Запустите любой доступный вам сервис для просмотра улиц (например, *Google Street View* или Яндекс Панорамы) и с его помощью изучите ваш участок дорожной сети, перемещаясь по улицам и осматриваясь. Обратите внимание на особенности разметки и наличие дорожных знаков. Разрешенные повороты и развороты можно также проверить с помощью сервисов построения маршрутов, ставя начальную и конечную точки пути рядом с нужным направлением движения: если полученный маршрут не проходит через рассматриваемый поворот или разворот, значит он запрещен.

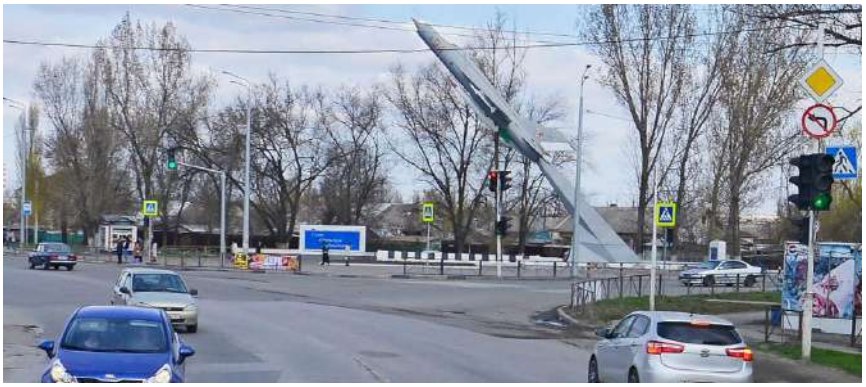
1. Задать разрешенные направления движения автомобилей. Отображаемые на перекрестке белые точечные линии – соединители полос – задают разрешенные направления движения транспорта на перекрестке. Вы можете удалить существующие соединители полос, добавить отсутствующие, а также изменить форму соответствующего соединителя полос. Щелкните по перекрестку, чтобы выделить его. Выбранный перекресток отобразит прямоугольные маркеры на конце каждой полосы присоединенной к перекрестку дороги. Щелкните по прямоугольному маркеру той полосы, для которой необходимо разрешить или запретить направление движения. Откроется режим редактирования соединителей полос, который содержит все добавленные (изображены сплошной белой линией) и

удаленные (изображены серой линией) соединители, ведущие от данной полосы. Щелкая по соединительным линиям, вы можете менять их цвет и тем самым разрешать/запрещать направления движения транспорта для выбранной полосы (рис 3.1). Щелкните в пустом месте графической диаграммы, чтобы выйти из режима редактирования соединителей полос. Теперь вы сможете увидеть внесенные изменения.

Рис 3.1. Режим редактирования соединителей полос

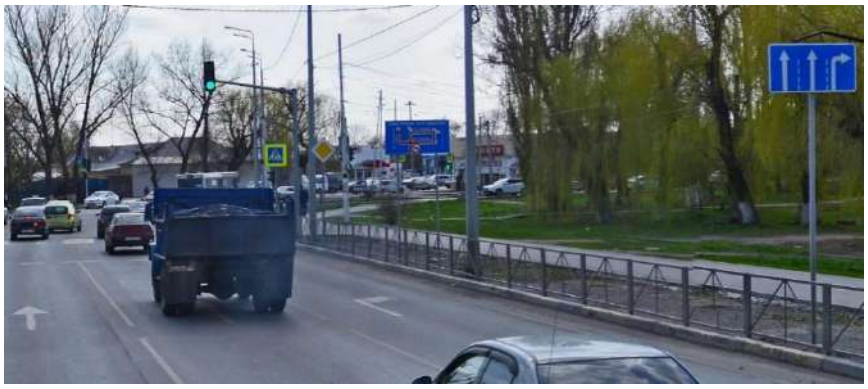
Покажем на примере из прошлого практического занятия.

- во-первых, согласно знаку, который можно увидеть на панорамах, двигаясь с улицы Гагарина, запрещен левый поворот на улицу Петрова;

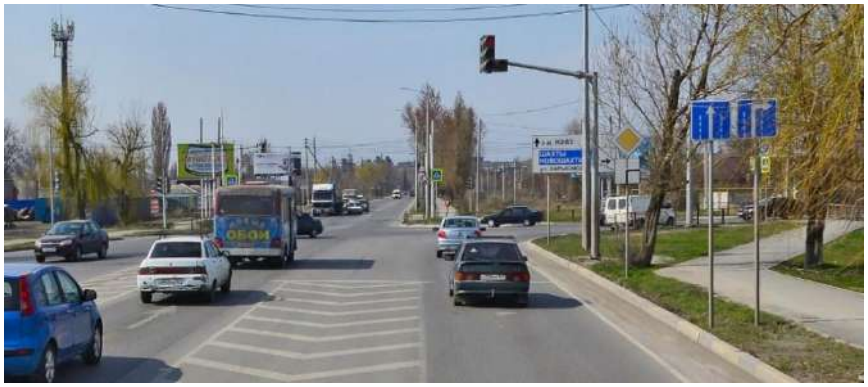


- во-вторых, с улицы Макаренко знак разрешает движение прямо с левой полосы, либо направо с правой полосы;

- в-третьих, при движении по улице Трамвайной с северо-запада на юго-восток согласно знаку и разметке с правой полосы возможен только правый поворот на улицу Петрова, а с остальных – только движение прямо на улицу Гагарина;



- в-четвертых, двигаясь по улице Трамвайной в обратном направлении (с юго-востока на северо-запад) с левой полосы возможно только продолжить движение прямо, с правой – повернуть на улицу Харьковскую, а с центральной полосы выезд на перекресток запрещен разметкой;



- в-пятых, любые развороты на рассматриваемых перекрестках не предусмотрены в силу высокого трафика, проходящего через развязку.

Остальные решения при отсутствии достоверных данных принимаются с позиции здравого смысла и с учетом косвенных источников информации.

2. Скорректировать форму соединителей полос. Форма соединителей полос изменяется теми же способами, что и форма дорог. Чтобы было удобно ориентироваться по снимку, снова временно уберите цвет заливки дорог в свойствах дорожной сети, как это делалось на прошлом практическом занятии. Кликните по перекрестку, а затем по соединителю полос, чтобы выделить его. Если фигу-

ра соединителя достаточно длинная, на ней может появиться квадратная метка. Фигуры соединителей меньшего размера по умолчанию не содержат метки. Если нужен соединитель полос сложной формы, добавьте еще одну квадратную метку двойным щелчком на соединителе. Перемещайте квадратные метки путем перетягивания, чтобы изменить форму соединителя полос. Вы также можете щелкнуть правой кнопкой по соединителю полос и выбрать опцию **Редактировать направляющие**, чтобы изменить изгиб соединителя полос, аналогично тому, как это делалось с дорогами (см. пояснения к предыдущему практическому занятию).

Описанным образом нужно скорректировать форму всех соединителей полос так, чтобы она соответствовала ожидаемой вами траектории движения автомобилей через перекресток (рис. 3.2).

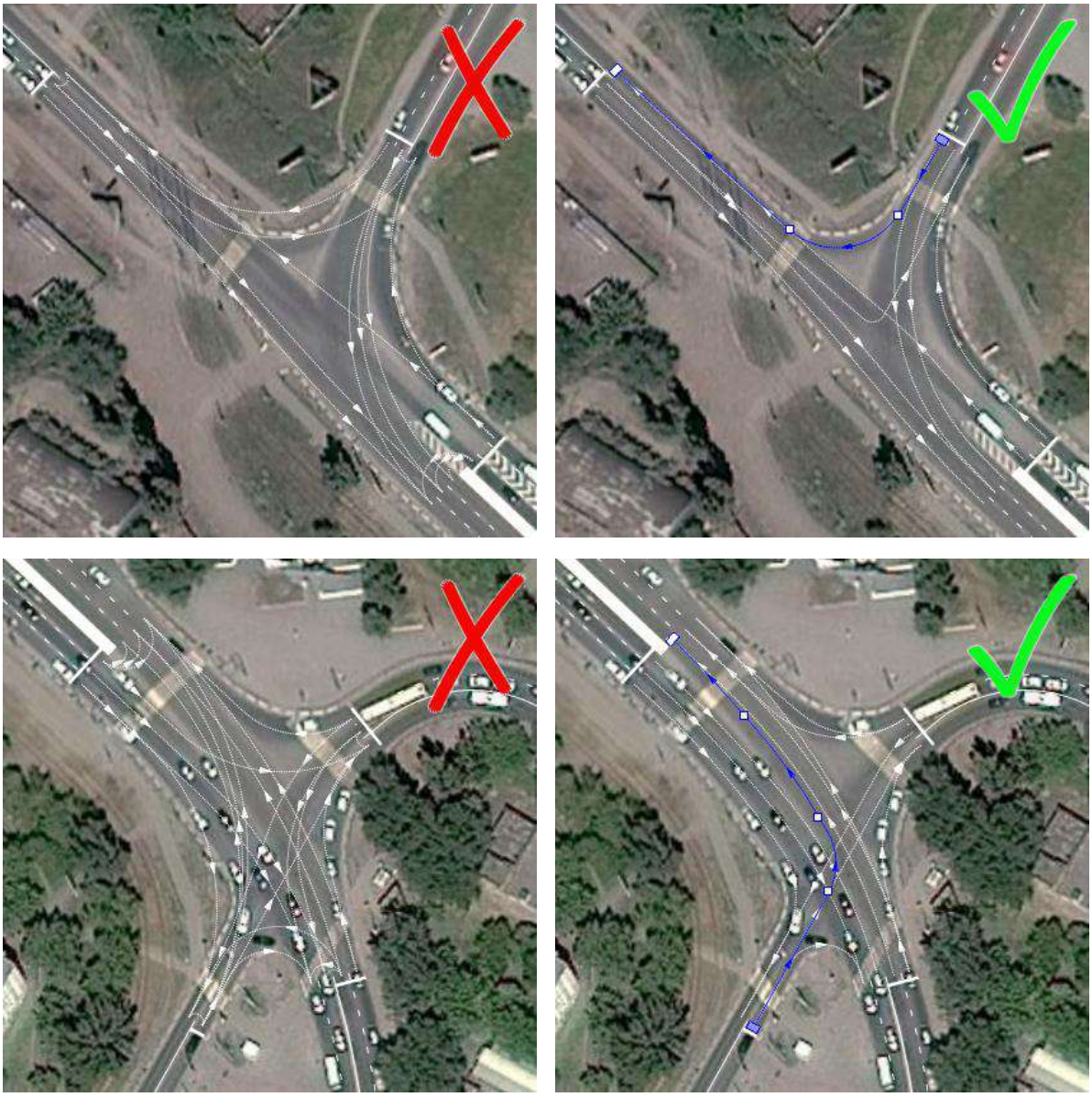


Рис 3.2. Редактирование направляющих

Все соединители полос должны оказаться в пределах перекрестка на снимке, как показано на рис. 3.2. Расстояние между параллельными соединителями полос должно оставаться не меньше ширины дорожной полосы, чтобы автомобили могли свободно двигаться, не создавая друг другу помеху.

В целом имеет смысл руководствоваться тем, как в действительности осуществляется движение. Например, левые повороты должны быть более резкими, чтобы автомобили не ехали против встречного потока, а пропускали его на середине перекрестка и затем совершали маневр. Именно так в рассматриваемом примере сделаны соединители полос для левых поворотов с ул. Трамвайной на ул. Харьковскую, и с ул. Петрова на ул. Трамвайную.

3. Установить приоритеты и ограничения движения. Приоритет дорожного движения и скоростной режим регулируются с помощью свойств элемента ***Стоп-линия***, который позволяет добавлять следующие дорожные знаки в местах собственного расположения на дорогах: *Ограничение скорости*; *Конец ограничения скорости*; *Уступить дорогу*. Дорожные знаки повлияют на дорожное движение у стоп-линии, у которой они установлены. Для установки приоритета движения через перекресток достаточно поставить флажок *Уступить дорогу* в свойствах тех стоп-линий, рядом с которыми стоит соответствующий знак.

Для ограничения максимальной скорости понадобится добавить дополнительную стоп-линию, перетащив ее в нужное место дороги из палитры **Библиотека дорожного движения**, затем в свойствах поставить флажок *Ограничение скорости* и указать значение. Чтобы дополнительная стоп-линия не отображалась при симуляции, отключите в свойствах ее *Видимость*.

В рассматриваемом примере главными являются дороги, соединяющие улицы Харьковскую и Гагарина, поэтому на остальных дорогах в свойствах стоп-линий перед перекрестками устанавливаем свойство *Уступить дорогу*. Также присутствуют знаки ограничения максимальной скорости до 40 км/ч с обеих сторон на улице Макаренко (рис. 3.3) и на улице Петрова, для которых добавляю дополнительные стоп-линии со свойством *Ограничение скорости*.

Рис 3.3. Знаки ограничения максимальной скорости на панорамах улиц

4. Создать диаграмму процесса движения автомобилей. Первым делом включите привязку к сетке, чтобы добавляемые блоки диаграммы процесса было легче выравнивать относительно друг друга. Откройте палитру **Библиотека дорожного движения** и добавьте блоки ***CarSource***, ***CarMoveTo*** и ***carDispose*** из этой палитры на диаграмму, перетащив их в свободное место графического редактора. Добавляемые блоки имеют порты (изначально желто-зеленые), к которым могут подходить соединительные линии. Когда блоки добавляются рядом друг с другом, соединительные линии между ними появляются автоматически.

Начать рисование соединительной линии можно с двойного клика по порту. Дополнительные щелчки в режиме рисования линии позволяют добавлять точки для поворотов и соединения с другими портами. Будьте внимательны, линии должны соединять только порты, находящиеся с правой или левой стороны блоков. Соедините добавленные блоки, как показано на рис. 3.4.

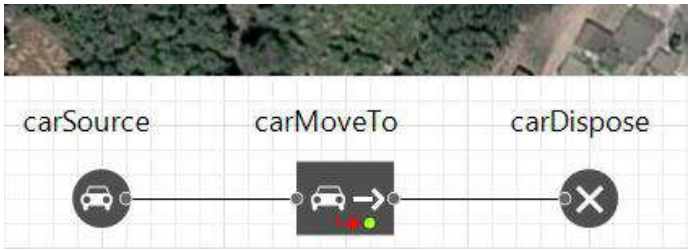


Рис 3.4. Диаграмма процесса движения автомобилей

Скажем пару слов о добавленных блоках:

- ***CarSource*** генерирует автомобили. Обычно используется как начальная точка автомобильной диаграммы процесса.
- ***CarMoveTo*** моделирует движение автомобиля к цели движения.

⊗ ***CarDispose*** удаляет поступающие на вход этого блока автомобили из модели. Обычно используется как конечная точка автомобильной диаграммы процесса.

Выделите добавленный блок *carSource*. На панели **Свойства** в раскрывающемся списке **Дорога** выберите одну из дорог либо кликните по иконке рядом со списком, чтобы выбрать дорогу непосредственно в графическом редакторе. Таким образом мы указываем блоку *carSource* дорогу, на которую надо помещать создаваемые автомобили. Укажите в свойствах блока, на полосу основного или встречного движения должны помещаться автомобили. Понять, какая полоса соответствует основному движению можно по белым стрелкам на ее концах.

Точно так же внесите изменения в свойства блока *carMoveTo*, указав другую дорогу, на которую возможен проезд с первой. В свойствах задайте полосу основного или встречного движения, до конца которой должно моделироваться движение автомобилей. В блоке *carDispose* оставьте настройки по умолчанию.

5. Построить и запустить модель. Убедитесь, что не допустили ошибок в структуре дорожной сети, построив модель. Построение не будет завершено, если в модели присутствуют ошибки, информация о которых будет выведена в панель **Ошибки**. Чтобы перейти к месту ошибки и исправить ее, совершите двойной щелчок мышью по ошибке в этом списке. После того, как все ошибки устранены и модель успешно построена, вы можете осуществить тестовый запуск и пронаблюдать, как поток автомобилей будет двигаться между выбранными дорогами. Таким образом мы задали движение на участке дорожной сети по одному из возможных путей.

6. Подготовить и оформить отчет о проделанной работе. В отчет включите панорамы и снимок модели полностью настроенного перекрёстка вашего индивидуального задания, подобно снимкам, представленным в методических указаниях.

Контрольные вопросы

1. Какие дорожные знаки вы учли в модели участка дорожной сети вашего индивидуального задания? Каким образом?

2. Какие особенности разметки вы учли в модели участка дорожной сети вашего индивидуального задания? Каким образом?

3. Как понять, какая полоса дороги в модели предназначена для основного движения, а какая – для встречного?

4. Что такое диаграмма процесса?

5. Какие блоки диаграммы процесса задают движение автомобилей? Чем они отличаются?

Практическое занятие №4

МОДЕЛИРОВАНИЕ ПОТОКОВ АВТОМОБИЛЬНОГО ТРАНСПОРТА

Цель занятия: научиться моделировать движение автомобилей по созданным дорогам и перекресткам в *AnyLogic*.

Программа занятия

1. Определиться с системой имен элементов.
2. Составить таблицу интенсивности прибытия автомобилей.
3. Доработать диаграмму процесса.
4. Задать интенсивности потоков автомобилей.
5. Исправить возможные ошибки и запустить модель.
6. Подготовить и оформить отчёт о проделанной работе.

Пояснения к занятию

На этом практическом занятии вам предстоит построить полную диаграмму процесса, чтобы запустить потоки автомобилей во всех возможных направлениях по ранее созданным дорогам и перекресткам вашего индивидуального участка дорожной сети.

1. Определиться с системой имен элементов. Все графические элементы модели имеют имена, заданные в свойствах по умолчанию. Для дорог используется имя *road*. При наличии нескольких однотипных элементов к имени автоматически приписывается индекс. Поскольку в дальнейшем при построении диаграммы процесса вам понадобится ссылаться на дороги, имеет смысл сделать «говорящими» имена по крайней мере тех из них, которые подходят к границам. Например, вместо индекса можно добавить часть названия улицы, получив имена *roadTram* (ул. Трамвайная), *roadPetr* (ул. Петрова) и т.п. Если две дороги относятся к одной улице, можно добавить к имени сторону света или другие понятные метки. К именам перекрестков для удобства различения можно приписать, например, их тип, получив *intersectionT* или *intersectionX*.

2. Составить таблицу интенсивности прибытия автомобилей. Определите все возможные пути движения автомобилей и выпишите из начальные и конечные пункты в первые два столбца таблицы. При отсутствии достоверной статистики можно оценить интенсивность исходя из количества полос в направлении прибытия автомо-

билей. Рекомендуемое суммарное значение 300 автомобилей в час на полосу обеспечит достаточно плотный поток для дальнейшего анализа. Какая часть от этого значения должна двигаться в каждом из возможных целевых направлений определите приблизительно исходя из количества полос целевой дороги (в направлении выезда), а также любой другой вспомогательной информации.

Описанным образом была подготовлена таблица 4.1 с интенсивностями прибытия для путей движения автомобилей в рассматриваемом примере. В четвертом столбце указывается суммарная интенсивность прибытия, пропорциональная количеству полос, а в третьем – приблизительная для каждого пути и соответствующий процент от суммарной. Составьте аналогичную таблицу для своего индивидуального варианта задания. В вашем индивидуальном задании не забудьте также учесть разрешенные развороты, для которых путь движения автомобилей начинается и заканчивается на одной и той же дороге, но в разных направлениях.

Таблица 4.1

Откуда	**Куда**	**Интенсивность прибытия в час**	
roadTram	*roadGaga*	450 (75%)	600
roadTram	*roadPetr*	90 (15%)	
roadTram	*roadKhar*	60 (10%)	
roadKhar	*roadGaga*	300 (50%)	600
roadKhar	*roadTram*	240 (40%)	
roadKhar	*roadPetr*	60 (10%)	
roadMaka	*roadKhar*	240 (40%)	600
roadMaka	*roadPetr*	240 (40%)	
roadMaka	*roadTram*	120 (20%)	
roadGaga	*roadMaka*	240 (40%)	600
roadGaga	*roadKhar*	240 (40%)	
roadGaga	*roadTram*	120 (20%)	
roadPetr	*roadMaka*	120 (40%)	300
roadPetr	*roadGaga*	90 (30%)	
roadPetr	*roadKhar*	60 (20%)	
roadPetr	*roadTram*	30 (10%)	

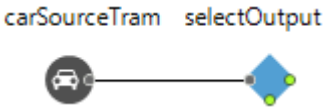
3. Доработать диаграмму процесса. Самый очевидный способ построить полную диаграмму процесса – добавить последовательности блоков ***CarSource***, ***CarMoveTo***, ***CarDispose*** для каждого допустимого пути. Однако в этом случае неизбежно дублирование некоторых элементов, из-за чего диаграмма процесса может ока-

заться громоздкой. Вместо этого воспользуемся блоком, который распределяет входящих агентов (автомобили) в один из выходных портов в зависимости от выполнения условия, заданного, например, с помощью вероятностей. Разберем по порядку:

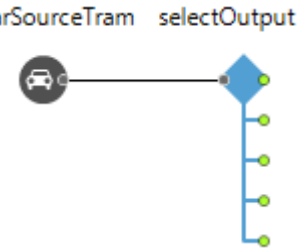
3.1 Добавьте блоки ***CarSource*** один под другим для каждой дороги, на которой должны появляться автомобили. Оставив свободное место, отступите вправо и добавьте по одному блоку ***CarMoveTo*** для каждой дороги, по которой автомобили должны уезжать. Чтобы было удобно ориентироваться среди элементов диаграммы, задавайте блокам понятные имена по аналогии с тем, как это делалось ранее для дорог. В свойствах добавленных блоков правильно укажите дороги и определите основные или встречные полосы, как это делалось на прошлом практическом занятии (на этом шаге часто допускаются ошибки, будьте внимательны).

3.2 Справа от каждого блока ***CarSource,*** для которого требуется разделить поток автомобилей, добавьте из палитры **Библиотека моделирования процессов** один из следующих блоков:

◇***SelectOutput,*** если поток требуется разделись на два направления;



SelectOutput5, если поток требуется разделить более чем на два направления.



3.3 Соедините выходные порты блоков ***SelectOutput*** с входными портами нужных блоков ***CarMoveTo***. Лишние выходные порты оставьте как есть.

3.4 Справа от диаграммы процесса добавьте блок ***CarDispose***. К одному и тому же порту блока может подходить сразу несколько соединительных линий, поэтому выходные порты всех блоков ***CarMoveTo*** соедините с входным портом единственного блока ***CarDispose***.

3.5 Убедившись, что включена привязка к сетке, приведите диаграмму процесса к понятному и строгому виду: двойным кликом добавляйте и удаляйте точки соединительных линий для поворотов, передвигайте и выравнивайте блоки. Пересекаться могут

любые линии, а накладываться только те, которые подходят к одному и тому же порту. На рис. 4.1 ниже представлена полная диаграмма процесса, моделирующая потоки транспортных средств на дорожной развязке из рассматриваемого примера.

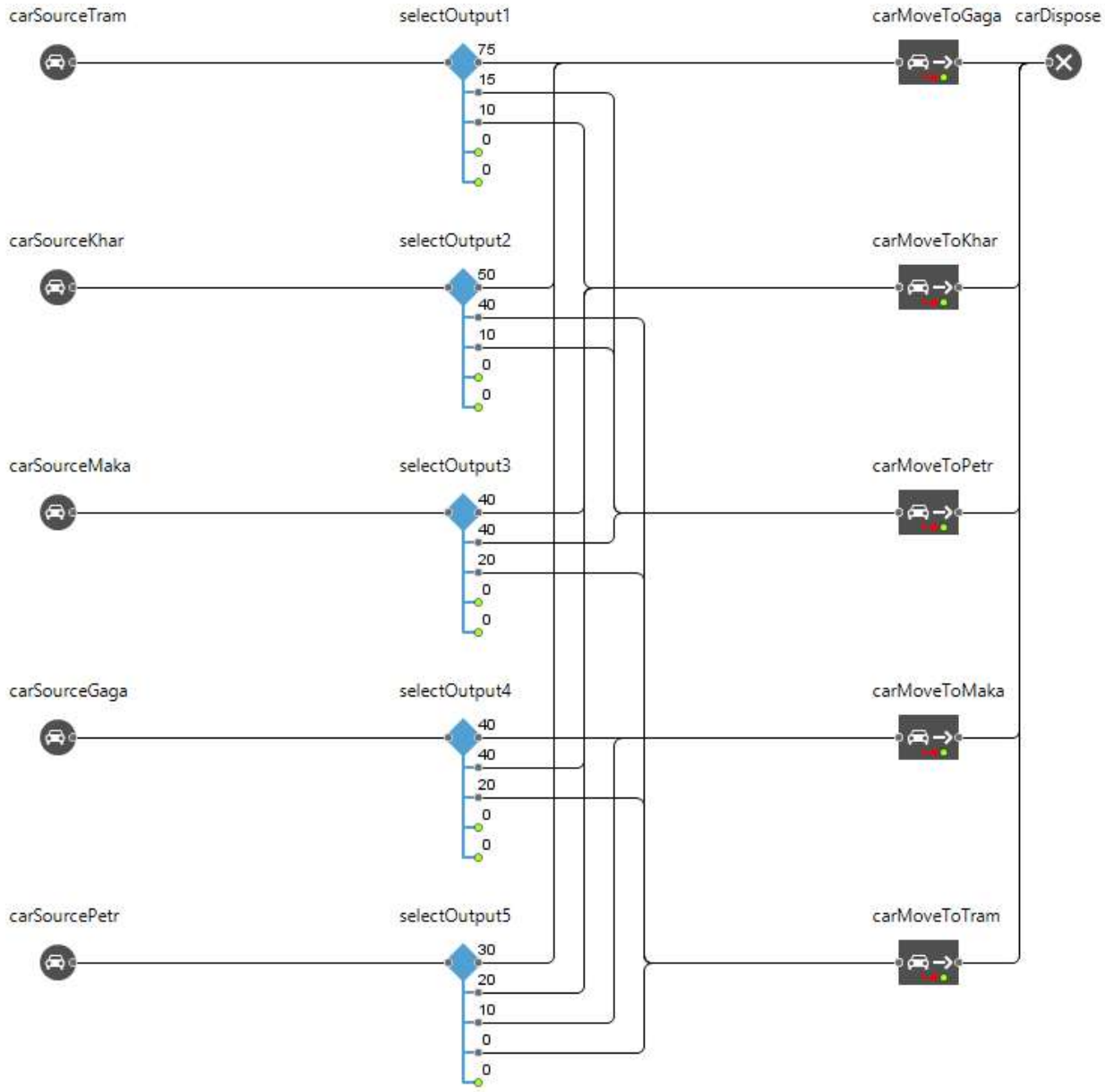


Рис 4.1. Диаграмма процесса движения автомобилей

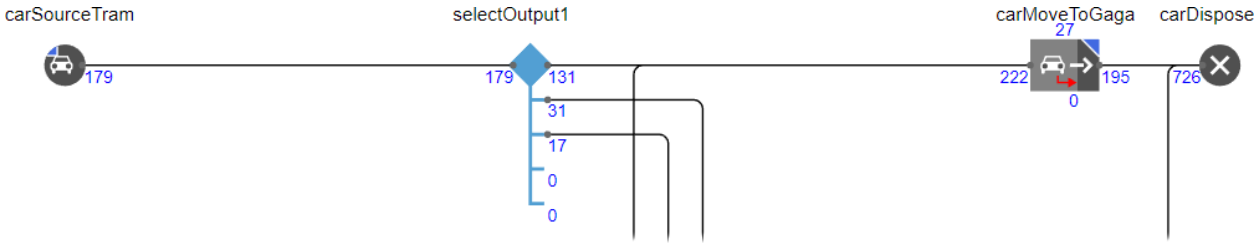
4. Задать интенсивности потоков автомобилей. Прибытие автомобилей указывается в свойствах блоков ***CarSource*** согласно интенсивности, времени между прибытиями, таблице базы данных и т.д. Мы воспользуемся первым способом как наиболее простым. Внесите значения из последнего столбца вашей таблицы в свойства блоков ***CarSource***. Затем основываясь на данных третьего столбца таблицы укажите в свойствах блоков ***SelectOutput*** и ***SelectOutput5***

вероятности с которыми автомобили будут выбирать то или иное направление движения в порядке следования выходных портов от верхнего к нижнему. Свободным портам должна быть задана нулевая вероятность выбора.

5. Исправить возможные ошибки и запустить модель. Убедитесь, что все соединительные линии доведены до портов нужных блоков и не подсвечены красным. Проверьте доступность путей, заданных в диаграмме процесса, и корректность свойства *Помещать на полосу* блоков ***CarSource*** и ***CarDispose***.

Исправьте возможные ошибки, препятствующие успешному построению и запуску модели: основная информация об ошибке, а также имя проблемного блока обычно содержатся в ее тексте, элемент выделяется в окне симуляции. В результате при симуляции у вас должны двигаться автомобили, обозначаемые цветными прямоугольниками, по всем возможным путям согласно диаграмме процесса. При необходимости, измените скорость выполнения модели, используя кнопки панели инструментов **Замедлить** и **Ускорить**. Обратите внимание, что для каждой модели, созданной с помощью объектов **Библиотеки дорожного движения**, автоматиче-

ски создается наглядная визуализация на диаграмме процесса. Она отображает количество автомобилей, которые прошли через блок или находятся в блоке в данный момент.



Убедитесь в корректности движения потоков автомобилей, сохраните модель и покажите результат преподавателю.

6. Подготовить и оформить отчёт о проделанной работе. В отчет обязательно включите вашу таблицу с направлениями потоков и интенсивностями прибытия, снимок диаграммы процесса и снимок модели в режиме симуляции в момент присутствия автомобилей на всех дорогах.

Контрольные вопросы

1. Опытным путем выясните: как свойство *Случайная полоса* блока ***CarSource*** влияет на генерацию автомобилей в модели?
2. Как ведут себя автомобили на перекрестке вашего индивидуального задания до и после установки приоритетов движения?
3. Как по-вашему должна быть изменена или дополнена модель для уменьшения автомобильных пробок при симуляции?

Практическое занятие №5

РАЗМЕЩЕНИЕ СВЕТОФОРОВ

Цель занятия: научиться регулировать движение автомобилей через перекрестки с помощью светофоров в моделях *AnyLogic*.

Программа занятия

1. Отключить авторазрешение заторов.
2. Добавить на перекрестки светофоры.
3. Задать логику работы светофоров.
4. Построить и запустить модель.
5. Подготовить и оформить отчет о проделанной работе.

Пояснения к занятию

На этом практическом занятии мы сделаем перекрестки регулируемыми, добавив светофоры. Предварительно найдите светофоры (если они присутствуют) на панорамах дорог, подходящих к перекресткам с разных сторон, и выясните, какие стрелки и дополнительные секции нужно будет учесть.

1. Отключить авторазрешение заторов. Затор – это ситуация, в которой автомобили создают друг другу помеху и не могут продолжить движение. По умолчанию библиотека дорожного движения определяет наличие заторов и время от времени позволит автомобилям проезжать сквозь друг друга, чтобы избежать полной блокировки модели. Для проведения корректных испытаний нужно отключить эту функцию.

Добавьте на диаграмму блок ***Road Network Descriptor*** из палитры **Библиотека дорожного движения**. В свойствах блока выберите имя нарисованной вами сети дорог `roadNetwork` в раскрывающемся списке *Дорожная сеть*. Этот блок не нужно соединять ни с какими другими блоками диаграммы процесса. С его помощью разработчики получают доступ к управлению всеми транспортными средствами, находящимися в одной дорожной сети. Блок позволяет задавать действия, которые будут выполняться при добавлении автомобиля в дорожную сеть, въезде на дорогу, остановке автомобиля, смене полосы и т.д. Откройте секцию свойств **Дополнительные настройки** и снимите флажок *Авторазрешение заторов*.

Кроме того, с помощью этого блока вы можете включить отображение пробок на дорогах. Для этого откройте секцию свойств **Карта пробок на дороге** и временно установите флажок *Показывать пробки*. Карта пробок представляет собой полупрозрачное окрашивание дороги в цвет, который зависит от средней скорости автомобиля на каждой секции дороги.

Запустите модель и убедитесь, что поведение автомобилей изменилось и теперь при высокой интенсивности их прибытия стал затруднительным проезд через нерегулируемые перекрестки. Возможно возникновение больших автомобильных пробок, которые будут подсвечиваться красным, если установлено свойство *Показывать пробки*.

2. Добавить на перекрестки светофоры. Перетащите на диаграмму блок ***Traffic Light*** из палитры **Библиотека дорожного движения**. У этого блока также нет портов, и его не нужно соединять ни с какими другими блоками диаграммы процесса. Откройте свойства блока. В свойстве *Перекресток* выберите имя нужного перекрестка, либо выберите его на диаграмме. Обратите внимание на ключевой параметр блока: *Задает режим работы для*. Три альтернативные опции определяют, для каких элементов разметки светофор задает режим работы:

- **Стоп-линий перекрестка** – используется для обычных светофоров (без дополнительных секций), когда нужно контролировать движение транспорта у стоп-линий перекрестка одинаково для всех полос примыкающей дороги.
- **Соединителей полос перекрестка** – используется для светофоров с дополнительными секциями, когда разрешенные направления движения машин во время одной и той же фазы светофора могут различаться для разных соединителей полос перекрестка.
- **Заданных стоп-линий** – используется для светофоров на пешеходных переходах, когда светофоры регулируют движение машин на одном или нескольких пешеходных переходах, заданных элементами ***Стоп-линия***.

Аналогичным образом добавьте в модель другие светофоры, если требуется. Так, в рассматриваемом примере имеется два перекрестка, которым даны имена `intersectionT` и `intersectionX`. На

первом, Т-образном, должен располагаться светофор с дополнительной секцией, поэтому он задает режим работы для соединителей полос перекрестка. На втором, Х-образном, должен располагаться обычный светофор (без дополнительных секций), поэтому он задает режим работы для стоп-линий перекрестка. Дадим им имена *trafficLightT* и *trafficLightX*.

3. Задать логику работы светофоров. В свойствах блока ***TrafficLight*** есть таблица *Фазы*. Здесь вы можете задать режим работы светофора – как будут меняться красная, зеленая (а опционально и желтая) фазы для входящих в выбранный перекресток стоп-линий или соединителей полос, в зависимости от того, какую опцию вы выбрали ранее. По умолчанию таблица содержит два столбца, по одному для каждой фазы.

а) *Светофор без дополнительных секций*. В таблице *Фазы* в должны отображаться стоп-линии перекрестка. Здесь вы можете задать изменение фазы для входящих в выбранный перекресток стоп-линий. Щелкните по заголовку столбца первой фазы (там, где указана ее длительность). Вы увидите, что в графическом редакторе все фигуры будут временно скрыты, а стоп-линии выделены красным цветом. Это означает, что мы находимся в режиме редактирования фазы, который позволяет менять состояние каждой стоп-линии в этой фазе прямо в графическом редакторе. Щелкните по стоп-линии и ее цвет изменится с красного на зеленый как в графическом редакторе, так и в таблице *Фазы*. Сделайте первую фазу светофора зеленой и для другой стоп-линии, расположенной на противоположной стороне перекрестка (если есть). Аналогичным образом щелкните по заголовку столбца второй фазы и настройте ее, сделав цвета стоп-линий противоположными первой фазе, как показано на рис. 5.1.

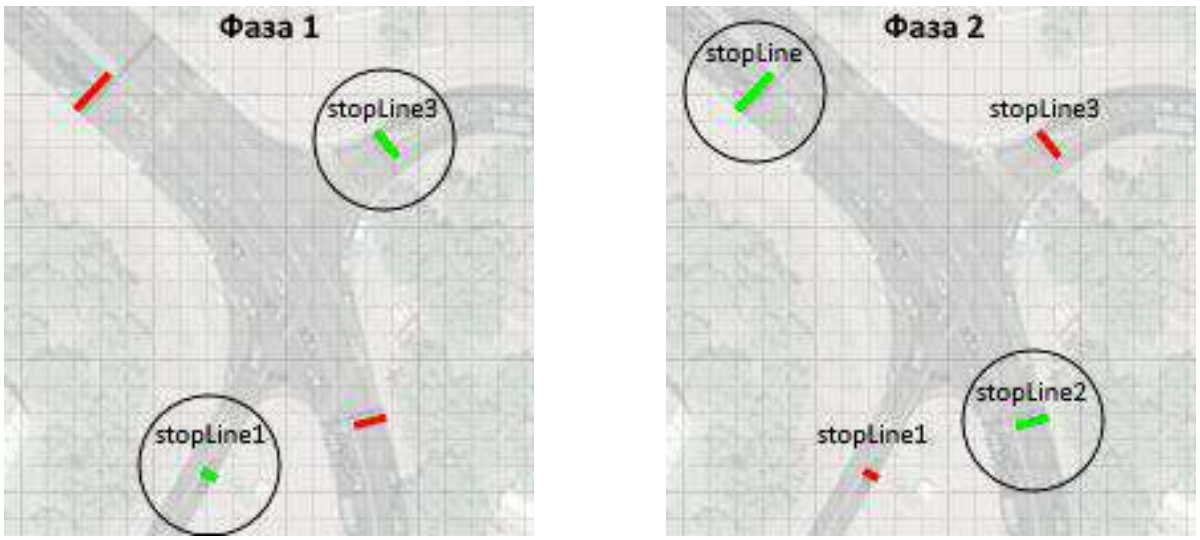


Рис. 5.1. Настройка стоплиний регулируемого перекрестка

b) *Светофор с дополнительными секциями*. В таблице *Фазы* в должны отображаться соединители полос перекрестка. Здесь вы можете задать изменение фазы для входящих в выбранный перекресток соединителей полос. Щелкните по заголовку столбца первой фазы (там, где указана ее длительность). Вы увидите, что в графическом редакторе все фигуры будут временно скрыты, а соединители полос выделены красным цветом. Это означает, что мы находимся в режиме редактирования фазы, который позволяет менять состояние каждого соединителя полос в этой фазе прямо в графическом редакторе. Щелкните по соединителю полос и его цвет изменится с красного на зеленый как в графическом редакторе, так и в таблице *Фазы*. Сделайте первую фазу светофора зеленой и для других соединителей полос, начинающихся на этой же и противоположной стороне перекрестка (если есть). Аналогичным образом щелкните по заголовку столбца второй фазы и настройте ее, сделав цвета соединителей полос противоположными первой фазе. Для того, чтобы учесть дополнительную секцию светофора достаточно сделать нужные соединители полос зелеными в обеих фазах, как показано на рис. 5.2. В рассматриваемом примере на Т-образном перекрестке дополнительная секция разрешает правый поворот с юго-востока на северо-восток по соединителю полос *laneConnector33*.

Независимо от типа светофора нужно добавить отсутствующие желтые фазы, за время которых автомобили будут успевать покинуть перекресток.

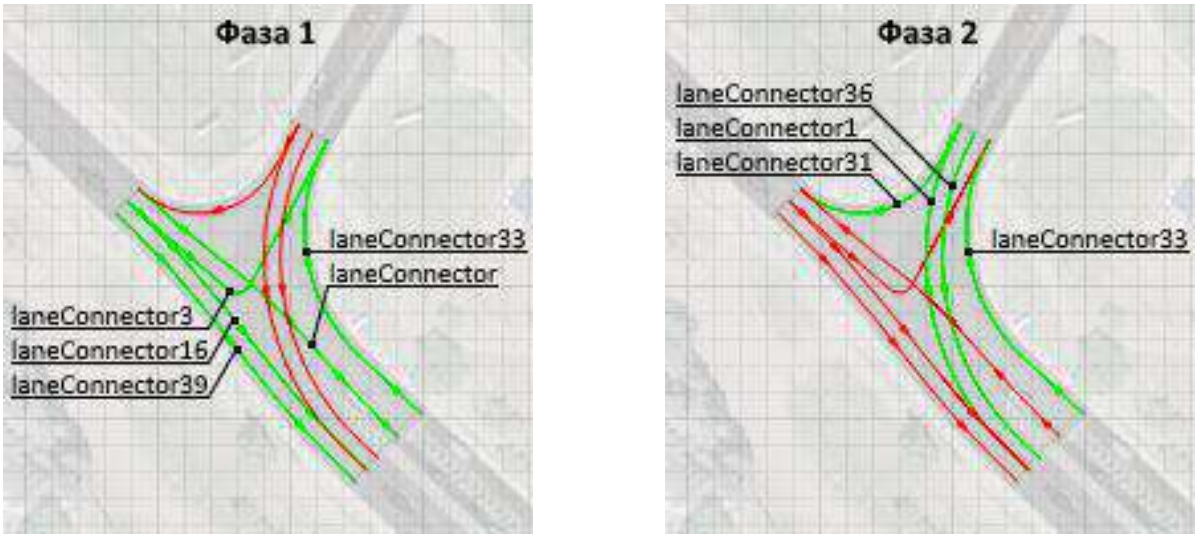


Рис 5.2. Настройка соединителей полос регулируемого перекрестка

Дважды кликните по кнопке **Добавить желтую фазу**, и с помощью стрелок **Переместить влево** и **Переместить вправо** разместите их так, чтобы они чередовались с изначальными фазами.

4. Построить и запустить модель. Пронаблюдайте, как изменилось дорожное движение после добавления светофоров. Ускорьте процесс моделирования и оцените объем трафика, скапливающегося перед перекрестками. На основе полученного опыта пытайтесь найти более удачные значения длительности фаз светофора для каждого перекрестка и внесите соответствующие изменения в таблицу *Фазы*. Имейте в виду, что желтая фаза при моделировании должна включать в себя и то время, которое красный свет должен гореть для всех направлений, что актуально для больших перекрестков. Описанным образом были получены данные таблицы фаз для блоков `trafficLightT` (рис. 5.3, а) и `trafficLightX` (рис 5.3, б) из рассматриваемого примера.

а) б)

Рис 5.3. Настройка фаз светофоров:
а – на основе соединителей полос; б – на основе стоплиний

Также можете попробовать добавить и настроить дополнительные фазы, например, позволяющие выезд только с одной дороги для беспрепятственного проезда через перекресток в любых направлениях.

5. Подготовить и оформить отчет о проделанной работе. В отчет помимо прочего обязательно включите снимки перекрестка в режиме настройки каждой красно-зеленой фазы, аналогично тому, как это сделано в методических указаниях. На каждой фазе так же подпишите названия зеленых соединителей полос, по которым разрешено движение.

Контрольные вопросы

1. В чем заключается основное отличие моделирования обычных светофоров и светофоров с дополнительными секциями?
2. Как добавляются и убираются фазы светофоров?
3. Как организовано движение на вашем участке дорожной сети после добавления в модель светофоров?

Практическое занятие №6

ОПТИМИЗАЦИЯ СВЕТОФОРНЫХ ФАЗ

Цель занятия: получить навыки оптимизации дорожного движения посредством проведения оптимизационного эксперимента в моделях *AnyLogic*.

Программа занятия

1. Сформировать критерий оптимальности.
2. Добавить оптимизируемые параметры.
3. Подготовить оптимизационный эксперимент.
4. Выполнить оптимизационный эксперимент.
5. Построить и запустить модель с найденными параметрами.
6. Подготовить и оформить отчёт о проделанной работе.

Пояснения к занятию

1. Сформировать критерий оптимальности. Автоматическая настройка фаз светофоров возможна посредством проведения оптимизационного эксперимента. Для его подготовки нужно определить критерий оптимальности и варьируемые параметры, изменение которых позволяет влиять на эффективность моделируемого процесса.

Существуют разные варианты критерия оптимальности при моделировании дорожного движения. Вот некоторые из них:

- среднее время проезда автомобилей через участок дорожной сети;
- предельно возможная интенсивность прибивания автомобилей без образования затора (пропускная способность);
- наибольшее количество автомобилей, одновременно присутствующих на участке дорожной сети.

Эти и другие критерии приводят к немного отличающимся оптимальным параметрам, но как правило имеют корреляционную связь друг с другом. Мы остановимся на наиболее простом последнем варианте критерия оптимальности.

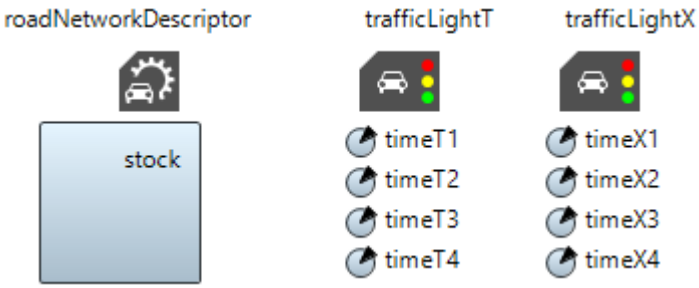
Количество автомобилей в дорожной сети хранит блок ***RoadNetworkDescriptor***. Однако нам нужно получить наибольшее количество не в конкретный момент, а за все время симуляции. Перей-

дите в библиотеку **Системная динамика** и перетащите на диаграмму элемент ***Накопитель*** чуть ниже ранее добавленного блока `roadNetworkDescriptor`. Появится прямоугольник с именем `stok`. Чтобы этот элемент хранил нужное значение наибольшего количества автомобилей, откройте свойства блока `roadNetworkDescriptor` и в секции **Действия** изменить поле **При входе в сеть**, добавив туда две строки кода:

```
if (roadNetworkDescriptor.size()>stock)
  stock = roadNetworkDescriptor.size()
```

Таким образом при генерации каждого нового автомобиля будет выполняться действие, которое словесно можно описать так: «*Если текущее количество автомобилей в дорожной сети, хранящееся в блоке* `roadNetworkDescriptor`*, больше величины накопителя* `stock`*, то записать это количество в качестве новой величины накопителя* `stock`».

2. Добавить оптимизируемые параметры. Теперь добавим варьируемые параметры, которые будут определять длительности фаз светофоров. Из библиотеки **Системная динамика** перетащите на диаграмму несколько элементов ***Параметр*** чуть ниже ранее добавленных блоков `trafficLight`. Количество таких параметров должно равняться суммарному числу фаз светофоров. Задайте периметрам понятные имена (например, `time1`, `time2` и т.д.). Укажите *Тип* – время, а *Единицы измерения* – секунды. После этого каждому имеющемуся в модели блоку `trafficLight` в заголовке таблицы *Фаза* нужно указать имена ваших параметров вместо конкретных значений длительностей фаз.



3. Подготовить оптимизационный эксперимент. Создайте новый эксперимент командой **Файл > Создать > Эксперимент**. Появится диалоговое окно **Новый эксперимент**. Выберите **Оптимизация** из списка **Тип эксперимента** и щелкните по кнопке **Готово**.

На панели Проекты дважды кликните по появившемуся пункту *Optimization*, чтобы открыть диаграмму оптимизационного эксперимента. Перейдите на панель **Свойства**. В поле *Целевая функция* введите величину *root.stock*, которую нужно *минимизировать*. Таким образом будет происходить обращение к накопителю *stock*, агента верхнего уровня (*Main)*.

Количество итераций указывает на число наборов значений параметров, для которых будет повторяться эксперимент. Поскольку эксперимент можно прервать в любой момент вручную, можно указать достаточно большое значение, например, 500 итераций.

В секции **Параметры** должна отображаться таблица со всеми созданными вами параметрами. Укажите для них **Тип** дискретный, чтобы длительности фаз были округлены до целых секунд. Укажите минимальное и максимальное значения параметров (обычно от 10 до 90 секунд для красно-зеленых фаз и от 3 до 10 секунд для желтых фаз). Укажите шаг (5 для красно-зеленых фаз и 1 для желтых).

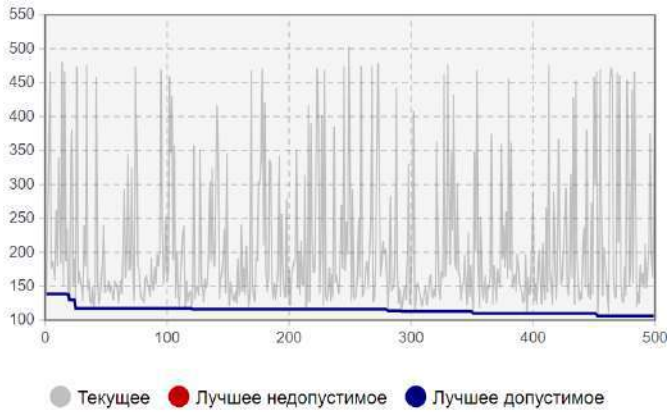
Бесплатная учебная версия *AnyLogic* допускает использование не более 7 варьируемых параметров. Поэтому для остальных нужно выбрать **Тип** фиксированный и задать свое значение, которое не будет меняться в ходе эксперимента. В рассматриваемом примере для двух светофоров создано 8 параметров, поэтому длительность одной из желтых фаз (*TimeX4*) установлена фиксированной.

В секции **Модельное время** укажите в секундах **Конечное время**, например, 1800 (полчаса). В верхней части свойств найдите и нажмите кнопку **Создать интерфейс**. При этом будет создан использующийся по умолчанию интерфейс эксперимента – на презентацию будут добавлены элементы управления, отображающие текущий прогресс оптимизации и облегчающие управление процессом.

4. Выполнить оптимизационный эксперимент. Теперь можно запустить эксперимент, выбрав *Optimization* в меню запуска. По окончанию эксперимента (либо если его выполнения идет дольше приемлемого времени) скопируйте лучшие найденные значения и укажите их в свойствах параметров на диаграмме *Main* в качестве *Значений по умолчанию*.

Road Traffic Common New : Optimization

	Текущее	Лучшее
Итерация:	500	455
Функционал: ↓	453	106
Параметры		Copy best
timeT1	80	80
timeT2	5	4
timeT3	70	70
timeT4	4	3
timeX1	45	45
timeX2	4	4
timeX3	50	50
timeX4	8	8



5. Построить и запустить модель с найденными параметрами. Устраните ошибки, препятствующие построению и запуску модели. Поскольку последним запускался оптимизационный эксперимент, нужно выбрать в меню запуска *Simulation*. После успешного запуска модели пронаблюдайте, как происходит дорожное движения при полученных оптимальных параметрах для заданных условий выполненного оптимизационного эксперимента. Включите ускоренный режим и оцените, насколько эффективно светофоры теперь справляются со своей задачей.

6. Подготовить и оформить отчёт о проделанной работе. В отчет обязательно включите снимок результата оптимизационного эксперимента. Сделайте выводы о выполненной вами оптимизации светофорных фаз: удалось ли добиться меньшего числа автомобилей на вашем участке дорожной сети и возникают ли заторы?

Контрольные вопросы

1. Что такое оптимизация?
2. Что необходимо для организации оптимизационного эксперимента в *AnyLogic*?
3. Какие еще свойства элементов модели дорожного движения на ваш взгляд можно использовать в качестве оптимизируемых параметров для модернизации участка дорожной сети?

Практическое занятие №7

ДОБАВЛЕНИЕ ПАРКОВОК И АВТОБУСНЫХ МАРШРУТОВ

Цель занятия: научиться добавлять парковки и автобусные маршруты в модель дорожного движения в *AnyLogic*.

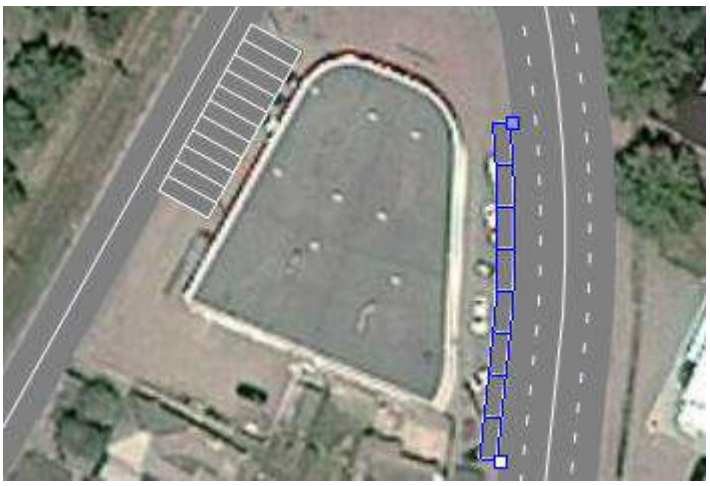
Программа занятия

1. Добавить парковки.
2. Создать процесс для парковок.
3. Добавить автобусные остановки.
4. Создать процесс для автобусных маршрутов.
5. Построить и запустить модель
6. Подготовить и оформить отчет о проделанной работе.

Пояснения к занятию

В ходе этого практического занятия вы дополните моделируемую дорожную сеть парковками и автобусными остановками, а также внесете изменения в диаграмму процесса для их использования. С помощью спутникового снимка и панорам определитесь с местоположением по крайней мере двух автобусных остановок и одной парковки на вашем участке дорожной сети. Если обнаружить их не удается, выберите местоположения на свое усмотрение.

1. Добавить парковки. Перетащите элемент ***Парковка*** из палитры **Библиотека дорожного движения** на графическую диаграмму. Перетаскивая парковку по диаграмме, вы увидите, что все фигуры, кроме дорог, будут временно скрыты. Поместите курсор у дороги в том месте, где должна появиться парковка. Парковка прикрепится к дороге. По умолчанию парковка создается с пятью парковочными местами. Если нужно переместить парковку, то это можно сделать, перетащив ее с помощью мыши. В свойствах можете выбрать параллельный или перпендикулярный *Тип парковки*, задать *Количество парковочных мест* и *Длину парковочного места*.



2. Создать процесс для парковок. Теперь нужно добавить в диаграмму процесса блоки, которые будут моделировать заезд автомобилей на парковку и пребывание там в течение определенного времени. С учетом местоположения парковок на графической диаграмме выясните, где именно на диаграмме процесса должны появиться изменения. Если посещение парковки должно происходить перед тем, как автомобиль доедет до конца полосы дороги, указанной в одном из блоков ***carMoveTo***, то добавлять новые блоки нужно именно перед ним. Если же парковка находится на полосе, в начале которой где генерируются автомобили одним из блоков ***carSource***, добавлять блоки нужно сразу после него.

На место соединительной линии, где планируете создать для парковки процесс, добавьте блок ***selectOutput***, который позволит направить на парковку лишь малую часть автомобилей от общего потока. Укажите значение вероятности, близкое к единице (например, значение 0.95 приведет к тому, что 95 % автомобилей проедут мимо). Ниже добавьте блок ***carMoveTo***. В его свойствах укажите *Цель движения: парковка* и затем выберите в поле *Парковка* имя созданной ранее парковки. Теперь нужно промоделировать, как приехавшие на парковку автомобили пребывают на ней в течение определенного времени. Для этого из **Библиотеки моделирования процессов** перетащите блок ***Delay***. Данный блок моделирует задержку, связанную с выполнением агентом определенной операции (в нашем случае - ожидание на парковке). Откройте свойства этого блока. *Время задержки* можно задать случайное с помощью треугольной функции, например, *`triangular(1,10,30) минут`*. Первое число в скобках определит минимальное время парковки, второе – самое вероятное, третье – максимальное. Рядом с полем список с возможными единицами измерения времени. Также укажите

Вместимость, равную количеству парковочных мест. Этот параметр задает максимальное количество агентов (в нашем случае – автомобилей), которое может одновременно находиться в данном блоке.

Назовите добавленные блоки понятным образом и соедините их так, как это показано на рис. 7.1. Как вы могли заметить, у блока ***CarMoveTo*** помимо двух стандартных портов на левой и на правой границе блока есть и еще один, помещенный на нижнюю границу значка. К нему ведет красная стрелочка. Этот порт называется *outWayNotFound* и представляет собой выходной порт, в который перенаправляются те автомобили, для которых не удалось построить маршрут к заданной в этом блоке цели движения. В нашем случае автомобили будут направляться в этот порт, если на момент поступления автомобиля в блок `carMoveToPark1` на парковке не окажется свободных мест. Таким образом, если парковка будет полностью занята, то автомобили будут просто продолжать движение дальше по дороге. Свободные концы соединительных линий должны быть подключены к портам тех же блоков, к которым были они подключены изначально, до внесения изменений

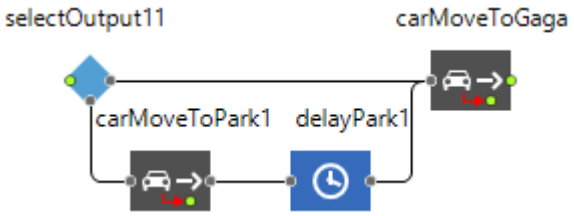


Рис 7.1. Диаграмма процесса парковки

3. Добавить автобусные остановки. Перетащите элемент ***Автобусная остановка*** из секции **Разметка пространства** палитры **Библиотека дорожного движения** на графическую диаграмму. Чтобы удлинить остановку выделите ее щелчком мыши и затем перетащите квадратный маркер с краю.

Если для автобусной остановки не предусмотрен «карман», то можете организовать ее прямо на полосе движения с помощью элемента ***Стоп-линия***. Перетащите стоп-линию в то место, где должна появиться остановка. Выделите ее и перемещая круглый маркер уменьшите ширину стоп-линии так, чтобы она заняла только крайнюю правую полосу.

4. Создать процесс для автобусных маршрутов. Ниже всей диаграммы процесса добавим блоки, моделирующие движение ав-

тобусов по маршрутам. Всем добавляемым блокам давайте имена по аналогии с блоками для легковых автомобилей, заменяя приставку `car` на `bus`.

Для генерации автобусов перетащите на диаграмму блок ***CarSource***. Этот блок будет моделировать появление автобусов. Автобусы должны появляться в модели существенно реже, чем легковые автомобили, поэтому задайте *Интенсивность* появления не более 20 в час. Правильно выберите значения свойств *Дорога* и *Помещается на полосу*. Чтобы автобусы появлялись только на нужной полосе для общественного транспорта, снимите флажок *Случайная полоса*, а ниже укажите *Номер полосы*. Нумерация полос идет в направлении от обочины к разделительной полосе и начинается с нуля. Следовательно, для указания крайней правой полосы появления автобусов нужно ввести `0` в поле *Номер полосы*. Раскройте секцию свойств **Автомобиль** и в поле *Длина* задайте длину создаваемого автобуса: 10 метров. В поле *Начальная скорость* введите 40 км/ч.

Рядом добавьте блок ***CarMoveTo***. Он будет моделировать движение автобусов к остановке. В свойствах блока выберите *Цель движения: автобусная остановка* либо *Цель движения: стоп-линия*, в зависимости от того, какой элемент разметки вы использовали для обозначения автобусной остановки. Затем выберите в поле *Автобусная остановка* имя созданной нами ранее остановки либо стоп-линии. Если использовалась стоп-линия, то нужно также задать *Действие у стоп-линии: Остановиться*.

Чтобы промоделировать, как приехавшие на остановку автобусы пребывают на ней в течение определенного времени добавьте блок ***Delay*** и настройте его по аналогии с тем, как это делалось для парковки.

Если один маршрут включает несколько остановок, добавьте для них нужное количество подряд идущих последовательностей нужных блоков. Чтобы автобус покинул участок дорожной сети добавьте еще один блок ***CarMoveTo***, указав для него дорогу и полосу, до конца которой автобус должен доехать. Последним добавьте блок ***carDispose*** для удаления автобусов из модели.

На рис. 7.2 показан рассматриваемый пример с расположением двух парковок и пяти остановок, одна из которых задана стоплинией.

Рис 7.2. Пример размещения парковок и остановок в модели

5. Построить и запустить модель. Устраните все ошибки, постройте и запустите модель. Вы увидите, как время от времени легковые автомобили будут заезжать на парковки, автобусы – на автобусные остановки, а после случайного по времени ожидания продолжать свое движение. Попробуйте повторно выполнить оптимизационный эксперимент, уже подготовленный на прошлом практическом занятии, чтобы получить новые оптимальные значения длительностей светофорных фаз, с учетом появившихся изменений в модели.

6. Подготовить и оформить отчет о проделанной работе. В отчет обязательно добавьте снимок, демонстрирующий использование транспортными средствами парковок и автобусных остановок, а также окончательный вид всей диаграммы процесса, убедившись,

что все блоки имеют понятные имена. На рис. 7.3. дана окончательная диаграмма процесса для рассматриваемого примера.

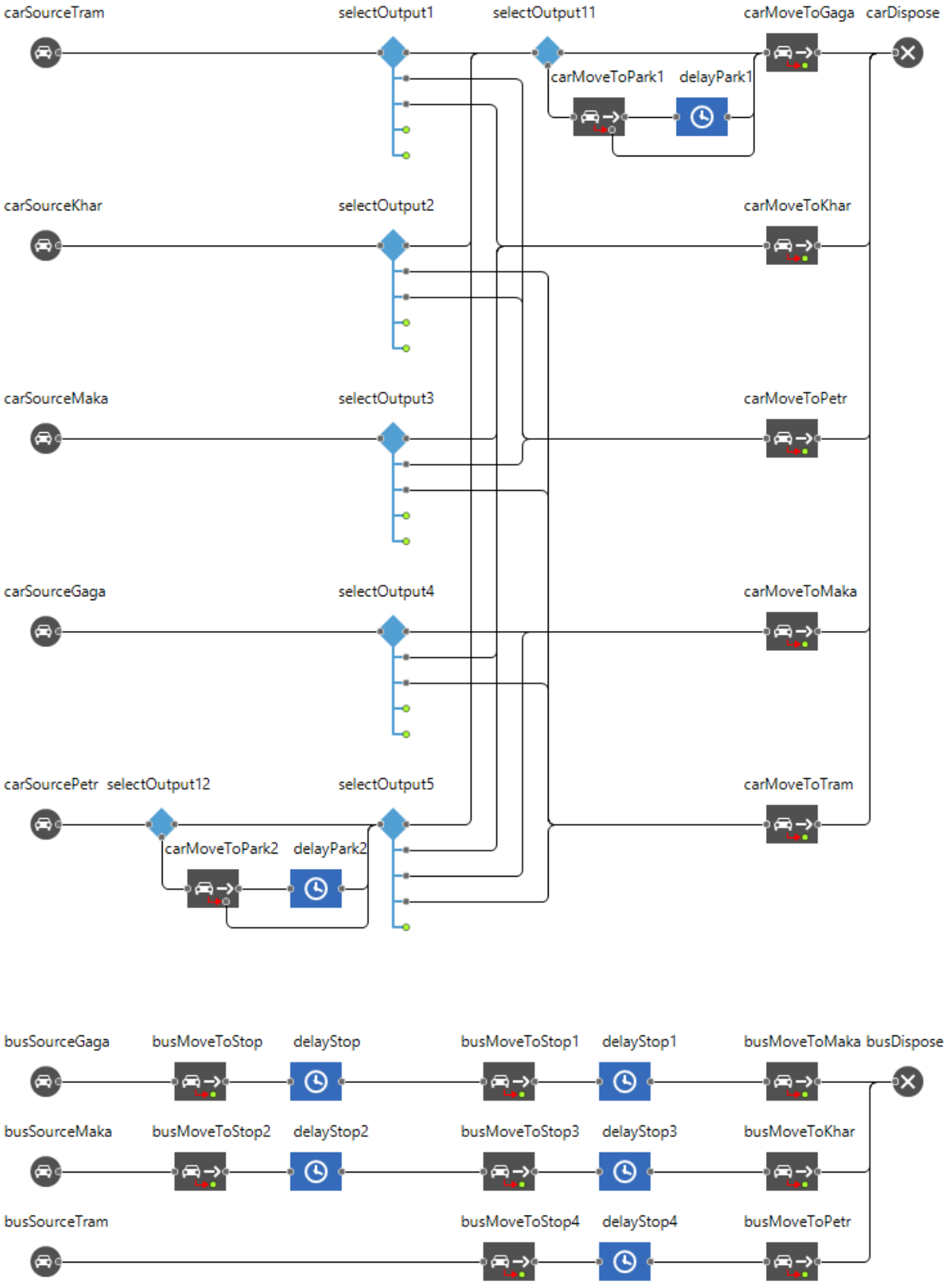


Рис 7.3. Полная диаграмма процесса движения автомобилей

Контрольные вопросы

1. Для чего время ожидания на парковке или автобусной остановке задается треугольной функцией распределения, а не фиксированным значением?

2. Как появление парковок и автобусных маршрутов повлияло на дорожную ситуацию?

3. Изменились ли оптимальные значения параметров после повторного проведения оптимизационного эксперимента? Как бы вы это объяснили?

Практическое занятие №8

СОЗДАНИЕ 3*D* АНИМАЦИИ

Цель занятия: научиться создавать трехмерные сцены с анимацией для модели дорожного движения в *AnyLogic*.

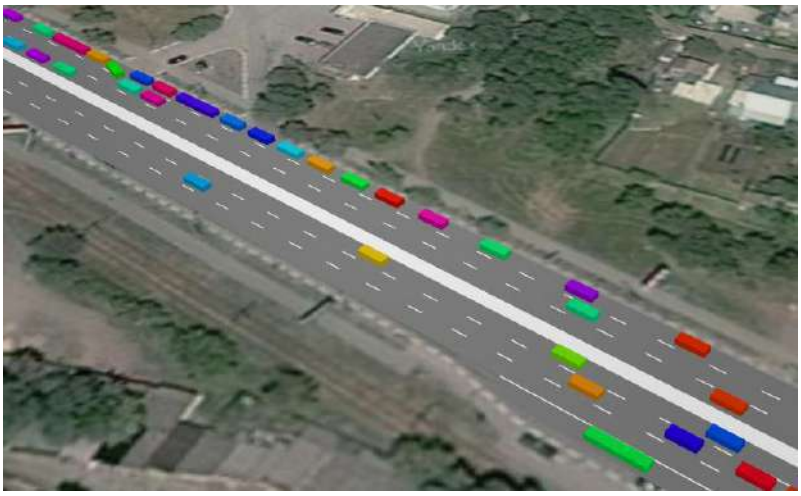
Программа занятия

1. Добавить 3*D* окно.
2. Изменить *Z*-уровень рисунка.
3. Создать новые типы агентов.
4. Использовать новые типы агентов.
5. Построить и запустить модель.
6. Подготовить и оформить отчет о проделанной работе.

Пояснения к занятию

1. Добавить 3*D* окно. 3*D* окно используется для задания на диаграмме агента области, в которой во время запуска модели будет отображаться трехмерная анимация этой модели.

Откройте палитру **Презентация** и перетащите элемент ***3D Окно*** в графический редактор. Появившуюся закрашенную серым область поместите туда, где вы хотите видеть 3*D* анимацию во время запуска модели. Поскольку окно модели (прямоугольная синяя рамка) занято спутниковым снимком, мы расположим ***3D Окно*** выше. Уже сейчас вы можете запустить модель и наблюдать простую 3*D* анимацию, если во время выполнения модели прокрутите область просмотра туда, где разместили ***3D Окно***.



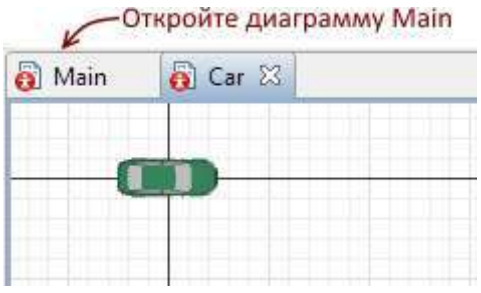
2. Изменить *Z*-уровень рисунка. Теперь надо решить проблему с чересполосицей в трехмерной анимации. Причина проблемы в том, что и дорожная сеть, и спутниковый снимок отображаются на одном *Z*-уровне (0), и поэтому в какие-то моменты на верхний уровень вылезает рисунок, а в какие-то – дорога.

Выделите изображение со спутниковым снимком моделируемой местности. Если изображение заблокировано, то для его выделения перейдите на панель **Проекты**, найдите и выберите там элемент *image* в ветке **Main > Презентация**. Откройте секцию свойств изображения **Местоположение и размер** и в поле *Z* введите: *-1*. Тем самым, мы поместим рисунок в трехмерной анимации чуть ниже, чем дорожную сеть.

3. Создать новые типы агентов. Теперь мы зададим автомобилям определенную фигуру анимации вместо используемых по умолчанию параллелепипедов. Это потребует создания новых типов автомобилей.

Откройте палитру **Библиотека дорожного движения** и перетащите элемент ***Тип автомобиля*** в графический редактор. Откроется диалоговое окно **Мастера создания агентов**. Оставьте заданное по умолчанию значения полей и нажмите **Далее.** Вы увидите следующую страницу **Мастера**, на которой предлагается выбрать фигуру анимации для этого агента. Поскольку мы хотим выбрать 3*D* фигуру, оставьте опцию **3D** выбранной. В списке ниже, выберите фигуру анимации **Автомобиль.**

Автоматически откроется диаграмма нового агента *Car*. В начале координат этой диаграммы вы увидите выбранную 3*D* фигуру для автомобиля. Переключитесь обратно на диаграмму *Main*.



На панели **Проекты** должен отобразиться новый тип агента – *Car*. Создаваемые с его помощью фигуры легковых автомобилей будут иметь одинаковый внешний вид по умолчанию.

Чтобы было удобнее отслеживать движение потоков автомобилей сделаем фигуры автомобилей отличающимися, например, по

цвету. Для этого добавьте еще несколько типов автомобилей с точно такими же фигурами. Их количество должно совпадать с количеством блоков ***carSource*** на диаграмме процесса. Затем на диаграмме каждого созданного типа агента выделите фигуру автомобиля в центре координат и в свойствах измените цвет кузова, задаваемый полем *Material__4__Surf*, на свое усмотрение.

Теперь аналогичным образом создайте тип автомобиля для автобусов. На первой странице **Мастера создания агентов** укажите **Имя нового типа** *Bus*, а на второй – выберите фигуру анимации автобуса.

4. Использовать новые типы агентов. На диаграмме *Main* в графическом редакторе по очереди выделяйте каждый блок ***carSource*** и на панели **Свойства** в секции **Автомобиль** из раскрывающегося списка **Новый автомобиль** выбирайте один из созданных типов автомобилей *Car*. Точно так же задайте нужный тип автомобиля *Bas* для блоков ***busSource***.

5. Построить и запустить модель. Чтобы переключиться к 3D анимации в запущенной модели, откройте панель разработчика, щелкнув по кнопке **Показать/скрыть панель разработчика** в правом углу панели управления. В открывшейся панели разработчика, раскройте список **Выбрать область и показать** и выберите опцию **[window3d]**.

Попробуйте перемещаться по трехмерной сцене с помощью описанных ниже команд навигации:

Чтобы	Выполните следующие действия
Переместить сцену	1. Нажмите левую кнопку мыши в области 3D окна и держите ее нажатой. 2. Передвиньте мышь в направлении перемещения.
Повернуть сцену	1. Нажмите клавишу Alt и держите ее нажатой. 2. Нажмите левую кнопку мыши в области 3D окна и держите ее нажатой. 3. Передвиньте мышь в направлении вращения.
Приблизить/отдалить сцену	1. Покрутите колесо мыши от/на себя в области 3D окна.

6. Подготовить и оформить отчет о проделанной работе. В запущенной модели подберите удачный ракурс и сделайте для отчета снимок 3*D* сцены в момент присутствия на ней агентов всех типов.

Контрольные вопросы

1. На что влияет *Z*-уровень элементов модели?
2. Какие новые типы агентов появились в модели?
3. Каким образом помимо уже существующих легковых автомобилей и автобусов в модель можно добавить грузовые автомобили с 3*D* анимацией?

Самостоятельная работа студентов (СРС)

СРС – темы для самостоятельного изучения, в том числе конспектирования – 91,1 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 1.** Обзор библиотек и нововведений среды для моделирования *AnyLogic*. Изучения демонстрационных примеров моделей в различных областях.	15	[4-5]
2	**Тема 2.** Моделирование логистических процессов и цепей поставок в *AnyLogic*. Планирование и оптимизация перевозок с помощью моделей.	15	[4-5]
3	**Тема 3.** Железнодорожная библиотека *AnyLogic*. Моделирование железнодорожных станций, сортировочных узлов и разгрузки поезда.	15	[4-5]
4	**Тема 4.** Библиотека производственных систем *AnyLogic*. Элементы разметки моделей производственных систем. Создание моделей конвейерных сетей.	15	[4-5]
5	**Тема 5.** Пешеходная библиотека *AnyLogic*. Разметка пространства пешеходной модели. Сбор статистики.	15	[4-5]
6	**Тема 6.** Основы *Java* в *AnyLogic*. Функции, константы и классы *AnyLogic*. Исправление ошибок и отладка моделей. Библиотеки и внешние *Jar* файлы. Интеграция моделей со сторонними *Java* приложениями. Командная разработка с использованием *SVN*.	16,1	[4-5]

Библиографический список

1. Березовская Е.А. Имитационное моделирование: учеб. пособие [Электронный ресурс]. – Ростов-н.-Д., Таганрог: Изд-во ЮФУ, 2018. – 76 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=499496 (дата обращения: 18.11.2019).

2. Современное образование: векторы развития. Цифровизация экономики и общества: вызовы для системы образования: материалы междун. конф. (г. Москва, МПГУ, 24– 25 апреля 2018 г.). Избранные статьи: сб. науч. тр. [Электронный ресурс]. – М.: МПГУ, 2018. – 376 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=500557

3. Национальная технологическая инициатива. Программа мер по формированию принципиально новых рынков и созданию условий для глобального технологического лидерства России к 2035 году [Электронный ресурс] // Агентство стратегических инициатив по продвижению новых проектов (АСИ). – Режим доступа: https://asi.ru/nti (дата обращения: 18.11.2019).

Дополнительная учебная литература

4. Григорьев И. AnyLogic за три дня: практ. пособие по имитационному моделированию [Электронный ресурс] // The AnyLogic Company, 2016. – 273 с. – Режим доступа: https://www.anylogic.ru/resources/books/free-simulation-book-and-modeling-tutorials (дата обращения: 18.11.2019).

5. Система справочной документации *AnyLogic* [Электронный ресурс] // The AnyLogic Company. – Режим доступа: https://help.anylogic.ru/ (дата обращения: 18.11.2019).

Учебное издание

Кубил Виктор Николаевич

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ДОРОЖНОГО ДВИЖЕНИЯ В СРЕДЕ *AnyLogic*

Учебное пособие по дисциплине «Цифровые технологии в инженерии»

Редактор *Я.В. Максименко*

Подписано в печать 13.08.2020.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 3,25. Уч.-изд. л. 3,5 . Тираж 10 экз. Заказ 46-0397.

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

В.А. Мохов, А.В. Кузнецова

СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА: СОВРЕМЕННЫЕ МЕТОДЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2021

УДК 004.8(075.8)
ББК 32.813 я73
М86

Рецензенты: кандидат технических наук, доцент, заведующий кафедрой
«Программное обеспечение вычислительной техники»
Гринченков Дмитрий Валерьевич,
кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

Мохов В.А., Кузнецова А.В.

М86 **Системы искусственного интеллекта: современные методы программной инженерии** : учебное пособие / В.А. Мохов, А.В. Кузнецова; Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2021. – 150 с.

ISBN 978-5-9997-0756-7

Пособие содержит материалы по основам искусственного интеллекта и применению его методов и инструментов в программной инженерии. Приведены примеры, направленные на решение задач программной инженерии: управление проектами программного обеспечения, работа со знаниями, разработка программных агентов и других. Включены тестовые вопросы, индивидуальное задание и кейсы для группового выполнения, которые ориентированы на углубленное изучение приёмов релевантного информационного поиска, разработки интеллект-карт и проектирования сервисных чат-ботов с целью их последующей реализации в виде программ и создания различных модификаций. Также по темам пособия приведен краткий конспект, ориентированный на студентов, имеющих продвинутый уровень подготовки.

Учебное пособие предназначено для студентов, изучающих дисциплину «Системы искусственного интеллекта», обучающихся по программам бакалавриата направлений подготовки 09.03.01 «Информатика и вычислительная техника» очной и заочной форм обучения, 09.03.04 «Программная инженерия», 02.03.03 «Математические основы и алгоритмизация информационных систем» и студентов, изучающих дисциплину «Современные методы программной инженерии», обучающихся по программам магистратуры направлений подготовки 09.04.01 «Информатика и вычислительная техника» очной и заочной форм обучения, 09.04.04 «Программная инженерия».

УДК 004.8(075.8)
ББК 32.813 я73

ISBN 978-5-9997-0756-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5

1. ОСНОВНЫЕ ПОНЯТИЯ И АЛГОРИТМЫ ПРИМЕНЕНИЯ 6

1.1. Основные понятия и история ИИ 6

1.2. Роль данных и знаний в ИИ 9

1.3. Вопросы применения алгоритмов и аппаратных средств 11

2. ПРИЛОЖЕНИЯ И ТЕХНОЛОГИИ В ОБЩЕСТВЕ 18

2.1. ИИ в компьютерных приложениях 18

2.2. Классификация технологий ИИ 20

2.3. ИИ в технике, медицине и общении 29

3. ПРОГРАММНЫЕ ПРИЛОЖЕНИЯ 32

3.1. Анализ и извлечение данных 32

3.2. Нейронные сети и машинное обучение 37

3.3. Глубокое обучение 40

4. АППАРАТНЫЕ ПРИЛОЖЕНИЯ 44

4.1. Агенты: роботы, дроны и другие 44

4.2. Мультиагентные системы 48

4.3. Беспилотные автомобили 53

5. РАБОТА СО ЗНАНИЯМИ 56

5.1. Онтологии 56

5.2. Интеллект-карты 63

5.3. Нечёткая логика 66

6. ЛУЧШИЕ ПОЛЕЗНЫЕ ДОСТИЖЕНИЯ 72

6.1. Наиболее яркие достижения разработчиков и создателей ИИ 72

6.2. Результаты ИИ за последнее десятилетие 75

6.3. Решения и проекты ИИ 2020 года 78

6.4. Что будет в ближайшие 10 лет? 82

7. ИНТЕЛЛЕКТУАЛЬНЫЕ ИНСТРУМЕНТЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ 85

7.1. ИИ в управлении проектами программного обеспечения 85

7.2. Виртуальные помощники и интеллектуальные инструменты 89

7.3. Чат-боты 94

8. БУДУЩЕЕ ПРОФЕССИЙ И ПРОФЕССИИ БУДУЩЕГО 99

8.1. Профессии, которые не заменит ИИ 100

8.2. Новые профессии от ИИ 103

8.3. Там, где ИИ бессилен… 106

ИТОГОВЫЙ ТЕСТ ПО ТЕОРЕТИЧЕСКОЙ ЧАСТИ 111
ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ 117
«Релевантный информационный поиск» 117
ГРУППОВЫЕ КЕЙСЫ 120
Кейс 1. Интеллект-карта востребованных услуг 120
Кейс 2. Проектирование сервисных чат-ботов 121
КЛЮЧИ К ИТОГОВОМУ ТЕСТУ 124
ЛИТЕРАТУРА 125
ПРИЛОЖЕНИЕ 1 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 1 135
ПРИЛОЖЕНИЕ 2 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 2 137
ПРИЛОЖЕНИЕ 3 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 3 139
ПРИЛОЖЕНИЕ 4 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 4 141
ПРИЛОЖЕНИЕ 5 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 5 143
ПРИЛОЖЕНИЕ 6 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 6 145
ПРИЛОЖЕНИЕ 7 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 7 147
ПРИЛОЖЕНИЕ 8 КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 8 149

ВВЕДЕНИЕ

Искусственный интеллект (ИИ) давно и прочно вошел в нашу жизнь. Тем не менее этот раздел современной науки не сбавляет темпов развития в отношении появления всё новых интеллектуальных технологий и инструментов для их применения в различных сферах жизни и деятельности человека. Искусственный интеллект меняет окружающую нас действительность и меняется сам, превращая свежие разработки в классику вчерашнего дня, и, тем самым, вновь ускоряя собственное развитие.

В такой ситуации специалисты разных профессий, разработчики проектов, инженеры, студенты и исследователи с целью сохранения своей востребованности на рынке труда должны регулярно актуализировать собственные знания в области новых трендов ИИ. В рамках пособия представлена попытка решения этой задачи на текущий момент времени – на основе имеющегося опыта и анализа множества профильных информационных источников авторами подготовлены:

– восемь тем для изучения, каждая из которых сопровождается кратким конспектом содержания и множественными ссылками на примеры в Интернет;

– итоговый тест для самопроверки;

– индивидуальное задание;

– два кейса для выполнения в группах;

– ключи к итоговому тесту.

Предполагается, что в результате работы с данным учебным пособием студенты ознакомятся с основными понятиями, краткой историей и современными направлениями развития ИИ, а главное:

– классификацией технологий, а также программными и аппаратными приложениями ИИ;

– технологиями работы со знаниями;

– анализом инструментов ИИ для управления проектами;

– последними достижениями ИИ и рекомендациями к освоению вновь появляющихся компетенций.

Получат опыт и приобретут навыки:

– релевантного поиска информации;

– разработки интеллект-карт с учётом таймлайн трендов;

– разработки сервисных чат-ботов.

1. ОСНОВНЫЕ ПОНЯТИЯ И АЛГОРИТМЫ ПРИМЕНЕНИЯ

Официально считается, что рождение ИИ как научного направления произошло в 40-х годах XX века после создания ЭВМ. В 1956 году на семинаре «*Artificial intelligence*», проходившем в Дартсмутском колледже (США), был предложен сам термин «искусственный интеллект» – ИИ (*AI – artificial intelligence*). При этом в английском оригинале термин *intelligence* означает «умение рассуждать разумно» (а не «интеллект», для которого есть термин *intellect*).

После признания ИИ отдельной областью науки в нём сформировались два основных направления: нейрокибернетика и «кибернетика черного ящика». Эти направления развивались практически независимо, существенно различаясь как в методологии, так и в технологии. И только в настоящее время части ИИ стали объединяться в единое целое.

На практике ни одно решение ИИ невозможно реализовать без использования соответствующих данных и алгоритмов. В начале исторического развития ИИ больше внимания уделялось разработке новых алгоритмов. Сегодня же, когда объемы накопленных человеком данных стали превышать осязаемые границы, большую важность стали приобретать методы хранения, структурирования и обработки этих больших данных.

1.1. Основные понятия и история ИИ

Основную идею нейрокибернетики можно сформулировать следующим образом: «Единственный объект, способный мыслить, – это человеческий мозг. Поэтому любое «мыслящее» устройство должно каким-то образом воспроизводить его структуру». Отсюда основной ориентир нейрокибернетики – программно-аппаратное моделирование структур, подобных структуре мозга. Физиологами давно установлено, что основой человеческого мозга является большое количество (до 10^{21}) связанных между собой и взаимодействующих нервных клеток – нейронов. Поэтому усилия нейрокибернетики были сосредоточены на создании элементов, аналогичных нейронам, и их объединении в функционирующие системы. Эти системы принято называть нейронными сетями, или нейросетями.

Первые нейросети были созданы Розенблаттом и Мак-Каллокам в 1956–1965 годах. Это были попытки создать системы, моделирующие человеческий глаз и его взаимодействие с мозгом. Устройство, созданное ими тогда, получило название персетпрона (*perceptron*). Оно умело различать буквы алфавита, но было чувствительно к их написанию. Например, буквы *A*, A и ***A*** для этого устройства были тремя разными знаками. В 70–80 годах количество работ по этому направлению ИИ стало снижаться.

К концу 1980-х годов аппаратные ограничения по памяти и быстродействию компьютеров для моделирования нейросетей были практически сняты. В это время в Японии в рамках реализации проекта «ЭВМ V поколения» была создана транспьютерная технология – новый подход к аппаратной реализации нейросетей на базе параллельных компьютеров с большим количеством процессоров.

На сегодняшний день можно выделить три подхода к созданию нейросетей [1]:

1. Аппаратный – создание специальных компьютеров, нейрочипов, плат расширения, наборов микросхем, реализующих все необходимые алгоритмы.

2. Программный – создание программ и инструментариев, рассчитанных на высокопроизводительные компьютеры. Нейросети создаются в памяти компьютера, всю работу выполняют его собственные процессоры.

3. Гибридный – комбинация первых двух. Часть вычислений выполняют специальные платы расширения (сопроцессоры), часть – программные средства.

В основу подхода «черного ящика» положен принцип, противоположный нейрокибернетике: «Не имеет значения, как устроено «мыслящее» устройство. Главное, чтобы на заданные входные воздействия оно реагировало так же, как человеческий мозг». Сторонники этого направления мотивировали свой подход тем, что человек не должен слепо следовать природе в своих научных и технологических поисках. Это направление ИИ было ориентировано на поиски алгоритмов решения интеллектуальных задач на существующих моделях компьютеров.

В 1956–1973 годах велись интенсивные поиски моделей и алгоритмов человеческого мышления и разработка первых программ на их основе. Представители существующих гуманитарных наук – фи-

лософы, психологи, лингвисты – ни тогда, ни сейчас не в состоянии были предложить таких алгоритмов. Тогда кибернетики начали создавать собственные модели и разрабатывать собственные технологии. Так последовательно были созданы и опробованы следующие решения ИИ.

1. В конце 50-х годов родилась модель лабиринтного поиска. Этот подход представляет задачу, как некоторое пространство состояний в форме графа, и в этом графе проводится поиск оптимального пути от входных данных к результирующим. Была проделана большая работа по разработке этой модели, но для решения практических задач эта идея не нашла широкого применения.

2. Начало 60-х годов стало эпохой эвристического программирования. Эвристика – правило, теоретически (математически) не обоснованное, но позволяющее сократить количество переборов в пространстве поиска. Эвристическое программирование – разработка стратегии действий на основе известных, заранее заданных эвристик.

3. В 1963–1970 годы к решению интеллектуальных задач стали подключать методы математической логики. Был разработан метод резолюций, который позволяет автоматически доказывать теоремы при наличии набора исходных аксиом. На основе метода резолюций в 1973 году был создан язык логического программирования «ПРОЛОГ». Однако вскоре стало очевидным, что логические модели при всех своих преимуществах имеют существенные ограничения по классам решаемых задач.

В 1973 году в Великобритании по заказу Британского совета научных исследований известный математик Д. Лайтхилл, никак с ИИ профессионально несвязанный, подготовил обзор состояния дел в области ИИ. В докладе были признаны определенные достижения в области ИИ, однако уровень их определялся как разочаровывающий, и общая оценка была отрицательная с позиций практической значимости. Этот отчет отбросил европейских исследователей примерно на 5 лет назад, так как финансирование ИИ существенно сократилось.

Примерно в это же время существенный прорыв в развитии практических приложений ИИ произошел в США, когда к середине 1970-х годов на смену поискам универсального алгоритма мышления пришла идея моделировать конкретные знания специалистов-экспертов. В США появились первые коммерческие системы, осно-

ванные на знаниях, или экспертные системы (ЭС). Появилась новая технология решения задач ИИ – представление знаний. В это время были созданы две первые экспертные системы для медицины и химии – *MYCIN* и *DENDRAL*, которые впоследствии стали классическими, а также многие другие.

В конце 70-х годов в Японии был объявлен старт проекта вычислительных машин V поколения, основанных на знаниях. Проект был рассчитан на 10 лет и объединял лучших молодых специалистов (в возрасте до 35 лет) крупнейших японских компьютерных корпораций. Для этих специалистов был специально создан новый Институт компьютерных технологий нового поколения (*ICOT*), и они получили полную свободу действий, правда, без права публикации предварительных результатов. В результате они создали достаточно громоздкий и дорогой символьный процессор, программно реализующий язык подобный языку «ПРОЛОГ», но не получивший впоследствии широкого признания. Однако положительный эффект этого проекта был очевиден. В Японии появилась значительная группа высококвалифицированных специалистов в области ИИ, которая добилась существенных результатов в различных прикладных задачах. К середине 90-х японская ассоциация ИИ стала насчитывать 40 тыс. человек.

Начиная с середины 1980-х годов, повсеместно происходит коммерциализация ИИ. Растут ежегодные капиталовложения, создаются промышленные экспертные системы. Растет интерес к самообучающимся системам. Издаются десятки научных журналов, ежегодно собираются международные и национальные конференции по различным направлениям ИИ. Он становится одной из наиболее перспективных и престижных областей компьютерных наук (информатики).

1.2. Роль данных и знаний в ИИ

Практически каждый день каждый из нас пользуется двумя терминами: «данные» и «знания». При этом, на практике важно чётко различать границу между значениями этих слов, так как они относятся к категории фундаментальных понятий ИИ.

Данные – это отдельные факты, характеризующие объекты, процессы и явления предметной области, а также их свойства.

Знания же основываются на данных, полученных эмпирическим путем. Они представляют собой результат мыслительной деятельности человека (направленной на обобщение его опыта, полученного в ходе практической деятельности), позволяющий специалистам ставить и решать задачи в конкретной предметной области. В литературе часто используется следующее определение знаний.

Знания – это хорошо структурированные данные, или данные о данных, или – метаданные. Для хранения данных используются базы данных (для них характерны большой объем и относительно небольшая удельная стоимость информации), для хранения знаний – базы знаний (исключительно дорогие информационные массивы небольшого объема).

Знания могут быть классифицированы по двум категориям: поверхностные и глубинные.

Поверхностные – знания о видимых взаимосвязях между отдельными событиями и фактами в предметной области (пример: «Если нажать на кнопку звонка, раздаётся звук»).

Глубинные – абстракции, аналогии, схемы, отображающие структуру и природу процессов, протекающих в предметной области. Эти знания объясняют явления и могут использоваться для прогнозирования поведения объектов (пример: «Принципиальная электрическая схема звонка и проводки»).

Современные экспертные системы работают в основном с поверхностными знаниями. Это связано с тем, что на данный момент нет универсальных методик, позволяющих выявлять глубинные структуры знаний и работать с ними.

Кроме того, традиционно знания делятся на процедурные и декларативные. Исторически первичными были процедурные знания, то есть знания, «растворенные» в алгоритмах. Они управляют данными. Для изменения процедурных знаний требуется изменять алгоритмы (программы).

С развитием ИИ приоритеты знаний постепенно изменились – большая часть знаний стала сосредотачиваться в структурах данных (таблицы, списки, абстрактные типы данных). Таким образом увеличилась роль декларативных знаний (которые содержат информацию о свойствах и фактах в предметных областях). Эта тенденция объясняется следующим.

Первый этап развития ИИ был связан с разработкой методов решения задач – алгоритмов поиска решения и построения логического вывода. В настоящее время это уже достаточно проработанный и изученный вопрос.

Однако для нетривиальных случаев сильно возрастает трудоемкость поиска решения. Трудоемкость поиска решения – это быстрорастущая функция (в большинстве случаев растущая как экспонента или быстрее), аргументом которой является общий объем базы знаний. Одним из способов повышения эффективности поиска решения и уменьшения трудоемкости является изменение алгоритмов поиска. Однако на данный момент серьезного улучшения в этом направлении ожидать трудно. Гораздо более продуктивным методом, позволяющим уменьшить трудоемкость поиска решения, является представление знаний. Если выбор изощренного представления позволяет уменьшить объем базы знаний, то тот же стандартный алгоритм поиска становится значительно эффективнее в том смысле, что расширяются границы его применимости.

Таким образом, на сегодняшний день компактность базы – это определяющий фактор эффективности любой прикладной системы и, следовательно, границ применимости интеллектуальных технологий.

1.3. Вопросы применения алгоритмов и аппаратных средств

Как уже было отмечено ранее, данные являют собой решающий фактор ИИ. Однако независимо от количества имеющихся данных, чтобы сделать их полезными, необходимы алгоритмы. Кроме того, чтобы данные правильно работали с алгоритмами, следует выполнить предварительный анализ данных.

В любой интеллектуальной области невозможно создать универсальный алгоритм, подходящий для решения любой задачи. Метод решения – это всегда метод проб и ошибок, поиск новых подходов, новых путей, в конце концов – банальный перебор возможных решений. Но сегодня вы ещё не найдете машин, обладающих самодостаточностью и способных полностью избежать привлечения интеллектуальных усилий со стороны человека.

Алгоритм – это процедура, представляющая собой последовательность операций (обычно выполняемых компьютером) и гаран-

тирующая нахождение правильного решения задачи за конечный промежуток времени или уведомляющая об отсутствии решения. Даже при том, что люди использовали алгоритмы вручную на протяжении буквально тысяч лет, в зависимости от сложности решаемой задачи они могут потребовать огромных периодов времени и множества вычислений. Чем быстрее и проще алгоритмы находят решение, тем лучше. Алгоритмы являются продуктом интеллекта создавших их людей, поэтому любая работающая на основе алгоритмов машина не может не отражать человеческий интеллект, встроенный в такие алгоритмические процедуры.

Среди всего множества задач, решаемых с помощью алгоритмов, можно выделить «самые сложные» – *NP*-полные задачи (*non-deterministic polynomial* или *NP* – это недетерминированное полиномиальное время). *NP*-полные задачи отличаются от других тем, что поиск их решения за разумный период времени все еще невозможен (их нельзя решить перебором всех возможностей и их комбинаций). Даже если бы были компьютеры, много более мощные, чем современные, поиск решения затянулся бы почти навсегда.

Алгоритмы ИИ отличаются от обобщенных алгоритмов решения задач тем, что они имеют дело со сложными проблемами, зачастую являющимися частями *NP*-полных задач. Так, например, в шашках, относительно простой игре, возможно 500 квинтиллионов (пять с двадцатью нулями) позиций на доске. Это значение соответствует количеству всех песчинок на Земле. Конечно, во время игры в шашки делается куда меньше ходов, но все же количество потенциальных вариантов при каждом ходе слишком велико для вычислений. Чтобы вычислить все 500 квинтиллионов возможных ходов, современному мощному компьютеру потребуется около 18 лет.

Большинство обобщенных алгоритмов связаны с ИИ, поскольку все они – решения повторяющихся, жестко разграниченных, сложных задач. Они требуют, чтобы архитектор изучил задачу и выбрал правильный алгоритм для её решения. Однако смена задачи или её коррекция могут стать реальной проблемой для успешного применения существующего алгоритма. Это связано с тем, что изучение задачи и ее решение осуществляются раз и навсегда, пока алгоритм используется в программном обеспечении. Например, вы можете создать программу с ИИ для решения Судоку (популярная

головоломка с числами) и даже обеспечить алгоритму возможность учитывать изменяющийся размер игрового поля и дополнительные правила. Но как бы не был хорош ваш алгоритм, он всё равно будет работать с головоломками, правила составления и разгадывания которых чётко формализованы. И здесь важно отметить, что проблемы реального мира никогда не возникают в таких простых мирах совершенной информации и четких действий.

Рассмотрим, к примеру, а) задачу поиска страхового мошенника или б) задачу диагностики заболеваний. В целом, каждую из них можно охарактеризовать следующим образом.

1. Большой набор правил и возможностей: а) количество возможных обманов невероятно велико; б) у многих болезней схожие симптомы.

2. Недостаток информации: а) мошенники могут скрывать информацию; б) врачи зачастую полагаются на неполную информацию (анализы могут еще отсутствовать).

3. Правила могут изменяться: в) мошенники изобретают все новые и новые способы надувательства; б) постоянно возникают и обнаруживаются новые болезни.

Для решения таких задач нельзя использовать предопределенный (заранее формализованный, как с программой для Судоку) подход. Чтобы справляться с любыми новыми ситуациями, подход должен быть гибким, необходимо накопление полезных знаний. Другими словами, для адаптации алгоритма к изменениям в сложной окружающей обстановке ему необходимо продолжать учиться, как постоянно учатся люди.

Решения, способные к самостоятельному обучению непосредственно на данных без их предварительного представления в качестве символов, возникли не так уж давно. Одни из них были по природе статистическими, другие по-разному подражали природе, а третьи пытались создать автономную символическую логику в форме правил для необработанной информации. Сегодня эти, выработанные различными школами решения, принято относить к машинному обучению (*machine learning*), алгоритмы которого отличаются от своих собратьев тем, что они не являются наборами предопределенных этапов, предназначенных для решения задачи. Как правило, машинное обучение имеет дело с задачами, которые люди не умеют подразделять на этапы, но сами их, естественно,

решают. Примером таких задач является распознавание лиц на изображениях или слов в разговорной речи.

Далее параграфа обратимся к характеристикам аппаратных средств, на которых выполняются алгоритмы (программы) ИИ.

Отметим, что одной из причин первых неудач при создании ИИ была нехватка подходящих аппаратных средств. Существующие тогда аппаратные средства не могли выполнять задачи достаточно быстро, чтобы быть полезными даже для обыденных потребностей, а о моделировании процесса мышления человека не могло быть и речи.

Действительно, чтобы начать моделировать процесс мышления человека, нужны не просто производительные, но и специализированные аппаратные средства. Но даже лучшие из них сегодня с такой задачей не справляются. Почти все стандартные аппаратные средства (компьютеры) обладают архитектурой фон Неймана [2], согласно которой память отделена от вычислений. Это позволяет создать среду исполнения, в которой работают хорошо очень многие виды алгоритмов. Но для такой архитектуры характерна одна особенность – низкая скорость передачи данных между процессором и памятью, что создает так называемое «узкое место фон Неймана» (*Von Neumann bottleneck*).

Даже с аппаратными средствами, специально разработанными для ускорения вычислений, машина, предназначенная для моделирования процесса мышления человека, способна работать только с такой скоростью, с какой позволяют ее механизмы ввода и вывода. Следовательно, необходимо создать лучшую, более эффективную и быструю среду. Это можно сделать множеством способов, но современные усилия в основном сосредоточены на двух из них: улучшение возможностей используемого оборудования и применение специализированных сенсоров. Такие улучшения хороши, но их всё еще недостаточно для моделирования работы человеческого мозга.

В большинстве проектов ИИ предполагается, что их создание по крайней мере начнется со стандартных аппаратных средств, поскольку современные общедоступные компоненты фактически обладают весьма высокой мощностью обработки, особенно по сравнению с компонентами 1980-х годов, когда ИИ начал давать результаты, пригодные для практического применения. Следовательно, даже если стандартных аппаратных средств не хватит для создания полностью работоспособной системы, вы можете работать на них со своим экс-

периментальным кодом и формировать рабочую модель, которая в конечном счете заработает с полным набором данных.

После создания прототипа, выполняющего задачи, необходимые для моделирования процесса мышления человека в заданной области, как правило, требуются дополнительные аппаратные средства. Эти средства, обладающие достаточной вычислительной мощностью для работы с полным набором данных, необходимы для рабочей системы. Для обеспечения требуемой вычислительной мощности доступно множество путей, но сегодня в дополнение к центральному процессору машины в основном используют графические процессоры.

С появлением графического процессора (*GPU, Graphics Processing Unit*) обработка графики переместилась с системной платы на плату периферийного графического устройства. Центральный процессор может указать процессору *GPU* выполнить некую задачу, а *GPU* сам определяет наилучший метод для этого и делает всё независимо от центрального процессора. У *GPU* есть отдельная память и собственный «очень широкий» путь получения данных из шины. Кроме того, процессор *GPU* способен напрямую обращаться к оперативной памяти, чтобы получать необходимые для выполнения задачи данные и переносить туда результаты независимо от центрального процессора.

Современные графические процессоры отличает то что они обычно содержат сотни вычислительных ядер (в отличие от лишь нескольких ядер центрального процессора). Центральный процессор обладает куда более универсальными функциональными возможностями, чем *GPU*, но *GPU* выполняет вычисления невероятно быстро, а их результат на дисплей передает еще быстрее. Эта способность делает специальные процессоры *GPU* критически важным компонентом современных вычислительных систем. Их вычислительные ядра, работающие параллельно, способны эффективно ускорять вычисления для решения задач ИИ.

В проекте *Google Brain* 2011 года ИИ обучали распознавать котов и людей при просмотре роликов на *YouTube* [3]. Для решения этой задачи *Google* задействовал две тысячи процессоров в одном из своих гигантских центров данных. Немногим людям удастся разжиться подобным количеством ресурсов, чтобы повторить эксперимент *Google*. Однако очень скоро Брайану Катанзаро удалось

повторить эксперимент *Google*, используя набор из двенадцати плат *GPU NVidia*. После того как стало понятно, что платы *GPU* способны заменить множество компьютерных систем, снабженных процессорами с архитектурой фон Неймана, появилась возможность запуска новых сложных проектов ИИ.

Процессоры на базе архитектуры фон Неймана все еще хороши для бизнес-систем и приложений, требующих общей гибкости, когда факторы программирования перевешивают чистую вычислительную мощь. Однако сейчас *GPU* стали стандартом в науке о данных, машинном обучении, ИИ и глубоком обучении. Конечно, изыскания и разработки в этом направлении продолжаются. Центральный процессор и процессор *GPU* – это процессоры рабочего уровня. В будущем вместо этих двух видов процессоров может использоваться нечто иное.

Так, например, интегральные схемы специального назначения (*ASIC, Application-Specific Integrated Circuit*) в отличие от процессоров общего назначения, создаются для вполне конкретной цели. Решения на базе *ASIC* обеспечивают чрезвычайно высокую производительность, потребляя очень мало мощности, однако гибкость у них отсутствует. Примером решения *ASIC* является тензорный процессор *Google* (*TPU*), который используется только для обработки речи [4].

Ещё одни пример – это использование программируемых логических интегральных схем (*FPGA, Field-Programmable Gate Array*). Подобно *ASIC*, производители обычно создают *FPGA* для вполне определенной цели. Однако в отличие от *ASIC*, матрицу *FPGA* можно запрограммировать так, чтобы изменить ее основные функциональные возможности. Примером решения *FPGA* является *Brainwave* от *Microsoft* для проектов глубокого обучения [5].

Важнейший компонент ИИ – это его способность моделировать человеческий интеллект, используя полный набор чувств. Человеческие органы чувств обеспечивают правильный способ восприятия для работы интеллекта человека. Даже если предположить, что ИИ станет на это способен, для полной его реализации потребуется соответствующий интерфейс для ввода данных, который сделает этот интеллект функциональным.

С одной стороны, в ряде случаев люди хотят, чтобы органы чувств у ИИ превосходили или отличались от человеческих. С другой

стороны, возможность работать в окружающей среде, в которой люди работать не могут, является одной из причин огромной популярности роботов некоторых типов, но работа в этих средах зачастую требует набора чувств (сенсоров), отличных от человеческих. Следовательно, тема сенсоров фактически относится к двум категориям (ни одна из которых не определена полностью): сенсоры, подобные человеческим, и альтернативные сенсоры окружающей обстановки.

Взаимодействие с ИИ все чаще осуществляется способами, более приемлемыми для людей, чем непосредственный контакт с компьютером. Например, когда вы задаете *Alexa* (голосовой помощник от *Amazon*) вопрос, ввод осуществляется через набор микрофонов. ИИ превращает ключевые слова вопроса в лексемы, которые она может понять. Затем эти лексемы инициализируют вычисления, в результате которых формируется вывод. ИИ преобразует вывод в форму, понятную человеку: разговорную речь. Затем вы слышите сентенцию, проговариваемую *Alexa* через динамик. Короче говоря, для обеспечения полезных функциональных возможностей *Alexa* должна взаимодействовать с окружающей средой двумя разными способами, присущими людям, но фактически непонятными для самой *Alexa*.

Взаимодействия могут принимать множество форм. Например, ИИ сегодня уже может различать запахи. Нас уже почти не удивляют роботы, способные управлять автомобилем (область применения физического взаимодействия). ИИ становится все более и более способным выполнять сложные вычисления с большими наборами данных, а его возможности решать сложные задачи постоянно растет.

2. ПРИЛОЖЕНИЯ И ТЕХНОЛОГИИ В ОБЩЕСТВЕ

Очевидно, большинство из нас уже давно пользуются ИИ, порой даже не осознавая этого. Так, например, при разговоре со смартфоном ИИ занимается распознаванием речи, он предупреждает о входящих спам-звонках на смартфон, фильтрует электронные письма, поступающие в почтовый ящик, а также решает множество других задач. В первой части этой главы обсуждается ряд аспектов применения ИИ в компьютерных приложениях.

Во второй части главы приводится классификация технологий, которые в настоящее время вызывают наибольший интерес у практиков и исследователей, и рассматриваются характеристики их текущего состояния.

В заключительной части главы представлен обзор некоторых достижений ИИ на уровне компьютерных приложений в области техники, медицины и человеческого общения.

2.1. ИИ в компьютерных приложениях

Компьютерные приложения определяют конкретные виды применения ИИ. Достаточно часто сегодня ИИ применяется в приложениях из следующих областей: искусственная жизнь, автоматизация, бионика, интеллектуальный анализ данных, фильтрация спама, интеллектуальное управление, представление знаний, инженерная робототехника, структурирование и представление знаний и многие другие.

В самой простой форме ИИ может помогать при пользовательском вводе. Например, когда пользователь вводит несколько букв некого слова, ИИ предлагает остальные буквы. Оказывая эту услугу, ИИ достигает нескольких целей:

– пользователь работает эффективнее, вводя меньше знаков;

– приложение получает меньше записей с ошибками из-за опечаток;

– пользователь и приложение, оба, участвуют в высокоуровневой коммуникации; приложение узнает у пользователя правильные или улучшенные термины, которых пользователь мог бы и не помнить, а также позволяет избегать альтернативных терминов, которые компьютер не может распознать.

Искусственный интеллект может также обучаться на предыдущем пользовательском вводе, видоизменяя рекомендации таким способом, который лучше соответствует подходам пользователя к решению задач. Рекомендации могут также включать подсказку пользователю идей, которые в противном случае пользователь, возможно, и не рассматривал бы.

Даже в области рекомендаций у людей может сложиться мнение, что ИИ способен думать, но это не так. ИИ просто выполняет расширенный поиск по шаблону и анализ вероятности уместности термина в текущем вводе.

Искусственный интеллект позволяет людям осуществлять и другие виды интеллектуального ввода. Распознавание голоса – одна из главных общепринятых методик интеллектуального ввода. В качестве визуальных способов можно упомянуть распознавание лица владельца электронного устройства.

Однако ввод данных может также подразумевать их мониторинг. Вполне можно отслеживать показатели состояния организма пользователя для выявления потенциальных проблем. Фактически ИИ может использовать огромное количество методов интеллектуального ввода данных, большинство из которых еще даже не изобретено.

По мере роста интеллекта ИИ становится немаловажным фактором действия дружественного ИИ (ДИИ), совместимого с общим искусственным интеллектом (ОИИ), оказывающим положительное влияние на человечество. При этом важно отметить, что задачи ОИИ могут не иметь ничего общего с человеческой этикой. Это сегодня вызывает тревогу. Однако интеллект ДИИ обладает логикой, гарантирующей соответствие задач ИИ человеческим. Последнее определяется на уровне трёх законов робототехники А. Азимова [6]. Однако многие утверждают, что эти три закона – хорошая отправная точка и что необходимы дополнительные гарантии [7].

Несмотря на перечисленное ранее разнообразие областей применения ИИ, он чаще всего используется на уровне исправлений и рекомендаций. Исправление не предполагает обязательного предоставления ответа на ошибку, а рекомендация – обязательного ответа на вопрос.

Исправления могут принимать разные формы и необязательно означать, что ошибка произошла или произойдет в будущем.

Например, автомобиль может помогать водителю, регулярно корректируя маршрут. Водитель вполне может и сам хорошо вести машину, но ИИ способен вносить микро-коррекции, чтобы помочь водителю оставаться в границах безопасности.

Рекомендация – это идея, выдвинутая как возможное решение задачи. Рекомендация подразумевает возможность и других решений, а принятие рекомендации не означает её неизбежной реализации (она может даже не сработать). Наиболее распространенный способ использования ИИ для создания рекомендаций подразумевает сбор данных о действиях при предыдущих событиях и выработку новых рекомендаций на их основании.

Важно всегда помнить, что ИИ способен помочь людям избегать наиболее распространенных или повторяющихся ошибок. В некоторых случаях ошибки ИИ способны даже вносить немного юмора в повседневную жизнь. Однако ИИ не думает, и он не может сегодня заменить людей во многих ситуациях. Он работает лучше, когда решение контролирует человек или когда окружающая обстановка достаточно статична, чтобы предсказуемость хороших результатов была высока.

2.2. Классификация технологий ИИ

Приложения ИИ применимы в любой сфере почти для любых целей. Гибкость предложений ИИ означает то, что некоторые из них могут использоваться в тех сферах, для которых они первоначально даже не предназначались. Из множества технологий искусственного интеллекта можно выделить несколько технологий, которые в настоящее время вызывают наибольший интерес у практиков и исследователей. Среди них следующие:

– понимание естественного языка;
– машинный перевод;
– интеллектуальные базы данных;
– экспертные системы;
– автоматическое доказательство теорем;
– планирование действий робота;
– распознавание образов;
– интеллектуальные игры.

Понимание естественного языка

Существует два принципиально разных способа общения человека с компьютером. Первый из них, когда человек способен понимать компьютер. Это не заслуга компьютера, а способность человека, которого все окружающие называют программистом. Второй, когда компьютер способен понимать человека. Такой компьютер обязательно должен обладать элементами ИИ. И такие программы были созданы уже сравнительно давно.

Одной из первых прикладных систем, построенных на основе интеллектуальных технологий, является программа *ELIZA*, созданная Дж. Вейценбаумом в 1966 году и понимающая естественный язык. Позднее Терри Виноград написал книгу «Программа, понимающая естественный язык», описывающую *ELIZA*. Структура *ELIZA* была чрезвычайно проста и включала в себя всего несколько десятков эвристических правил, предназначенных для распознавания грамматических конструкций. В результате эта программа могла вести вполне правдоподобный диалог, при этом, зачастую, отвечая вопросом на вопрос. Пример диалога с *ELIZA*:

– *ELIZA, ты компьютерная программа?*
– *Почему Вы думаете, что я компьютерная программа?*
– *Потому что тебя составил Вейнценбаум.*
– *Расскажите мне о Вейнценбауме.*

Несмотря на всю свою простоту, эта программа до сих пор находит применение в психиатрии. Разговор психиатра с невротиком продолжается не более 20 минут, после чего врачу просто необходим отдых. Программа ELIZA способна общаться с больным часами, при этом наблюдаются явные улучшения в психическом состоянии невротика.

Машинный перевод

Важный класс примеров, связанных с пониманием естественного языка, образуют системы автоматического перевода текстов с одного естественного языка на другой. Уровень автоматического перевода сегодня существенно продвинули искусственные нейронные сети. Так, например, весь объём современной Википедии (статического текста) может быть переведен нейросетевыми средствами облака *Microsoft* с одного языка на другой менее чем за 20 миллисекунд.

Принято выделять два основных варианта организации межязыкового перевода: статический и динамический. В текущий Топ-5 популярных ресурсов статического перевода входят:

1) *Google* Переводчик (поддерживает 109 языков);
2) Яндекс. Переводчик (поддерживает 97 языков);
3) *Bing Microsoft Translator* (поддерживает 63 языка);
4) *Systran* (поддерживает 41 язык);
5) *PROMT* (поддерживает 20 языков).

В области динамического перевода лидером является голосовой переводчик *Skype Translator*. На данный момент он может переводить беседы с 60 на 11 языков, включая английский, испанский, французский, немецкий, китайский, итальянский, португальский, арабский и русский.

Интеллектуальные базы данных

Интеллектуальные базы данных предоставляют эффективные способы хранения, представления и извлечения огромного числа фактов. Более того, такие базы позволяют отвечать на вопросы, требующие дедуктивного рассуждения. Идея состоит в следующем. Допустим, что кроме обычных реляционных таблиц (см. табл. 2.1) в базе данных хранятся и некоторые правила. Например,

If *x.Должность = «Зав.*» & x.Работа = y.Работа*
Then *x начальник y*

Таблица 2.1

Пример содержимого таблицы базы данных

Фамилия	Должность	Подразделение
Иванов	инженер	Отдел автоматизации производства
Петров	заведующий	Отдел автоматизации производства
Сидоров	инженер	Отдел автоматизации производства

Приведенный пример позволяет выяснить отношения на уровне начальник-подчиненный. В данном случае, очевидно, что Петров является начальником не только у Иванова, но и у любого другого сотрудника отдела автоматизации производства.

Экспертные системы

Экспертная система (ЭС) – это сложный программный комплекс, аккумулирующий знания специалистов в конкретной предметной области и тиражирующий этот эмпирический опыт для консультаций менее квалифицированных пользователей.

Реальные ЭС могут иметь сложную, разветвленную структуру модулей, но для любой ЭС необходимо наличие следующих основных блоков (рис. 2.1).

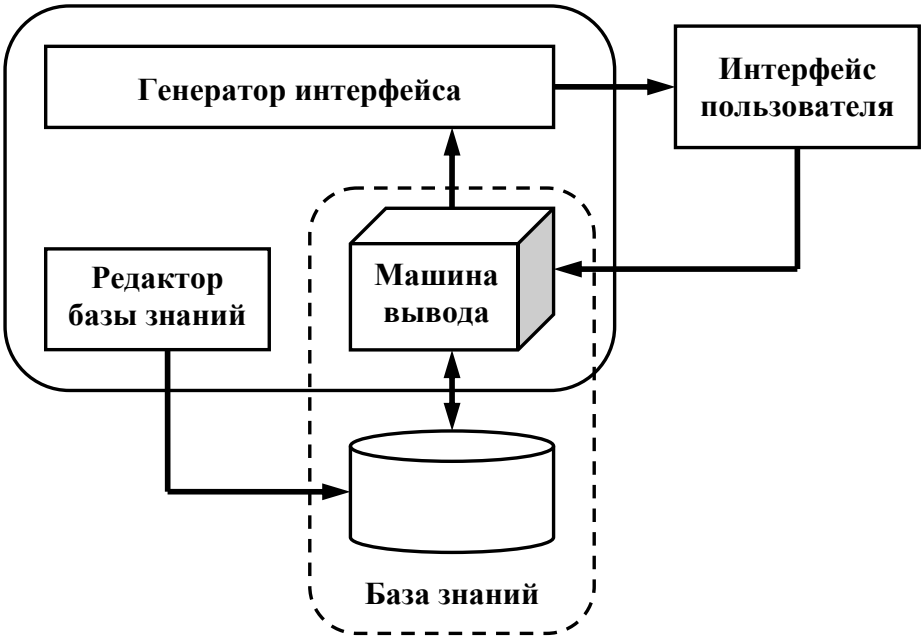


Рис. 2.1. Обобщенная структура ЭС

База знаний – наиболее ценный компонент ядра ЭС, совокупность знаний о предметной области и способах решения задач, записанная в форме, понятной неспециалистам в программировании: эксперту, пользователю и др. Обычно знания в БЗ записываются в форме, приближенной к естественному языку. Форма записи знаний получила название язык представления знаний (ЯПЗ). В различных системах могут использоваться различные ЯПЗ. Параллельно такому «человеческому» представлению БЗ может существовать во внутреннем «машинном» представлении. Преобразование между различными формами представления БЗ должно осуществляться автоматически, так как редактирование БЗ не подразумевает участие программиста-разработчика.

Машина вывода – блок, моделирующий ход рассуждений эксперта на основании знаний, заложенных в БЗ. Машина вывода яв-

ляется неизменной частью ЭС. Однако большинство реальных ЭС имеют встроенные средства управлением ходом логического вывода с помощью так называемых метаправил, записываемых в БЗ.

Редактор базы знаний – предназначен для разработчиков ЭС. С помощью этого редактора в БЗ добавляются новые знания или редактируются существующие.

Интерфейс пользователя – блок, предназначенный для взаимодействия ЭС с пользователем, через который система запрашивает необходимые для ее работы данные, и выводит результат. Система может иметь «жесткий» интерфейс, ориентированный на определенный способ ввода и вывода информации, или может включать средства проектирования специализированных интерфейсов для более эффективного взаимодействия с пользователем.

С точки зрения изучения интеллектуальных технологий экспертных систем наибольший интерес представляют база знаний и машина вывода, различные аспекты реализации которых будут рассмотрены ниже.

В процессе функционирования ЭС считывает информацию из своей базы знаний и пытается осуществить логический вывод решения поставленной перед ней задачи. В базе знаний могут храниться два основных вида записей: факты, описывающие состояние предметной области, составляющие её объекты и их свойства, а также правила, описывающие способы решения задачи. Все правила БЗ имеют одинаковую форму записи и состоят из двух частей: условие и действие.

Предварительным этапом работы ЭС является сбор исходных фактов, описывающих проблему на языке представления знаний. Эти факты могут поступать в систему различными способами: в режиме диалога через интерфейс пользователя, посредством файлов или баз данных, от внешних датчиков или приборов. После считывания исходной информации машина вывода начинает просмотр базы знаний и последовательно сопоставляет описание задачи с записями БЗ, описывающими ход решения. Если условие текущего правила БЗ подтверждается множеством исходных фактов, то система выполняет действие, записанное в данном правиле, добавляя в БЗ новые, производные факты.

На первый взгляд процесс вывода кажется достаточно простым – выполняются однотипные операции по перебору записей БЗ

и сравнении их с имеющимися фактами, пока не будет найдено решение или некий целевой факт. Однако управление процессом вывода, независящее от контекста проблемы на практике мало эффективно. При решении реальных задач человек крайне редко прибегает к перебору данных. Вместо этого, люди пользуются эвристическими правилами, которые значительно ограничивают пространство поиска решения и позволяет быстро и эффективно решать задачи.

Следует отметить, что промышленные прикладные ЭС могут быть существенно сложнее и дополнительно включать базы данных, интерфейсы обмена данными с различными пакетами прикладных программ, электронными библиотеками и т.д.

Автоматическое доказательство теорем

Доказательство теорем, безусловно, предполагает затрату интеллектуальных усилий, требующий не только дедуктивных навыков, но и в значительной степени интуитивных предпосылок (например, какие леммы необходимы для той или иной теоремы). Другими словами, доказательство теорем подразумевает, в том числе, поиск уже доказанных теорем и решенных подзадач, которые могут оказаться полезными для достижения конечной цели. К настоящему времени разработано несколько программ с элементами ИИ, то есть программ автоматического доказательства теорем, способных работать в этом направлении.

Из курса математической логики известны три обстоятельства:

1) любое утверждение может быть сформулировано как формула прикладного исчисления предикатов;

2) все правильно построенные формулы могут быть перечислены;

3) вывод доказуемой формулы может быть найден.

Перечисленные обстоятельства позволяют сделать очевидный вывод о том, что автоматическое доказательство теорем возможно, и его нетрудно запрограммировать. Однако такой прямолинейный подход оказывается безнадежно неэффективным. Сложные, интересные теоремы автоматически доказать не удаётся, потому что пространство перебора столь велико, что даже самые быстродействующие компьютеры не в состоянии ничего сделать за разумное время.

Однако существуют примеры, когда в очень узком классе математических теорий, применяя очень специфические методы, уда-

валось автоматически находить и доказывать нетривиальные содержательные теоремы, которые были ранее неизвестны.

Автоматическое управление роботом

Идея создания роботов далеко не нова. Само слово «робот» появилось в 20-х годах, как производное от чешского «робота» – тяжелой грязной работы. Его автор – чешский писатель Карел Чапек, описавший роботов в своем рассказе «Р.У.Р».

Роботы – это электротехнические устройства, предназначенные для автоматизации человеческого труда. Управление роботом – задача чрезвычайно сложная. Казалось бы, нет ничего проще, чем передвигаться на двух ногах. Развитие любого человеческого существа начинается со способности видеть и ходить – тех навыков, которые вроде бы и не требуют значительных интеллектуальных усилий.

На первый взгляд, может показаться, что говорить, рассуждать и программировать значительно сложнее. Однако, если взглянуть на «развитие» роботов и робототехники, можно легко убедиться в обратном. Роботы сначала «научились» программировать и рассуждать, а только потом говорить, и ходить.

Успешное хождение двуного робота – достаточно сложная задача даже для современного развития науки и техники. Дело в том, что, когда мы ходим, центр тяжести нашего тела почти никогда не находится над площадью опоры. С точки зрения механики, мы не ходим, а непрерывно падаем, но не совсем, а удачно подставляем ногу в нужное место и продолжаем движение. Лобовой подход – составить кинематическую модель с нужным количеством степеней свободы и на каждом шаге быстро решать соответствующую систему дифференциальных уравнений не работает. Утрируя, можно сказать, что обезьяна превратилась в человека не потому, что стала ходить на двух ногах, а скорее человек стал достаточно умен, чтобы на двух ногах начать передвигаться.

В отношении человеческого зрения известно, что на третьей неделе развития ребенок уже обладает цветным стереоскопическим зрением. Важно подчеркнуть, что эта способность объясняется не физиологическими, а интеллектуальными причинами. В этом нетрудно убедиться: прикройте один глаз – способность видеть не плоский, а трехмерный мир и определять расстояния до предметов сохранится. С другой стороны, биологи установили, что если поместить человека в совершенно непривычную искусственную среду

(в которой ненормально устроено освещение, привычные предметы имеют непривычные размеры, процессы имеют ненормальный темп и т.д.), то способность к адекватному восприятию трехмерной действительности человеком утрачивается.

Таким образом, цветное стереоскопическое зрение – это способность мозга, а не глаза. Мы видим не то, что есть на самом деле, а то, что ожидаем увидеть, ту модель, которую очень быстро, за десятые доли секунды, строит мозг по информации, полученной из глаза. Решение задачи технического зрения заключается не в разрешающей способности камер и не в их количестве, а в сложности и качестве алгоритмов, адекватно восстанавливающих трехмерную модель по совокупности плоских изображений, полученных от камер.

Несмотря на все перечисленные сложности современная робототехника сегодня существенно продвинулась вперёд. Достаточно посмотреть, как сегодня крутят сальто [8] и танцуют двуногие роботы от компании *Boston Dinamics* [9].

Распознавание образов

Распознавание образов – классическая, едва ли не самая первая область применения методов ИИ. Можно упомянуть такие исследования, как персептрон Ф. Розенблатта – одна из первых искусственных сетей, способных к перцепции (восприятию) и формированию реакции на воспринятый стимул.

Персептрон рассматривался его автором не как конкретное техническое вычислительное устройство, а как модель работы мозга. Нужно заметить, что после нескольких десятилетий исследований современные работы по искусственным нейронным сетям редко преследуют такую цель.

Один из первых серьезных успехов в распознавании образов (или, другими словами, прообраза машинного зрения) связан с созданием в 1982 году фирмой Logica системы распознавания отпечатков пальцев, которая находит широкое применение не только в криминалистике, но и в разнообразных системах защиты, например, в смартфонах.

Другими примерами применения интеллектуальной технологии распознавания образов являются прикладные системы распознавания речи, распознавания сканированного текста (OCR), мониторинга среды по аэрофотоснимкам, лиц, автомобилей и т.п. При этом необходимо хорошо понимать, что синтез объекта всегда значи-

тельно проще, чем анализ того же самого объекта. Наглядный пример – рукописный текст. Его очень легко написать, но прочитать бывает иногда неимоверно трудно.

Интеллектуальные игры

Одно из самых известных и популярных применений интеллектуальных технологий – это разнообразные интеллектуальные игры. Почти наверняка каждый сталкивался с искусственным интеллектом, трудящимся именно в этой области. Шашки, шахматы, нарды работают в каждом смартфоне.

Достаточно сказать, что современные компьютерные шахматные программы играют на более высоком уровне, чем ведущие гроссмейстеры мира. При этом компьютерные шахматы – это не только развлечение, но и способ обучения новых шахматистов, создание шахматных баз данных, анализ окончаний, изменение правил. Но самое главное, в любом случае, это один из наиболее интересных способов представления знаний.

Любопытный пример. Ранее в шахматах действовало правило, что партия заканчивается вничью, если за пятьдесят ходов не было взятий или превращений. В результате компьютерного анализа было найдено результативное окончание, которое требовало 53 хода. В результате шахматные правила были изменены.

В интеллектуальных играх очень сложной является задача представления знаний. Дело в том, что некоторые интеллектуальные игры чрезвычайно сложны с количественной точки зрения. Например, шашки имеют порядка 10^{40} позиций, а шахматы – 10^{120} позиций. Это огромные числа, большие, чем число атомов во Вселенной и чем число наносекунд, оставшихся до конца света.

Другие интеллектуальные технологии

Искусственный интеллект – междисциплинарная наука, которая, как мощная река по дороге к морю, вбирает в себя ручейки и речки смежных наук. Помимо перечисленных выше интеллектуальных технологий существует множество зародившихся совсем недавно и сегодня активно развивающихся. Стоит лишь взглянуть на основные рубрикаторы конференций по ИИ, чтобы понять, насколько широко простирается область применения современных технологий ИИ:

- эволюционные алгоритмы;
- когнитивное моделирование;

- интеллектуальные интерфейсы;
- мультиагентные системы;
- менеджмент знаний;
- мягкие вычисления и многое другое.

Конечно, невозможно в рамках одного пособия рассмотреть все многообразие подходов и идей в области ИИ. Однако некоторые современные направления будут подробнее рассмотрены в последующих разделах.

2.3. ИИ в технике, медицине и общении

Одной из самых сложных задач ИИ является поддержка всего человеческого тела. Это случай, когда некто носит экзоскелет (по существу, разновидность робота). Искусственный интеллект считывает движения человека (или его намерение переместиться) и приводит экзоскелет в действие. Сегодня в использовании экзоскелетов продвинулись, прежде всего, военные. Однако промышленность тоже интересуется использованием этой технологии [10].

Самое интересное, что экзоскелеты позволяют людям делать не только то, на что они способны и сами, но и на что они не были раньше способны. Например, в одной из статей [11] рассказывается об использовании экзоскелета, позволившего ходить ребенку с церебральным параличом. Ещё один пример медицинского применения данной технологии – экзоскелет, позволяющий пациенту после инсульта снова нормально ходить [12]. Важно, что по мере восстановления экзоскелет поддерживает человека все меньше и меньше и в конце концов становится ненужным.

Когда-то потеря, например, конечности означала годы визитов к врачу, а также более короткую и менее счастливую жизнь. Однако хорошие эндопротезы и другие подобные устройства с поддержкой ИИ, сделали этот сценарий для многих людей ушедшей историей. Один из удивительных примеров эндопротеза с поддержкой ИИ – это полностью динамическая стопа, созданная Хью Херром [13]. Эта стопа и лодыжка работают настолько хорошо, что фактически позволили Хью заниматься скалолазанием.

Анализ данных – это одна из областей, в которых ИИ не превзойден. При многих болезнях время критически важно. Сегодня ИИ позволяет не только поставить диагноз и идентифицировать

опухоль на ранних этапах, но и сделать это с высокой точностью и очень быстро.

Роботы и ИИ сегодня уже участвуют в хирургических процедурах. Фактически некоторые хирургические операции были бы практически невозможны без использования роботов и ИИ. Однако история применения этой технологии еще не велика. Первый хирургический робот, *Arthrobot*, появился в 1983 году [14]. Даже в то время применение этих спасительных технологий снижало количество ошибок, улучшало результаты, ускоряло заживание и делало хирургию относительно проще.

Процесс помощи лечащему врачу со стороны современного ИИ начинается с помощи в мониторинге пациентов такими способами, которые для человеческого персонала просто невозможны. Мониторы относятся к категории полезных устройств. Большинство из них имеют опосредованное отношения к медицине, но все же дают положительный эффект для здоровья. Рассмотрим несколько примеров индивидуальных мониторов.

1. Монитор *Moov*, контролирующий пульс и движение в пространстве [15]. ИИ данного браслета отслеживает статистические данные и предоставляет совет о том, как лучше делать разминку. Человек получает советы о том, как лучше ставить ногу во время бега и стоит ли увеличивать шаг. Задача устройства – гарантировать наиболее эффективную разминку, которая улучшит здоровье, без риска получить травму.

2. Монитор *K'Watch* обеспечивает постоянный мониторинг суточных повышений и падений уровня глюкозы [16].

3. Монитор *Wearable Defibrillator Vest* непрерывно контролируют состояние сердца и при необходимости выполняет функции дефибриллятора, обеспечивая разряд для поддержания правильной работы сердца [17]. Этот монитор помогает в краткосрочном режиме принять решение врачу, нужна ли вживляемая версия устройства.

4. Монитор *QardioCore* снимает кардиограмму без проводов, и любой человек с минимальными медицинскими знаниями легко может его использовать [18].

Все рассмотренные устройства взаимодействуют со смартфоном, что позволяет передавать данные от них для обработки другим приложениям или пересылать своему врачу. Основные задачи при их создании – миниатюризация размеров и упрощение использования.

Когда люди обычно думают о коммуникации, они подразумевают литературу или разговор. Однако общение может иметь и множество других форм, включая визуальный контакт, интонацию голоса и даже запах. Примером компьютерной версии улучшения человеческих возможностей является электронный нос, использующий для решения своей задачи комбинацию электроники, биохимии и искусственного интеллекта, который нашел широкое применение в промышленности и научных исследованиях [19].

В заключение хочется обратить внимание на два новых алфавита, которые появились в век компьютеров: эмотиконы и эмодзи. На сайтах с изображениями этих двух графических алфавитов их содержатся сотни. По большей части люди легко интерпретируют эти графические алфавиты, поскольку они напоминают выражения лица, но у компьютерных приложений нет человеческого восприятия эмоций, поэтому искусственный интеллект им зачастую требуется только для того, чтобы выяснить, какую эмоцию человек пытается передать этим небольшим изображением. К счастью, для них есть стандартизированные списки, такие как таблица эмодзи Unicode [20].

Считается, что эмотикон сегодня – это устаревающая технология, а эмодзи – новая и весьма захватывающая. Последняя даже удостоена мультфильма [21]. Сегодня *Google* на основе ИИ позволяет превращать в эмодзи даже собственное селфи [22].

3. ПРОГРАММНЫЕ ПРИЛОЖЕНИЯ

Данные значимы сами по себе. Используя их, человечество может изучать критически важную информацию и передавать ее потомкам. Однако лишь недавно люди узнали, что данные могут содержать куда больше информации, чем кажется на первый взгляд. Если данные находятся в соответствующей числовой форме, к ним можно применить специальные методики, разработанные математиками и статистиками. Эти методики анализа данных позволяют извлечь из них куда больше информации. Кроме того, даже простой анализ данных позволяет извлечь из них осмысленную информацию, а подвергнув данные более совершенному анализу, с использованием алгоритмов машинного обучения можно предсказывать будущее, классифицировать информацию и эффективно принимать решения.

Анализ данных и машинное обучение позволяют перейти на следующий уровень использования данных, теперь – для разработки более умного ИИ. Эта глава посвящена анализу данных. Она демонстрирует применение данных в качестве инструмента обучения при решении сложных проблем ИИ, таких как правильная рекомендация товара клиенту, понимание разговорного языка и языкового перевода, автоматизация вождения автомобиля и многие другие.

3.1. Анализ и извлечение данных

Наше время называют веком информации не просто потому, что сейчас накоплено богатое разнообразие данных, но и потому, что общество достигло определенной зрелости в анализе данных и извлечении из них информации. Такие компании, как *Google*, *Amazon*, *Apple*, *Facebook* и *Microsoft* (пять самых дорогих компаний в мире), построили свой бизнес на данных. Они не просто собирают и хранят данные, полученные в результате цифровых процессов, они знают, как, используя точный и сложный анализ данных, сделать их такими же ценными, как нефть.

Впервые данные к нефти приравнял британский математик Клайв Хамбли (*Clive Humbly*) на основании своей практики с данными о потребителях в розничном секторе. В 2006 году Хамбли подчеркнул, что данные – это не просто деньги, которые «падают с

неба»: чтобы сделать их полезными, требуются усилия. Подобно тому, как нельзя непосредственно использовать неочищенную нефть, данные также следует существенно переработать, чтобы они приобрели значимость. Анализ данных, в зависимости от контекста, сводится к большому количеству возможных операций, иногда специфичных для конкретной отрасли или задачи. Все эти преобразования можно отнести к четырем основным категориям, концептуально отличающимся происходящим во время анализа:

1. Преобразование – изменяет внешний вид данных. Их, как правило, помещают в упорядоченные ряды и столбцы – переводят в матричный формат. Например, вы не можете эффективно обработать данные о товарах, купленных в супермаркете, прежде чем поместите каждого клиента в отдельный ряд и добавите купленные товары в один столбец в пределах этого ряда в виде числовых элементов, содержащих значения количества или платы. Чтобы сделать набор данных подходящим для алгоритма, могут также потребоваться специальные числовые преобразования, такие как масштабирование, вычисление среднего, минимального и максимального значений.

2. Чистка – исправляет дефектные данные. В зависимости от средств сбора данных могут возникнуть различные проблемы с отсутствием информации, выбросами значений из диапазона или просто неправильными значениями. Например, данные из супермаркета могут содержать ошибки, если у товаров неправильные ценники. Некоторые данные могут быть подложными, т.е. созданными специально, чтобы исказить заключение. Например, у товара могут быть поддельные отзывы в Интернете, которые изменят его ранг. Чистка помогает удалить подложные случаи из данных и сделать заключение объективней.

3. Проверка данных – по большей части человеческая работа, хотя программное обеспечение играет в ней важную роль. Люди могут легко распознавать шаблоны и выявлять странные элементы данных. Поэтому данный этап подразумевает множество статистических выкладок и графических представлений, таких как у *Health InfoScape* от *MIT Senseable City Lab* и *General Electric* [23], позволяющих сразу схватить информативное содержимое.

4. Моделирование – выявление отношений между элементами в данных. Для решения этой задачи необходимы такие статистиче-

ские инструменты, как корреляции, t-проверки, линейная регрессия и многие другие, позволяющие определить, действительно ли одно значение не зависит от другого или они взаимосвязаны. Например, анализируя продажи супермаркета, вы можете прийти к мнению, что люди, покупающие подгузники, имеют тенденцию покупать и пиво. Статистический анализ считает эти два товара взаимосвязанными, поскольку они многократно обнаруживаются в одних и тех же корзинах.

Более подробную информацию об этих методиках можно получить в книге «Машинное обучение для чайников (и не только)» Кимберли Невала [24].

В случае больших объемов данных их анализ становится существенно сложнее. Для этого требуются иные, более сложные инструменты.

Апофеозом анализа данных является машинное обучение. Вы можете успешно применить машинное обучение только после того, как анализ данных предоставляет правильные исходные данные. Но только машинное обучение способно ассоциировать наборы выходных и входных данных, а также эффективно выявить использованные при этом правила. Анализ данных концентрируется на понимании и манипулировании данными таким способом, чтобы они могли стать более полезными и способными обеспечить проникновение в суть вещей, тогда как машинное обучение строго сосредоточено на том, чтобы получить исходные данные и, сделав свою работу, выработать внутреннее представление о сути вещей, которое можно использовать практически. Машинное обучение позволяет людям решать такие задачи, как предсказание будущего, классификация осмысленным способом и выработка наиболее рационального решения в данном контексте.

Главная идея, лежащая в основе машинного обучения такова: «Можно представить действительность, используя математические функции, которые алгоритму заранее неизвестны, но которые он может предположить на основании наблюдения некоторых данных». Вы можете выразить действительность и всю ее комплексную сложность в терминах неизвестных математических функций, которые алгоритмы машинного обучения способны выявить и сделать доступными. Эта концепция лежит в основе идей всех видов алгоритмов машинного обучения.

Обучение при машинном обучении является просто математической функцией, и она заканчивается ассоциацией определенных входных данных с определенными выходными. Она не имеет отношения к пониманию того, что алгоритм изучил (анализ данных вырабатывает понимание до некоторой степени), поэтому процесс обучения зачастую называют тренировкой, поскольку алгоритм учится находить правильный ответ (вывод) на каждый предоставленный вопрос (ввод). Более подробная информация по этому вопросу приведена в книге Х. Бринка, Дж. Ричардса и М. Феверолфа «Машинное обучение» [25].

Несмотря на отсутствие осмысленного понимания, поскольку это просто математический процесс, машинное обучение может оказаться весьма полезным. Оно дает приложению с искусственным интеллектом силу наибольшей рациональности в текущем контексте, когда обучение происходит с использованием правильных данных.

Многие люди привыкли к идее, что приложения начинают работу с получения данных на входе, а затем предоставляют некий результат. Например, программист мог бы создать функцию *Add* (), которая получает на вход два значения, например, 1 и 2, а затем возвращает результат, 3. Выводом этого процесса является значение. В прошлом написание программы означало понимание того, как функция должна манипулировать данными, чтобы получить заданный результат при определенных входных данных.

Машинное обучение осуществляет переворот в этом процессе. В данном случае вы знаете входные данные, это 1 и 2. Вы также знаете, что желаемый результат – 3. Однако вы не знаете, какую функцию применить, чтобы получить желаемый результат. Обучение предоставляет алгоритм со всякого рода примерами входных данных и ожидаемыми результатами для этих входных данных. Затем алгоритм использует заданный ввод, чтобы создать функцию. Другими словами, обучение — это процесс, в ходе которого обучаемый алгоритм сопоставляет с данными гибкую функцию. Выводом обычно является вероятность определенного класса или числового значения.

Рассмотрим ещё один пример. Вообразите ребенка, учащегося отличать деревья от других объектов. Прежде чем ребенок сможет сделать это самостоятельно, учитель показывает ему изображения

деревьев, отображающие все факты, отличающие дерево от других объектов. К таким фактам могут относиться материал дерева (древесина), его части (ствол, ветви, листья и корни) и расположение (в почве). Ребенок вырабатывает представление о том, как выглядит дерево, в отличие от изображений других объектов, таких как предметы мебели, которые тоже состоят из древесины, но не имеют других характеристик, схожих с характеристиками дерева.

Классификатор машинного обучения работает точно также. Он формирует свои когнитивные способности, создавая математические формулировки, включающие все заданные средства, способом, который определяет функцию, способную отличить один класс от другого. Так, например, во избежание мошенничества ИИ Вашего банка может вас уведомить о выполнении покупки по вашей карте с незнакомым шаблоном расходов.

Для анализа огромных наборов данных машинное обучение полагается на алгоритмы. В настоящее время машинное обучение не способно обеспечить такой вид искусственного интеллекта, как в кино. Даже наилучшие алгоритмы не могут думать, чувствовать, обладать любой формой самосознания или иметь свободу выбора. На что машинное обучение способно, так это выполнять прогнозирующую аналитику намного быстрее, чем любой человек. В результате машинное обучение может помочь людям работать эффективнее. Да, искусственный интеллект в текущем состоянии способен выполнить выдающийся анализ, но смысл его результатов все еще должны осознавать люди, они же принимают необходимые моральные и этические решения на его основании. По существу, машинное обучение обеспечивает лишь часть обучения искусственного интеллекта, и эта часть ничуть не готова создать искусственный интеллект того вида, который вы видите в фильмах.

В машинном обучении все вращается вокруг алгоритмов. Алгоритм – это процедура или формула решения задачи. Предметная область включает и вид необходимого алгоритма, но базовая предпосылка всегда одинакова: решить своего рода задачу, такую как вождение автомобиля или игра в домино. В первом случае задача сложна и многогранна, но сводится в конечном счете к одному - доставить пассажира из одного пункта в другой, не разбив автомобиль. Аналогично задача при игре в домино заключается в победе.

Обучение бывает разным в зависимости от алгоритма и его целей. Алгоритмы машинного обучения можно отнести к трем основным группам на основании их задач.

Контролируемое обучение (обучение с учителем) происходит, когда алгоритм учится на основе данных примеров и относящихся к цели ответов, которые могут состоять из числовых значений или строковых меток, таких как классы или тэги, чтобы впоследствии спрогнозировать правильный ответ, когда появятся новые примеры. Контролируемый подход подобен обучению человека под присмотром учителя. Учитель обеспечивает ученику хорошие примеры для запоминания, а ученик усваивает общие правила из этих конкретных примеров.

Неконтролируемое обучение (обучение без учителя) происходит, когда алгоритм учится на основе простых примеров без какого-либо ассоциированного ответа, позволяя алгоритму самостоятельно определять шаблоны данных. Этот тип алгоритма стремится реструктурировать данные в нечто еще, такое как новая возможность, способная представить класс, или новая последовательность некоррелированных значений. Полученные данные весьма полезны людям со способностью проникновения в суть смысла первоначальных данных, а также как новые полезные входные данные для алгоритмов контролируемого машинного обучения.

Обучение с подкреплением происходит, когда алгоритму предоставляют примеры без меток, как при неконтролируемом обучении. Но вы можете сопроводить примеры положительной или отрицательной оценкой в зависимости от предлагаемого алгоритмом решения.

Обучение с подкреплением связано с приложениями, алгоритм которых должен принимать решения, и эти решения имеют последствия. В мире людей это точно то же, что и обучение методом проб и ошибок.

3.2. Нейронные сети и машинное обучение

Если говорить о нейронных сетях языком программистов, то это просто компьютерный код. Если говорить языком математиков – это сложная композиция функций, в которой есть числовые параметры, настраиваемые по обучающей выборке. Выборкой мо-

жет быть, например, большое количество сыгранных партий, в которых машина играла сама с собой. Алгоритм обучения нейронной сети подбирает параметры так, чтобы на этой выборке ошибаться как можно реже. В ходе обучения алгоритм фактически вырабатывает свои правила принятия решений.

Сейчас стало принято приравнивать нейронные сети и машинное обучение и говорить, что традиционные методы постепенно вытесняются нейронными сетями. Первая линейная математическая модель нервной клетки – нейрона – появилась в 1943 году в работах МакКаллока и Питтса. Уже в 1950-е годы искусственные нейронные сети начинают отходить от принципов нейрофизиологии и оформляются в отдельные, чисто математические, конструкции.

Искусственные нейронные сети пережили несколько волн энтузиазма, сменявшихся разочарованием от перегретых ожиданий. Сейчас мы снова на волне энтузиазма благодаря достижениям машинного обучения. Похоже, что на этот раз все всерьез и надолго.

В 2016 году нейронные сети, обученные на основе глубокого машинного обучения, опередили человека по уровню качества классификаций изображений. Человек в ходе тестов допускает 3÷5% ошибок, а у нейросетей эта доля меньше. То есть работу людей уже можно автоматизировать везде, где речь идет о распознавании, скажем, номеров автомобилей, лиц в толпе, брака на конвейере или военной техники с воздуха. Но надо понимать, что глубокие нейронные сети пока работают лишь для ограниченного круга задач. Это те задачи, которые не поддавались старым методам из-за сложной структуры самих исходных данных, таких как изображения, видео, звук, речь, текст и игры.

Машинное обучение строится на математическом анализе, теории вероятностей, оптимизации и статистике. Ажиотаж вокруг искусственного интеллекта добавил ему магической ауры. При этом у глубоких нейронных сетей есть реальный прорыв – они позволяют автоматизировать процесс извлечения признаков из сырых данных.

Раньше этим занимались целые отделы математиков и инженеров. Например, придумывали кучу изощренных формул, как перевести изображение в числовой вектор, чтобы распознавать иероглифы, автомобили или лица людей. Все это довольно медленно развивалось, потому что требовало многих лет исследований и высококвалифицированного труда.

Теперь произошел переход от инженерии признаков к инженерии нейросетевых архитектур, когда специалист придумывает, сколько в нейросети будет слоев и как он их расположит. Это намного проще, быстрее и дешевле. Но это совершенно не означает, что нейросети настолько универсальны, что способны решать все задачи на свете.

Многие задачи моделирования еще долго будут решаться людьми на основе глубоких знаний предметной области. Главное, чего ИИ пока не может даже близко, это заниматься целеполаганием и ставить задачи [26].

В поисковых системах Яндекс или *Google* нейросети реализуют огромное число сервисных функций. Первая из них – фильтрация поискового спама. Если бы не эта функция, то мы вообще ничего полезного не увидели бы в списках поисковой выдачи. Вторая функция – ранжирование результатов запроса по релевантности. Третья – демонстрация рекламы, на которую с большей вероятностью кликнет пользователь. Даже такая мелочь, как Выпадающий список вариантов продолжения запроса – это тоже сервисная функция на основе использования нейросетей.

Таких интеллектуальных сервисов только на стартовой странице «Яндекса» десятки. Поиск информации на сегодняшний день уже очень эффективен, и это огромный прорыв, по сравнению с тем, что было 30 лет назад, когда за информацией надо было ходить в библиотеки.

Самая большая беда для машинного обучения - это неочищенные (грязные) данные. Например, если в ритейле вы неаккуратно собираете данные об остатках и пересортице, неправильно считаете потери, то алгоритмы для планирования поставок будут работать неверно. Надо очень дисциплинированно отстроить бизнес-процессы, потратить на это деньги и время, чтобы затем строить интеллектуальные решения на чистых данных. Бизнес, работающий по старинке, этого часто не понимает.

В результате оказывается, что данные неполные, неточные или содержат намеренно искажаемую информацию. Цифровизация экономики – это в первую очередь выстраивание прозрачных бизнес-процессов сбора и обработки данных с учетом того, какие задачи предсказательного моделирования на этих данных должны решаться, с каким качеством и по каким бизнес-метрикам.

3.3. Глубокое обучение

Первым шагом к пониманию того, как работает глубокое обучение, является осмысление различий между понятиями нейронная сеть (искусственная нейронная сеть), искусственный интеллект и машинное обучение.

Подытожив рассмотренные материалы данной главы, перечисленные понятия возможно определить следующим образом:

– искусственный интеллект – это способность машины или программы находить решения при помощи вычислений;

– нейронная сеть – это попытка воспроизведения работы человеческого мозга на компьютере при помощи слоев искусственных нейронов;

– машинное обучение – это попытка научить компьютеры самостоятельно обучаться на большом количестве данных вместо жестко постулированных правил.

Уточним, что в отношении нейросетей контролируемое обучение или обучение с учителем подразумевает использование выбранных примеров данных, содержащих входные данные и ожидаемые выходные результаты (рис. 3.1).

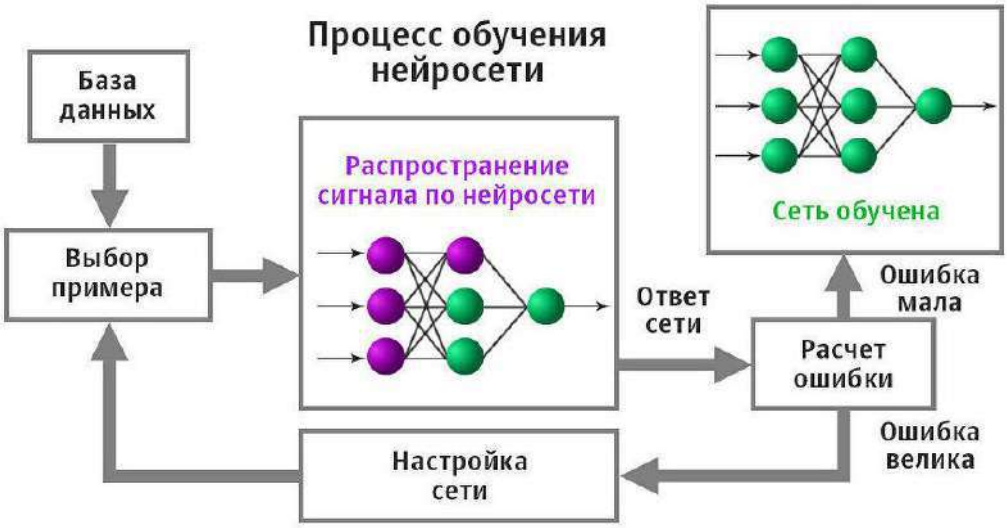


Рис. 3.1. Обучение нейронной сети

Когда выполняется процесс обучения нейронной сети с учителем, ей демонстрируются как входные данные, так и ожидаемые выходные результаты. Если результат, генерируемый нейронной сетью, неверен, выполняется коррекция настроек. Это итерационный процесс, оканчивающийся тогда, когда сеть перестает совер-

шать ошибки. Примером задачи с контролируемым обучением является предсказание погоды. Нейросеть учится делать прогноз погоды с использованием исторических данных. Обучающие данные включают в себя входные данные (давление, влажность, скорость ветра) и выходные результаты (температура).

Неконтролируемое обучение нейросети или её обучение без учителя – это машинное обучение с использованием наборов данных без определенной структуры. Когда нейросеть обучается неконтролируемо, она самостоятельно проводит логическую классификацию данных. Примером задачи с неконтролируемым обучением является предсказание поведения посетителей интернет-магазинов. В этом случае сеть не обучается на размеченных данных. Вместо этого она самостоятельно классифицирует входные данные и отвечает на вопрос, какие пользователи чаще всего покупают различные товары.

Глубокое обучение – это тоже метод машинного обучения. Оно позволяет обучать модель предсказывать результат по набору входных данных. При этом для обучения нейросети можно использовать как контролируемое, так и неконтролируемое обучение.

При глубоком обучении используются крупные нейронные сети, в которые передаются алгоритмы обучения и постоянно увеличивающиеся объемы данных, повышающие эффективность их «мышления» и «обучения». Обучение называется «глубоким», поскольку в течение времени нейронная сеть охватывает все большее число уровней, и чем «глубже» проникает сеть, тем выше ее производительность [27].

Рассмотрим, как работает глубокое обучение, на примере сервиса по оценке стоимости авиабилета. Мы будем обучать его контролируемым образом. При этом мы хотим, чтобы наш сервис предсказывал цену (*Price*) на авиабилет по следующим входным данным:

– аэропорт вылета (*Origin Airport*);

– аэропорт назначения (*Destination Airport*);

– дата отбытия (*Departure Date*);

– авиакомпания (*Airport*).

Как и у животных, искусственная нейронная сеть содержит взаимосвязанные нейроны. На рис. 3.2 они представлены кругами.

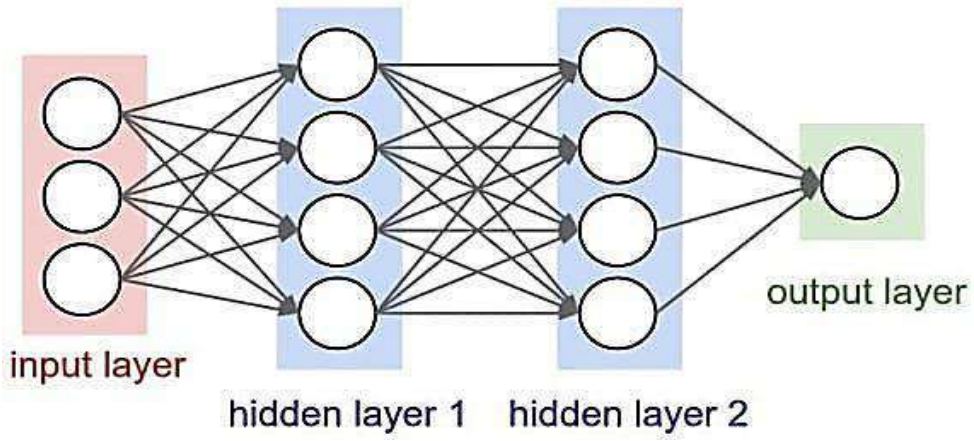


Рис. 3.2. Глубокая нейронная сеть (с двумя скрытыми слоями)

Нейроны сгруппированы в три различных типа слоев.

Входной слой (*input layer*) принимает входные данные. В нашем случае во входном слое должно быть четыре нейрона (вместо трёх указанных на рис. 3.1): аэропорт вылета, аэропорт назначения, дата вылета и авиакомпания. Входной уровень передает эти данные в первый скрытый слой.

Скрытые слои (*hidden layer* 1 и *hidden layer* 2) выполняют математические вычисления со входными данными. Одна из задач при создании нейронных сетей – определение количества скрытых слоев и нейронов на каждом слое.

Слово «глубина» в термине «глубокое обучение» означает наличие более чем одного скрытого слоя. Для примера увеличим число скрытых слоёв с двух до пяти (рис. 3.3).

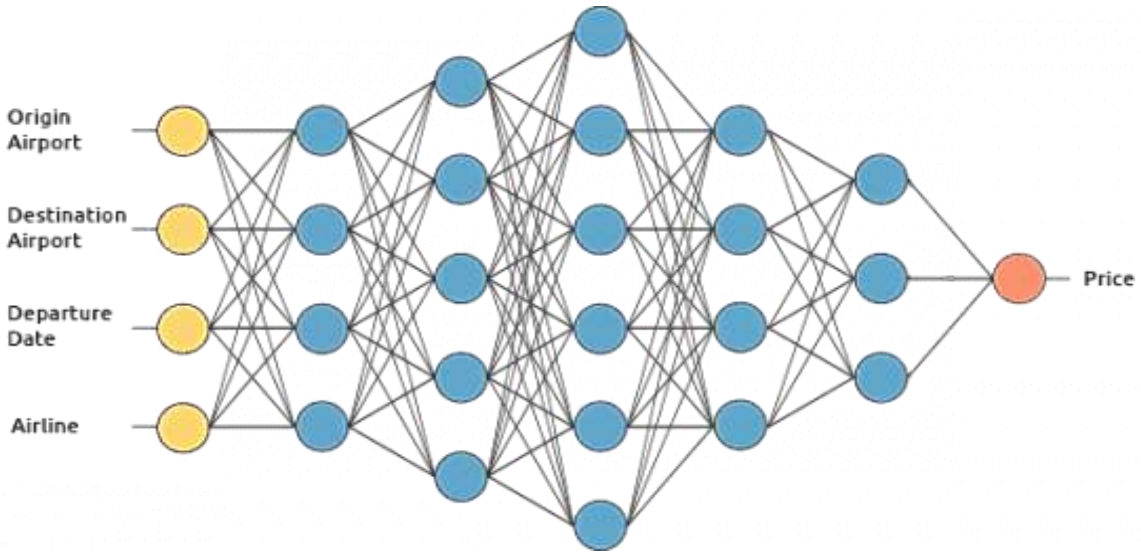


Рис. 3.3. Глубокая нейронная сеть для предсказания цены на авиабилеты

Выходной слой (*output layer*) выдает результат. В нашем случае это прогноз цены на билет.

Цена авиабилета рассчитывается после глубокого обучения применительно к нейронной сети. Её нейроны связаны между собой с определенным весом. Вес определяет важность элемента входных данных. Исходные веса задаются случайным образом.

Глубина нейронной сети позволяет ей построить иерархию объектов с возрастающей абстракцией, при этом каждый последующий слой выступает в качестве фильтра для все более сложных функций, объединяющих функции предыдущего уровня. Эта иерархия функций и фильтры, которые моделируют значимость данных, создаются автоматически, когда глубокие сети учатся восстанавливать неструктурированные данные.

Поскольку глубокие нейронные сети могут работать с неструктурированными данными, которые составляют большинство данных в мире, они могут стать более точными, чем традиционные алгоритмы машинного обучения, которые не могут обрабатывать неструктурированные данные. То есть глубокие нейронные сети с доступом к большему количеству данных всегда выигрывают.

4. АППАРАТНЫЕ ПРИЛОЖЕНИЯ

Современный этап развития ИИ характеризуется широким применением терминов «агент» и «мультиагентная система».

Под агентами понимаются разумные сущности, взаимодействующие с окружающей средой, их поведение рационально – они способны к пониманию. Действия агентов всегда направлены на достижение какой-либо цели. Агент может быть как роботом, так и встроенной программной системой, которая взаимодействует с окружающей средой примерно так же, как действовал бы человек. Поэтому агентов принято называть интеллектуальными.

Важно отметить, что совокупность из нескольких агентов может решать задачи, которые не под силу каждому из агентов в отдельности. В этом случае говорят, что начинает работать коллектив (рой) взаимодействующих агентов, который принято также называть мультиагентной системой.

В этой главе детализируется понятие агента, возможные варианты его исполнения в виде роботов и дронов, рассматривается понятие мультиагентной системы и вариант исполнения интеллектуального робота-агента – беспилотного автомобиля.

4.1. Агенты: роботы, дроны и другие

Общее понятие определяет агента, как всё то, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов (рис. 4.1).

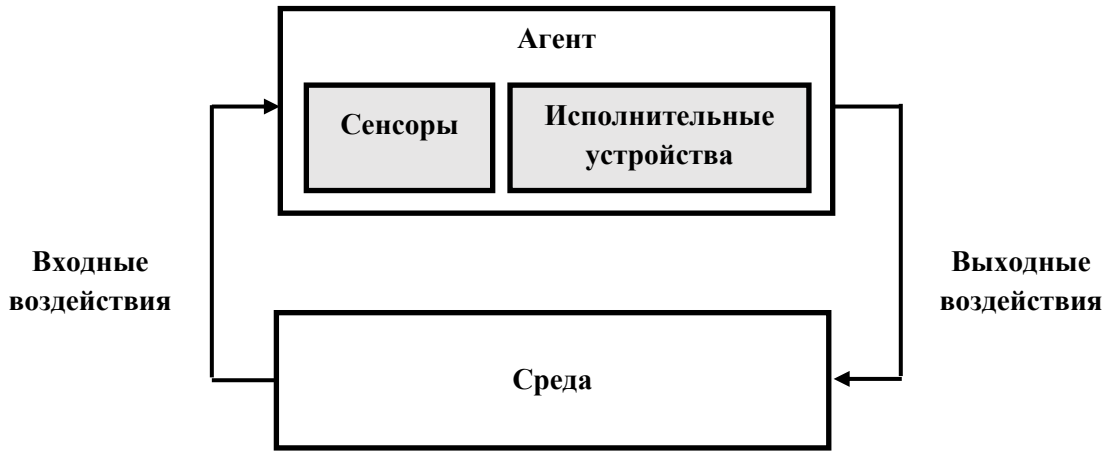


Рис. 4.1. Общая схема агента

Понятие агента на данный момент не является строго формализованным. С точки зрения программирования, агент – это развитие известного понятия объект, представляющего абстракцию множества экземпляров предметов реального мира, имеющих одни и те же свойства и правила поведения. Общепринято считать агента объектом, обладающим всеми или некоторыми из основных свойств, к которым относятся, в частности (рис. 4.2):

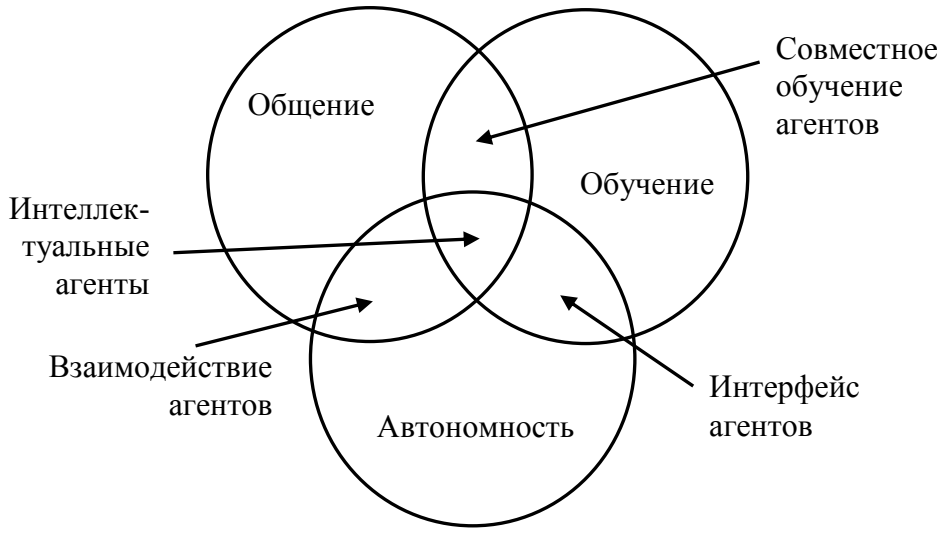


Рис. 4.2. Графическое представление основных свойств агента

– автономность: агенты функционируют без прямого вмешательства кого-либо и обладают определенной способностью контролировать свои действия и внутреннее состояние;

– способность общения: агенты взаимодействуют с другими агентами, с окружающей их средой (и, возможно, людьми) посредством какого-либо коммуникационного языка;

– реактивность: агенты обладают способностью воспринимать среду и адекватно реагировать в определенных временных рамках на происходящие изменения;

– проактивность: агенты не просто реагируют на изменения среды, но и обладают целенаправленным поведением и способностью проявлять инициативу;

– автономность: каждый агент имеет собственную модель окружающего его мира (среды), которая описывает то, как агент видит мир; агент строит свою модель мира на основе информации, которую получает из внешней среды;

– интеллектуальное поведение: поведение агента включает способность к обучению, логичной дедукции или конструированию модели окружающей среды для того, чтобы находить оптимальные способы поведения.

С точки зрения разработчиков информационных систем агент – это модуль ПО, выполняющийся на определенной платформе, совершающий некоторые действия, такие как отображение и ввод информации, и обменивающийся сообщениями с другими агентами или человеком.

Как отмечал Й. Шохем, «агент обладает ментальным поведением, которое зависит от среды, в которой он находится». Ментальное поведение – способность животных изменять свои действия под влиянием внутренних и внешних факторов, характерная черта животного типа организации.

Само понятие «ментáльность» (от лат. mens, mentis, «разум, ум, интеллект») не имеет точного аналога в русском языке. Ментальность противопоставляется «материи» или, в более современных контекстах, «мозгу», при этом отношение между «мозгом» и «ментальностью» образно сравнивается с отношением между аппаратным и программным обеспечением, то есть мозг понимается как нейрофизиологический (материальный) субстрат «ментальности», тогда как «ментальность» – в качестве присущей этому субстрату функциональности.

В психологии «ментальность» – это интеллектуально-эмоциональные особенности индивида, мысли и эмоции которого неразделимы, где мысли – диктуются культурой, а эмоции – реакция на изменения внешней среды, которая опирается на культурные ценности индивида. При этом ментальность формируется в процессе воспитания и обретения жизненного опыта. Таким образом, ментальность – это то, чем различаются индивиды, получившие воспитание в различных культурных средах.

В традиционном значении «ментальность» синонимична «менталитету» и подразумевает (как правило, в социологических контекстах) тот или иной «склад ума», то есть устойчивые интеллектуальные и эмоциональные особенности, присущие тому или иному индивиду (обычно как представителю некоторой социальной группы). Как вариант, ментальность – способ видения мира, в котором мысль не отделена от эмоций.

Если же говорить об аппаратном воплощении (разработке и реализации) агентов – робототехнике, то люди привыкли принимать её за ИИ. Однако робототехника отличается от него. ИИ нацелен на поиск решения некоторых трудных задач, связанных с человеческими способностями (такими, как распознавание объектов, понимание речи и т.п.). Робототехника же стремится использовать машины для решения задач в физическом мире за счет частичной или полной автоматизации. Поэтому ИИ можно считать программным обеспечением (в виде агента) для решения задач, а робототехнику – аппаратными средствами для того, чтобы сделать эти решения реальностью.

Робототехнические аппаратные средства могут использовать программные средства систем ИИ, а могут и не использовать. Во многих случаях ИИ действительно обеспечивает усиление способностей человека, но контролирует ситуацию все еще человек.

Первый коммерческий робот, *Unimate*, появился в 1961 году. Это был просто роботизированный манипулятор (программируемая механическая рука, состоящая из металлических балок и суставов), рабочий конец которого мог держать, вращать или сваривать объекты согласно инструкциям, заданным человеком-оператором. Он был продан компании *General Motors* для использования в автомобилестроении. Робот *Unimate* должен был извлекать отливки из форм на сборочной линии и сваривать их вместе – задача физически опасная для рабочих-людей. Возможности данной машины отражены в видео [28] (реалиям современных роботов посвящены следующие разделы).

С тех пор роботы существенно продвинулись и могут быть очень разными, от получающих абстрактные распоряжения от людей (такие, как переместиться из точки А в точку В по карте или поднять объект) до полагающихся на ИИ для выполнения более сложных распоряжений. Некоторые роботы способны решать задачи автономно, безо всякого человеческого вмешательства. Интеграция ИИ делает робота более умным и более полезным в решении задач, но роботам ИИ для работы нужен отнюдь не всегда. Человеческое воображение сделало эти два понятия неразделимыми благодаря научно-фантастическим романам и фильмам.

Люди все чаще применяют роботов в промышленности, в научных изысканиях, в медицине и в военных целях. Новейшие исследования в области искусственного интеллекта ускоряют этот процесс, поскольку они решают такие трудные для роботов задачи, как распознавание объектов в реальном мире, прогнозирование поведения людей, понимание голосовых команд, правильная речь, ходьба, плавание, уборка мусора [29] и т.п.

4.2. Мультиагентные системы

Для определения агентной системы сначала детализируем понятие агента. Как уже было отмечено, в самом общем понимании агент – система, находящаяся в среде, действующая в среде и воспринимающая среду (рис. 4.3).

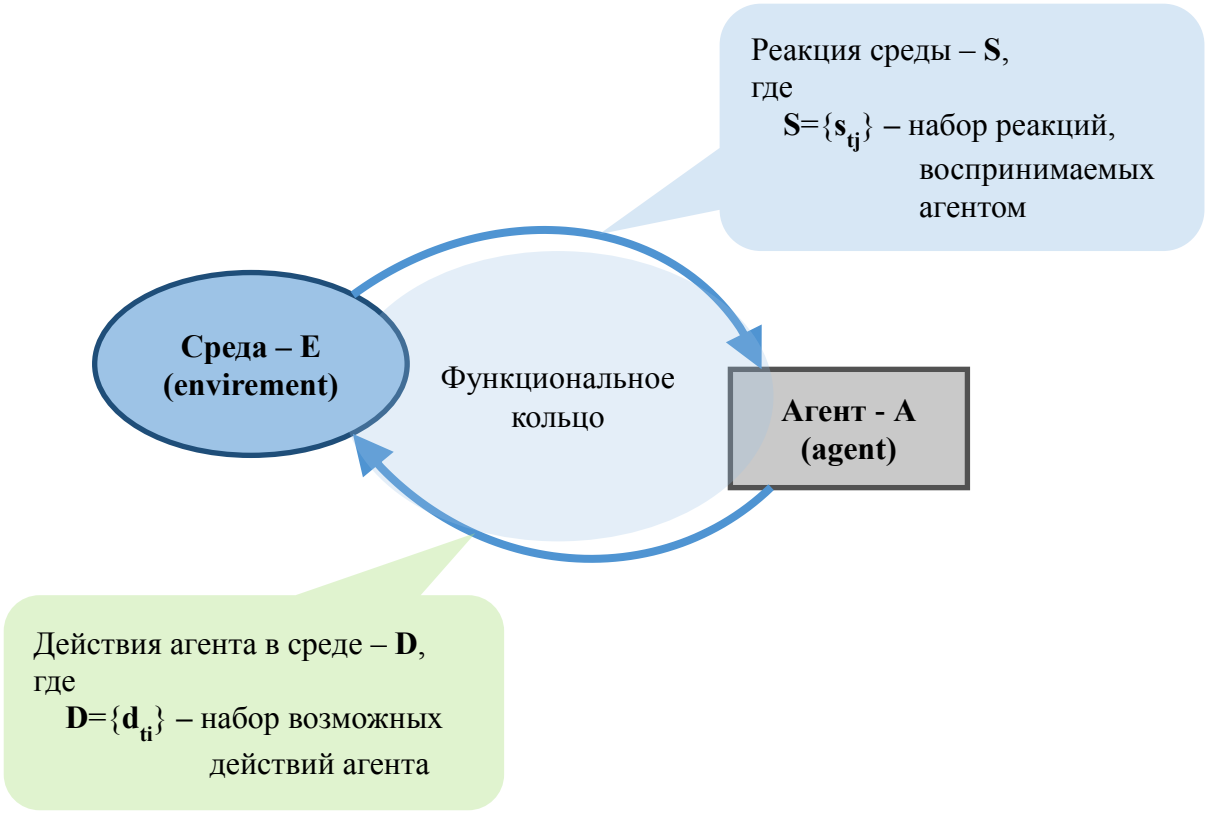


Рис. 4.3. Взаимодействие агента со средой

Среду (*envirement, E*) можно рассматривать как ближайшее окружение агента, с которым он непосредственно взаимодействует. Например, для программного агента средой является «операционная система», для робота марсохода – некоторый участок марсианской поверхности (рис. 4.4).

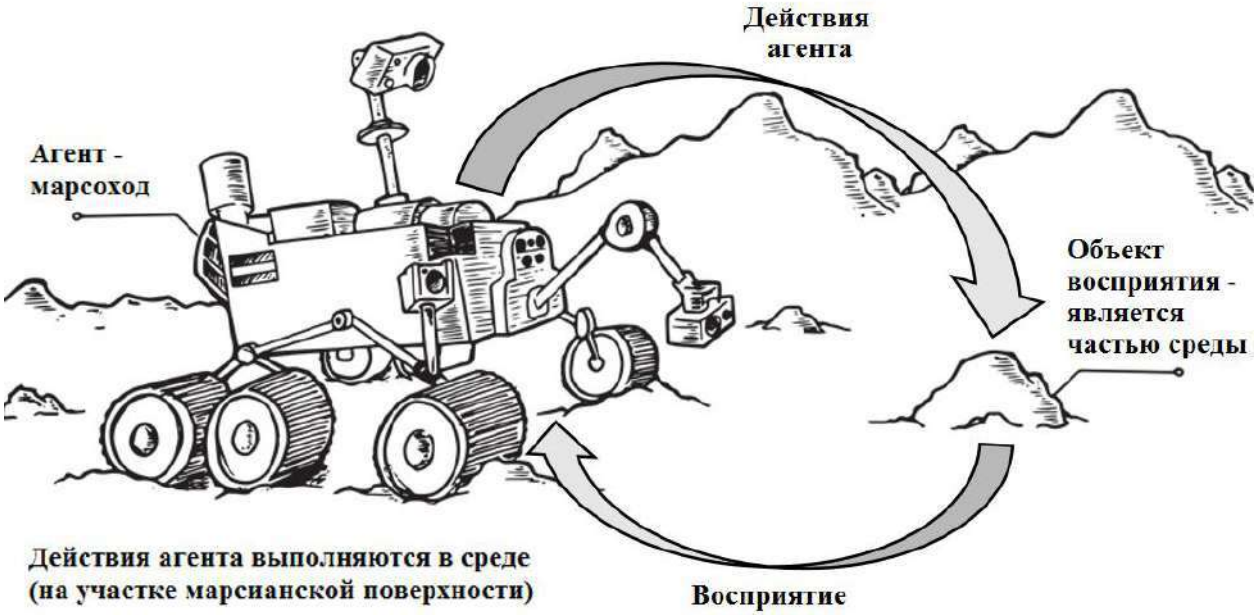


Рис. 4.4. Робот-агент – марсоход

Характеристики среды влияют на архитектуру агента и мультиагентной системы. Чем проще среда, тем проще построить агента (например, мир шахмат для этого – идеальный: обозримый и замкнутый). Сложнее всего строить агентов для реального мира – сложной среды (например, для уборки мусора с пола в квартире, для вождения автомобиля, поиска информации на вебе и т.п.).

Сложность среды, в которой функционирует агент, влияет на сложность процесса принятия решений (агентом). Очевидно, что среда имеет множество параметров, влияющих на архитектуру агента, ведь параметры определяются именно с его точки зрения.

Среда определяется через множество её возможных характеристик:

– полностью/частично обозримая;
– детерминированная/случайная;
– эпизодическая/последовательная;
– дискретная/непрерывная;
– статичная/динамичная;
– одноагентная/многоагентная.

В полностью обозримой среде сенсоры агента полностью воспринимают состояние среды в каждый момент времени. Агент может получить полную, точную, своевременную информацию о состоянии среды.

Частично обозримой среда становится из-за зашумленных, неточных сенсоров или ограничений области восприятия.

В детерминированной среде любое действие приводит к единственному гарантированному результату – то есть отсутствует неопределенность состояния, вызванного действием. Последующее состояние определяется предыдущим состоянием и действием агента. Среда должна быть полностью обозримой.

Случайная среда характеризуется тем, что все события по отношению к агенту являются внешними. Реакция среды на действия агента неоднозначна/нечетка.

Для эпизодической среды поведение агента можно разделить на отдельные эпизоды. Эпизоды не связаны между собой. Нет необходимости «думать» наперёд.

В последовательной среде эпизоды связаны между собой или отсутствуют. Кратковременные действия имеют долговременные последствия. Свои действия агенту необходимо планировать.

Статичная среда может меняться только от действий агента. Нет необходимости наблюдать за средой во время принятия решений.

Динамичная среда может меняться от внешних по отношению к агенту факторов. Среда может измениться во время принятия решений, «думать» надо быстрее.

Дискретная среда предполагает дискретность состояния или времени, или процесса восприятия, или действий.

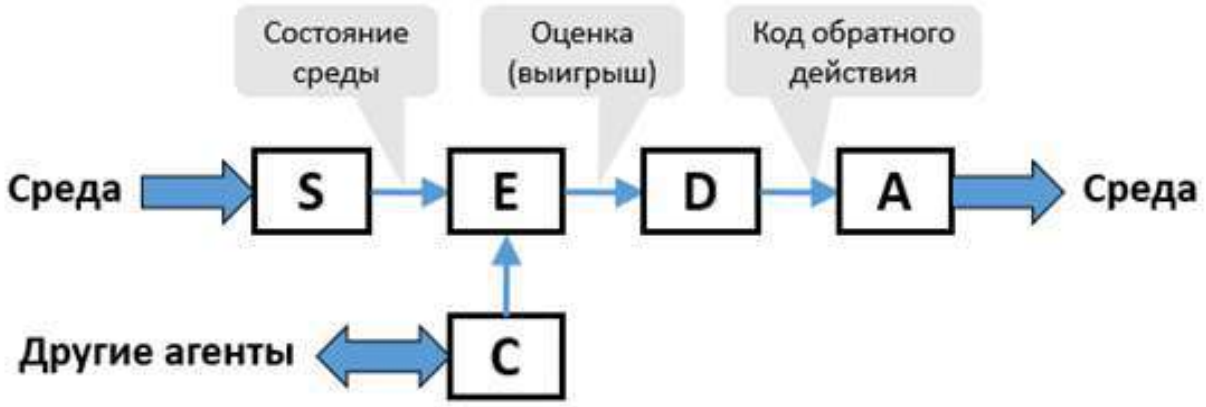
Непрерывная среда может рассматриваться как противоположность дискретной.

Для одноагентной среды характерно то, что другие агенты не влияют на достижение цели данного агента (находящегося в среде). А вот в мультиагентной среде всё наоборот – все агенты действуют взаимосвязано и каждый из них влияет на достижение цели другого. Саму мультиагентную систему определяют как совокупность интеллектуальных агентов, которые действуют совместно для достижения конкретной цели.

Формально агент является элементом (частью) системы. Система (мультиагентная система) – это коллектив агентов. Функциональная целостность системы предполагает такой тип внутреннего взаимодействия её элементов, при котором свойства целого (т.е. всей системы) нельзя свести к сумме свойств элементов. При

этом проявляется эмерджентность – возникают качественно новые свойства при объединении элементов (агентов) в систему. Поэтому считается, что коллективными действиями можно достичь больших результатов чем индивидуальными.

Основная проблема, которую решает агент – это проблема выбора: какое из всех доступных действий выбрать и выполнить для того, чтобы достичь поставленной перед ним цели. Для этого общая функциональная структура агента состоит из следующих блоков (рис. 4.5).



S (*sense*) – сенсорная система
E (*estimate*) – блок оценки
D (*decide*) – блок принятия решения
A (*actuate*) – исполнительная система
C (*communicate*) – блок информационного взаимодействия с другими агентами

Рис. 4.5. Общая функциональная структура агента

S (*sense*) – сенсорная система. Она отвечает за получение информации о состоянии среды, например, в виде значений параметров, измеряемых датчиками сенсорной системы (температура, давление, радиоактивность), или в виде изображений, полученных с помощью видеокамеры.

C (*communicate*) – блок информационного взаимодействия с другими агентами. Он обеспечивает обмен информацией заданного содержания и формата с соседними агентами.

E (*estimate*) – блок оценки. Он формирует сигнал проигрыша или выигрыша на основе информации о текущем состоянии среды и информации от блока информационного взаимодействия.

D (*decide*) – блок принятия решения. Он отвечает за выбор следующего действия, исходя из информации об успешности преды-

дущих действий (пример: автомат с линейной тактикой, обеспечивающий сходимость к выигрышному решению в условиях стационарной случайной среды).

A (*actuate*) – исполнительная система. Она обеспечивает выполнение (реализацию) выбранных действий (принятых решений) (например, выполнение перемещения робота-агента в пространстве в выбранном направлении).

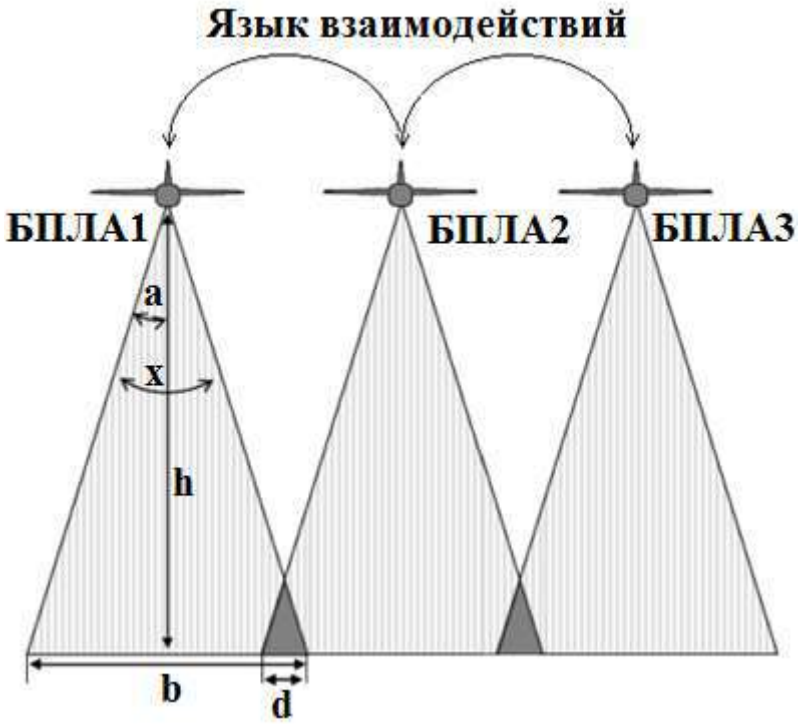


Рис. 4.6. Схема взаимодействия агентов-дронов

В качестве примера рассмотрим уже сложившуюся практику мультиагентного выполнения аэрофотосъемки на основе дронов (беспилотных летательных аппаратов – БПЛА). При этом каждый агент-дрон имеет ограниченное поле «зрения» и должен работать в команде, чётко согласовывая своё положение над снимаемой поверхностью, угол захвата (x), высоту полёта (h), ширину зоны перекрытия (d) (рис. 4.6) и ещё ряд параметров с другими агентами-дронами, работающими в одной команде.

Для согласования действий между такими агентами применяется язык обмена данными, основанный на специально разработанной и формализованный онтологической модели (о них речь пойдёт в следующей главе).

4.3. Беспилотные автомобили

Беспилотный автомобиль, робот-автомобиль, или, точнее интеллектуальный агент-автомобиль – это транспортное средство, оборудованное системой автоматического управления, которое может передвигаться без участия человека.

Очевидно, что беспилотные автомобили способны передвигаться самостоятельно благодаря специальному программному обеспечению и сенсорам – основным характерным признакам агента. Программное обеспечение управляет работой всех систем автомобиля: поворотами руля, сменой передач, газом и тормозом. Сенсоры собирают информацию об окружающей обстановке, которая ложится в основу действий автомобиля.

К набору датчиков интеллектуального агента-автомобиля относятся следующие: лидар (дальномер оптического распознавания), радар, камера, система глобального позиционирования (*GPS*, Глонасс), датчики одометрии и гиростабилизатор.

Программное обеспечение беспилотного автомобиля, как правило, основано на использовании машинного зрения и нейросетей.

В качестве примера рассмотрим устройство беспилотного автомобиля от компании Яндекс. Его датчики собирают информацию об окружающей среде, а интеллектуальное программное обеспечение её обрабатывает. Лидар (лазерный радар) создает трёхмерную карту окружающего пространства, сканируя его с полным обзором в 360 градусов. Трёхмерная карта позволяет определить расстояния до объектов. Для детализации карты данные принимаются не только с лидара, но и с камер и радаров, которые могут «видеть» намного дальше чем лидары (рис. 4.7).

В блок на крыше беспилотного автомобиля установлены 3 лидара, 5 камер, антенны *GPS* и мобильной связи *GSM* и антенна спутниковой системы навигации *GNSS*. В передней части машины установлены 4 радара, но могут присутствовать и дополнительные радары. Сочетание данных, полученных с разных приборов формирует наиболее точное позиционирование и безопасную траекторию движения беспилотного автомобиля.

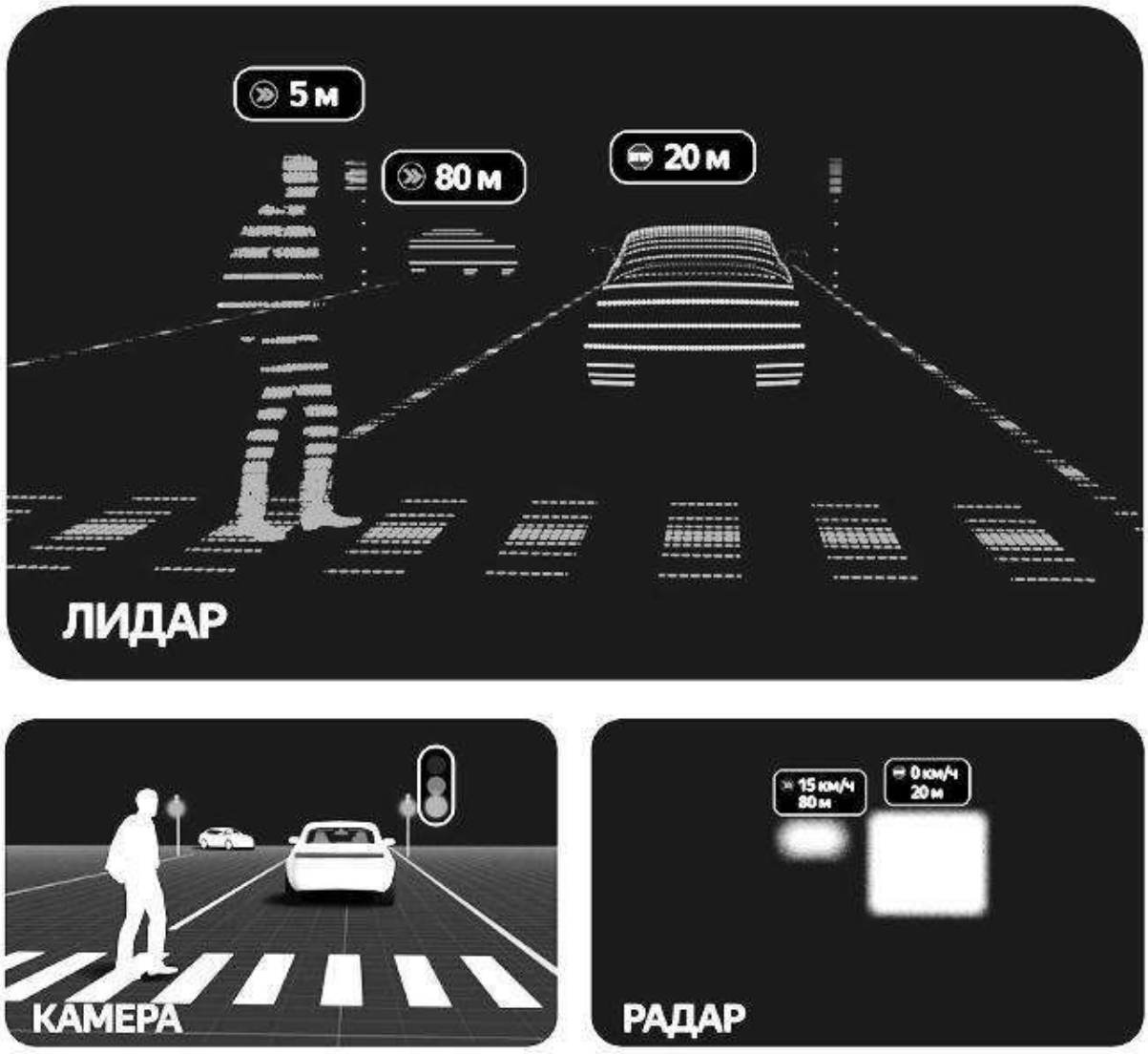


Рис. 4.7. Детализация трёхмерной карты окружающего пространства

Во время движения беспилотный автомобиль накапливает и анализирует все данные, поступающие на его компьютер, который располагается в задней части машины. Вычислительная мощность компьютера автомобиля позволяет обрабатывать сотни гигабайт поступающей информации в течение каждой поездки. При этом данные с каждой машины впоследствии анализируются в едином центре Яндекса после поездки.

В течение поездки информация не передается в центр управления, то есть машины ведут себя абсолютно автономно. Благодаря этому беспилотные автомобили Яндекса не нуждаются в постоянном и быстром мобильном Интернете, облачных вычислениях и защищены от внешнего взлома и хакерских атак.

При поездках на реальных дорогах и на полигонах оцениваются все возможные ситуации во время движения беспилотного автомобиля: неожиданные препятствия, тени и яркое освещение, проезд других машин на красный, перебегающий дорогу пешеход, подрезание другими автомобилями, снег, туман и дождь, проблемы со связью и т.д.

Беспилотный автомобиль оперативно реагирует на неожиданные препятствия: останавливается и продолжает движение, когда «решает», что путь свободен.

Для того, чтобы пассажиры в салоне не волновались и понимали, как оценивает среду и принимает решения автомобиль, в каждой машине установлен планшет, на котором видно, как машина воспринимает среду и окружающие объекты (в реальном времени отображается траектория движения машины, видео с камер, дорожные знаки, разметка, окружающая обстановка и другие участники движения).

5. РАБОТА СО ЗНАНИЯМИ

Данные – совокупность зафиксированных фактов. Информация – сведения, уменьшающие неопределённость. Знания – сведения, позволяющие действовать с прогнозируемым результатом. Типичная проблема: мы располагаем данными, они хранятся в цифровом виде, но мы не знаем, что в них.

Именно поэтому инженерия знаний сегодня является краеугольным камнем ИИ [30]. В данной главе рассмотрены основы наиболее востребованных направлений этой предметной области: онтологии, интеллект-карты и нечёткая логика.

5.1. Онтологии

В последние годы разработка онтологий – явное формальное описание терминов предметной области и отношений между ними – переходит из мира лабораторий по искусственному интеллекту на рабочие столы экспертов по предметным областям. Во всемирной паутине онтологии стали обычным явлением. Во многих дисциплинах сейчас разрабатываются стандартные онтологии, которые могут использоваться экспертами по предметным областям для совместного использования и аннотирования информации в своей области. Например, в области медицины созданы большие стандартные, структурированные словари, такие как *SNOMED CT* (*Systematized Nomenclature of Medicine – Clinical Terms*) и онтология для накопления знаний в сфере биомедицинской информации и информации в сфере здравоохранения *UMLS* (*Unified Medical Language System*) [31]. Также появляются обширные общецелевые онтологии. Например, Программа ООН по развитию (*the United Nations Development Program*) и компания *Dun & Bradstreet* объединили усилия для разработки онтологии *UNSPSC*, которая предоставляет терминологию товаров и услуг [32].

Онтология определяет общий словарь для ученых и специалистов, которым нужно совместно использовать информацию в предметной области. Она включает машинно-интерпретируемые формулировки основных понятий предметной области и отношения между ними.

Совместное использование людьми и/или программными агентами общего понимания структуры информации является одной из

наиболее общих целей разработки онтологий. К примеру, пусть несколько различных веб-сайтов содержат информацию по медицине или предоставляют информацию о платных медицинских услугах, оплачиваемых через Интернет. Если эти веб-сайты совместно используют и публикуют одну и ту же базовую онтологию терминов, которыми они все пользуются, то компьютерные агенты могут извлекать информацию из этих различных сайтов и накапливать ее. Агенты могут использовать накопленную информацию для ответов на запросы пользователей (людей или иных агентов) или как входные данные для других приложений.

Обеспечение возможности использования знаний предметной области стало одной из движущих сил недавнего всплеска в изучении онтологий. Например, для моделей многих различных предметных областей необходимо сформулировать понятие времени. Это представление включает понятие временных интервалов, моментов времени, относительных мер времени и т.д. Если одна группа ученых детально разработает такую онтологию, то другие могут просто повторно использовать ее в своих предметных областях. Кроме того, если нам нужно создать большую онтологию, мы можем интегрировать несколько существующих онтологий, описывающих части большой предметной области.

В России существенный вклад в развитие идей онтологического моделирования внесен школой Т.А. Гавриловой [33] и три этапа, лежащие в его основе, определяются следующим образом.

1. Онтологическое моделирование начинается с онтологического анализа предметной области, конкретно – с составления словаря терминов, который используется при исследовании характеристик системы и процессов, составляющих рассматриваемую область, а также создания базы данных определений этих терминов.

2. Любая система характеризуется, как минимум, двумя категориями предметов восприятия: непосредственно объекты-составляющие и взаимосвязи между ними (характеризующие состояние системы). Поэтому на втором этапе документируются основные логические взаимосвязи (отношения) между соответствующими терминами и понятиями. Результатом этого анализа является онтология системы, или же совокупность словаря терминов, точных их определений и взаимосвязей между ними.

3. Представление онтологии только на уровне пп.1 и 2 зачастую является недостаточным для её понимания без соответствующих примеров использования. Поэтому на практике для представления онтологии множество понятий и отношений между ними, как правило, дополняется набором вариантов интерпретации указанной совокупности на конкретных примерах.

Формальная модель онтологии *O* выражается упорядоченной тройкой вида

$$O = <X, R, F>,$$

где *X* – конечное множество концептов (классов, понятий, терминов) предметной области, которую представляет онтология O;

R – конечное множество отношений между концептами заданной предметной области;

F – конечное множество функций интерпретации (аксиоматизация), заданных на концептах и/или отношениях онтологии O.

Так, например, если необходимо подготовить онтологию подготовки карточек с описанием книг учебного назначения, то этапы формирования онтологии выльются в решение следующих задач (данный пример простой, но понятный и показательный).

Во-первых, составим словарь концептов для описания книг. С целью упрощения понимания материала используем, например, всего пять концептов. Для описания книг при этом используем список карточек, который содержит два раздела: описание служебной информации и непосредственно перечень книг. Служебную информация будем представлять формальным названием списка и датой его создания, а каждый литературный источник – названием книги и ФИО её автора.

Словарь концептов (конечное множество X подготавливаемой онтологии) может быть представлен набором пар вида (имя концепта – назначение) следующим образом:

booklist – основной класс, представляющий список книг;

speshial – класс, несущий служебную информацию о списке книг;

book – класс, представляющий один из литературных источников;

title – слот (атрибут), который будет содержать название книги;

author – слот (атрибут), который будет содержать ФИО автора;

data – слот (атрибут) класса *speshial* (для указания даты списка).

Во-вторых, упорядочим отношения между концептами предметной области. Для этого определим множество R в явном виде с помощью четырёх видов отношений:

1) необязательное отношение «один к одному»:1:0..1;

2) обязательное отношение «один к одному»: 1:1;

3) необязательное отношение «один ко многим»: 1:0..n;

4) обязательное отношение «один ко многим»: 1:1;

Результат, полученный в виде диаграммы класса *booklist*, будет выглядеть так, как это представлено на рис. 5.1. Эту диаграмму принято строить по стандарту *UML* (в ее концептуальном аспекте).

Один список должен включать хотя бы одну книгу. При этом служебная отметка датой создания списка может быть не проставлена. Книга из списка обязательно должна иметь название, однако может не иметь автора, либо иметь их сразу несколько.

В-третьих, для наглядности дальнейшего изложения определим свойства концептов единым типом данных – строкой текста.

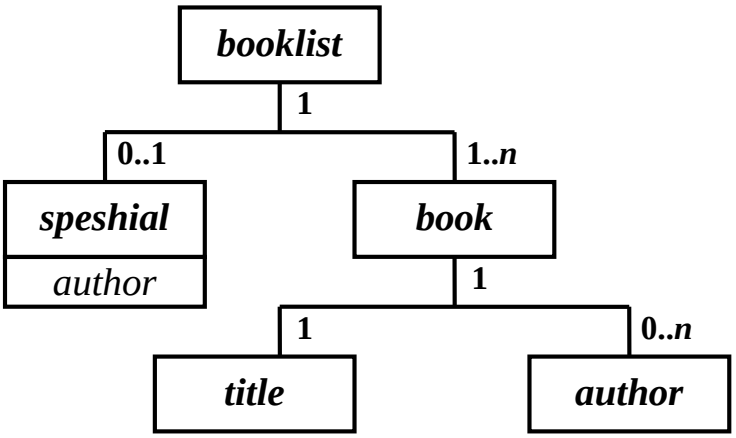


Рис. 5.1. Диаграмма класса *booklist*

Теперь, на основании правил расширяемого языка разметки *XML* (*eXtensible Markup Language*) возможно получение соответствующей грамматики в виде файла:

```
<?xml version=«1.0» encoding=«windows-1251» ?>
<!ELEMENT    booklist   (speshial?, book+)>
<!ELEMENT    speshial   (#PCDATA)>
<!ELEMENT    book   (title, author*)>
<!ELEMENT    title      (#PCDATA)>
<!ELEMENT    author     (#PCDATA)>
<!ATTLIST    speshial   data     CDATA     #REQUIRED>
```

Использование этого файла позволит нам продемонстрировать выполнение последнего, третьего этапа разработки онтологии – создание одного из возможных вариантов заполнения значений слотов.

В-третьих, составим и продемонстрируем вариант *XML*-документа, содержащего информацию о списке из двух публикаций, входящих в список литературы по онтологическому моделированию:

```
<?xml version=«1.0» encoding=«windows-1251» ?>
<!DOCTYPE booklist SYSTEM «booklist.dtd»>
<booklist>
<speshial data=«22.01.2020»>Книги об онтологическом моделировании</speshial>
<book>
    <title> Базы знаний интеллектуальных систем</title>
    <author> Гаврилова Т.А.</author>
    <author> Хоревский Ф.В.</author>
</book>
<book>
    <title>Онтологическое моделирование как средство борьбы с информационным мусором </title>
    <author> Мохов В.А.</author>
    <author> Гринченков Д.В.</author>
</book>
</booklist>
```

В отношении инструментальных средств для работы с онтологиями лидирующие позиции занимает Стэндфордская разработка *Protégé* – свободно распространяемый редактор онтологий с открытым исходным кодом и одновременно структура для создания интеллектуальных систем. При этом, как отмечается на официальном сайте *Protégé*, «поддерживается сильным сообществом академических, правительственных и корпоративных пользователей, которые используют его для создания решений на основе знаний в таких разнообразных областях, как биомедицина, электронная коммерция и организационное управление» [34].

Пример эффективного применения онтологий – это их использование для планирования пассажирских перевозок общественным транспортом во Франции (рис. 5.2).

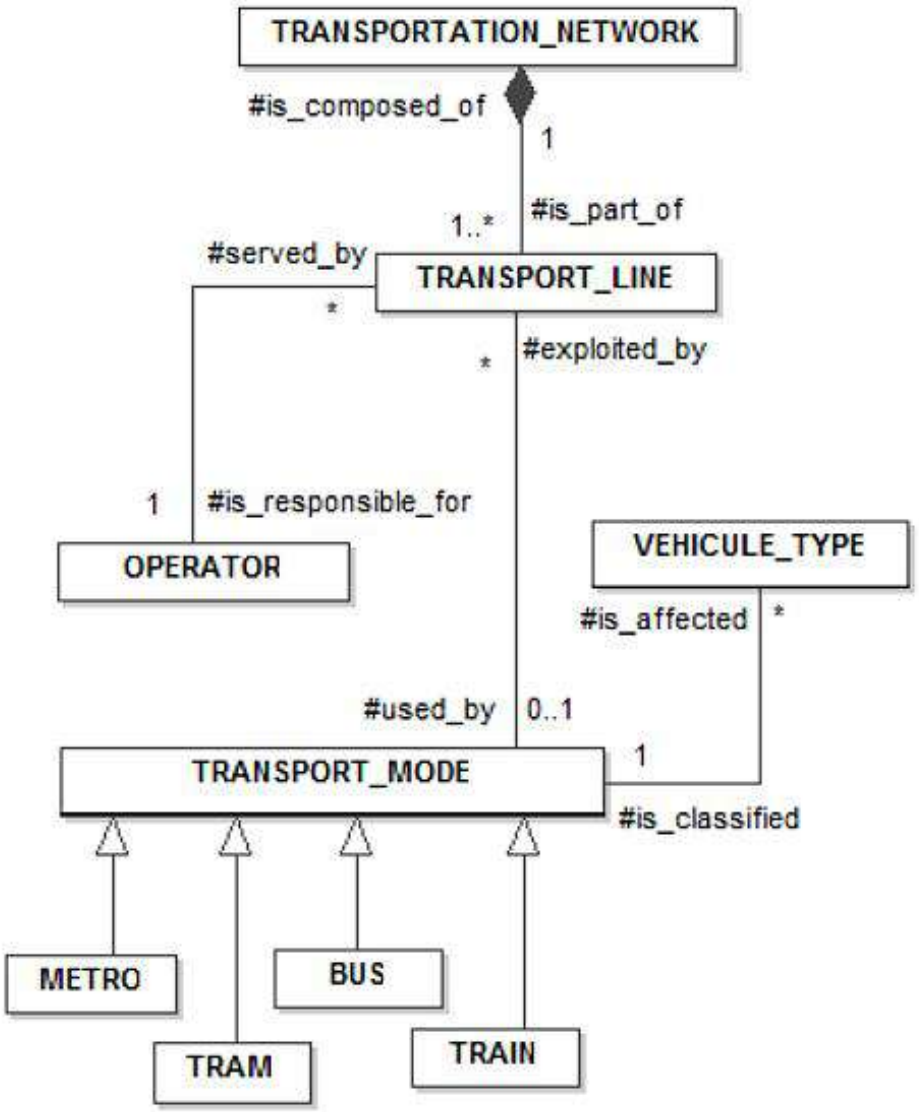


Рис. 5.2. Часть онтологии, связанной с транспортной мультимодальностью

На основе показанной онтологической модели общественного транспорта потенциальным пассажирам предоставляется возможность получения наиболее подходящих для них маршрутов следования из одной географической точки в другую. Маршрут формируется на основе сочетания различных видов транспорта, а также включения в него иной информации об услугах (таких как рестораны, библиотеки и т.д.), которые могут быть доступны на маршруте и полезны для пассажира.

В заключении параграфа рассмотрим ещё один пример использования онтологий. В российских реалиях для целого ряда предприятий, поставляющих свои изделия за рубеж, приобрела актуальность проблема разработки технической документации в соответствии с требованиями международного стандарта *S1000D* [35, 36]. Учитывая широкое распространение *S1000D* в странах НАТО, специалистами международной компании *PTC* (*Parametric Technology Corp.*) разработано решение, позволяющее разрабатывать техническую документацию в соответствии с требованиями *S1000D* на процессы создания, управления и эксплуатации технических публикаций применительно к гражданской и военной авиационной и оборонной промышленности [37].

В основе преобразования документов лежит принцип структурированного модульного представления и хранения технической документации на основе онтологических моделей (рис. 5.3) для документов.

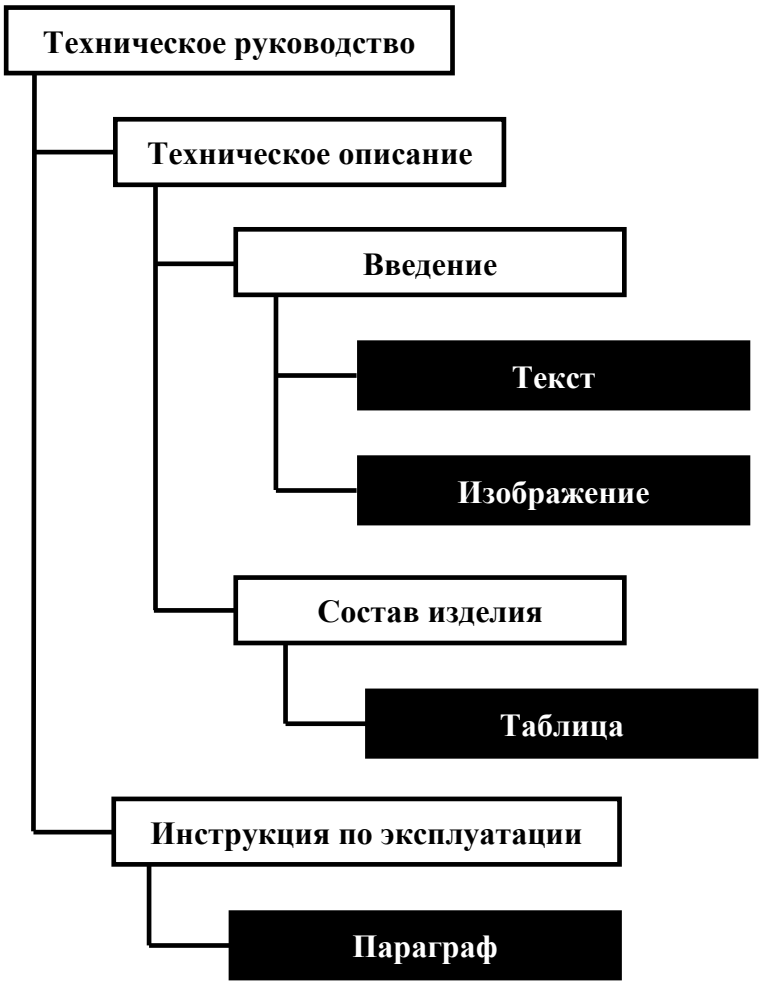


Рис. 5.3. Пример логической структуры базы данных

Важно отметить, что в современных условиях применение онтологического подхода к формированию, сохранению и накоплению знаний на любом предприятии в электронном виде повышает уровень его капитализации. Капитализация – экономический термин, одно из значений которого определяется как увеличение объема собственных средств компании в результате превращения дивидендов, прибавочной стоимости, всей или части прибыли в добавочные объекты производства (оборудование, средства и предметы труда, персонал и т.п.) или в добавочный капитал. С формальной точки зрения суть капитализации заключается в преобразовании будущих доходов в капитал (в данном случае – в интеллектуальный капитал в виде онтологий). При этом капитализированные средства пополняют фонд капиталистического накопления и, тем самым, увеличивают балансовую стоимость компании.

5.2. Интеллект-карты

Интеллектуальная карта (ментальная карта, карта мыслей, диаграмма связей, карта мышления, майндмэп, *mind map*) – это схема, наглядно представляющая главную мысль, её основные элементы и взаимосвязи между ними.

Методика построения интеллект-карт уже давно получила признание в мире как эффективное средство обработки информации. Интеллект-карты помогают не только привести в порядок некие данные, но и стимулировать процесс мышления и поиска решений.

Впервые интеллектуальные карты описал и систематизировал английский писатель и психолог Тони Бьюзен. Он представил этот метод публике в 1974 году после публикации своей книги «Работай головой». Следующей работой Тони была книга «Супер-мышление» («*The mind map book*»), которую он написал вместе со своим братом Барри. В ней они подробно изложили принципы работы головного мозга, теорию метода майндмэппинга и способы его применения. Сегодня для освоения метода работы с интеллектуальными картами доступна переведенная на русский язык и выпущенная в 2019 году книга Тони Бьюзена «Интеллект-карты» [38].

Официальным редактором для создания интеллект-карт является программа *iMindMap,* созданная под руководством изобрета-

теля методики построения интеллект-карт [39]. Важно отметить, что, несмотря на то, что *iMindMap* находится под непосредственным патронажем основателя методики, сегодня она является далеко не единственной в списке работы с ментальными картами: *iMindMap* уступает по функционалу и распространенности, например, редакторам *Mindjet MindManager* и *ConceptDraw Office MindMap*. *iMindMap* кроссплатформенна – есть версии как для настольных систем *Windows, Mac, Linux*, так и для мобильных устройств, а также «облачная» версия. Программа имеет понятный интерфейс и поддерживает русский язык. Неподготовленный пользователь быстро понимает, что представляет собой данная методика.

Методика построения интеллект-карт основана на визуализации и структурировании мышления. А значит, то, как выглядит карта, имеет решающее значение. Любая интеллект-карта – это дерево. Дерево имеет ствол и ветви, отходящие от него. Чем дальше от ствола, тем тоньше становятся ветви – этот простой принцип визуализации позволяет отобразить ход мыслей в правильном порядке (рис. 5.4).

Каждая ветвь – это отдельное направление или мысль, которую вы развиваете. Чем тоньше участок ветви, тем более новым, свежим или детальным он является по отношению к основной мысли.

По умолчанию все основные ветви дерева имеют разные цвета. Это тоже немаловажно и позволяет отделить одну мысль и ход её развития от другой, при этом сохраняя общую структуру. Цвет и форму ветвей можно менять по своему усмотрению.

Стоит отметить, что использование интеллект карт существенно повышает эффективность конспектирования (скорее рисовании) лекций. Так, например, с помощью одной интеллектуальной карты можно представить всё содержание лекции на одном листе. При этом информация оформляется сразу в сжатом и структурированном виде, легко воспринимается, основные понятия и категории сразу оказываются выделенными, конспект содержит минимальное количество текста. С особенностями искусства создания интеллект-карт и возникающими при этом сложностями можно познакомиться в видеопрезентации на странице Т.А. Гавриловой [40].

Рис. 5.4. Пример интеллект-карты, созданной в редакторе *iMindMap*

5.3. Нечёткая логика

При формализации человеческих знаний в различных предметных областях исследователь зачастую сталкивается с необходимостью количественного описания качественных характеристик объектов или отношений между ними. Такая интерпретация подобных параметров неоднозначна, а использование традиционного математического аппарата весьма затруднительно. Примером качественных характеристик могут служить следующие описания:

– возраст: младенческий, детский, юный, молодой, зрелый, преклонный, старческий;

– ветер: слабый, умеренный, сильный, очень сильный;

– интерфейс: дружелюбный, недружелюбный.

В задачах, решаемых интеллектуальными системами, также приходится применять ненадёжные факты, представить которые нельзя двумя противоположными значениями – «истина» или «ложь». Например, «клапан слегка приоткрыт», «осадков не ожидается», «цистерна практически пуста». В этом случае эксперты не в состоянии оперировать чёткими понятиями и используют для решения задач нечёткие представления, которые они подсознательно понимают, но выразить количественно затрудняются.

Неопределённости, подобные описанным, можно разделить на две категории: отсутствие достаточно полного и достоверного знания о предметной области и отсутствие возможности получить исчерпывающую информацию о конкретном состоянии среды, объекте, ситуации и т.п.

Для преодоления проблемы неопределённости знаний в области искусственного интеллекта были разработаны различные теории, в том числе математическая теория нечётких множеств (*fuzzy sets theory*) и нечёткая логика (*fuzzy logic*), которые ведут свое начало с 1965 г., когда профессор калифорнийского университета (г. Беркли) Лотфи Заде опубликовал основополагающую работу «Нечёткие множества» («*Fuzzy Sets*») в журнале «*Information and Control*». Статья заложила основы моделирования интеллектуальной деятельности человека и явилась начальным толчком к развитию новой математической теории. Вторая статья Заде «Тени нечётких множеств» была опубликована годом позже в журнале АН ССР «Проблемы передачи информации».

Первые две статьи по нечётким множествам были закончены Л. Заде в период 1965 г. и практически одновременно переданы им в научные журналы Америки и Советского Союза. Несомненно, Л. Заде стремился к тому, чтобы его новые идеи распространялись по обе стороны «железного занавеса», и советская математическая и кибернетическая общественность имела возможность получить их «из первых рук». Выходец из Азербайджана, увезённый родителями в детстве в Иран, а затем переехавший в США, Л. Заде не утратил связи с Россией. Профессор Л. Заде является прообразом универсального человека будущего, нити судьбы которого связаны с различными культурами и народами.

Лотфи Заде расширил классическое понятие множества, допустив, что функция принадлежности может принимать любые значения в интервале [0;1], а не только значения 0 либо 1. Назвав такие множества нечёткими, он определил ряд операций над ними и предложил обобщение известных методов логического вывода *modus ponens* и *modus tollens*. Введя понятие лингвистической переменной и допустив, что в качестве её значений выступают нечёткие множества, Л. Заде создал аппарат для описания процессов интеллектуальной деятельности, включая нечёткость и неопределенность выражений. Дальнейшие работы профессора и его последователей заложили прочный фундамент новой теории и создали предпосылки для внедрения методов нечёткого управления в инженерную практику.

Сейчас в мире существует ряд крупных научных школ (Беркли (США), Торонто (Канада), Тулуза (Франция), Милан (Италия), Зиген (Германия), Москва, Таганрог (Россия), Рига (Латвия), Баку (Азербайджан) и обществ, культивирующих данное направление. Ежегодно проводятся десятки научных конференций и симпозиумов, выпускаются разнообразные периодические издания научного и прикладного характера.

В теории множеств элемент либо принадлежит множеству, либо нет. Понятия множеств используются во многих математических теориях. Это важное понятие, однако, не рассматривает простые ситуации, когда не все ясно и понятно [41]. Например, при выборе фруктов очень несложно определить общий набор яблок. Однако при этом непросто определить набор спелых яблок. Мы понимаем, что яблоки созревают постепенно... Таким образом, понятие набора спелых яблок является нечетким (расплывчатым).

Понятие нечеткого множества было принято для того, чтобы учесть ситуации, аналогичные представленной выше. Теория нечетких множеств базируется на понятии частичной принадлежности к множеству: каждый элемент принадлежит к нечеткому множеству немного или частично.

Очертание нечеткого множества (рис. 5.5) не имеет «явной» границы, а представляется «нечетким» или «размытым».

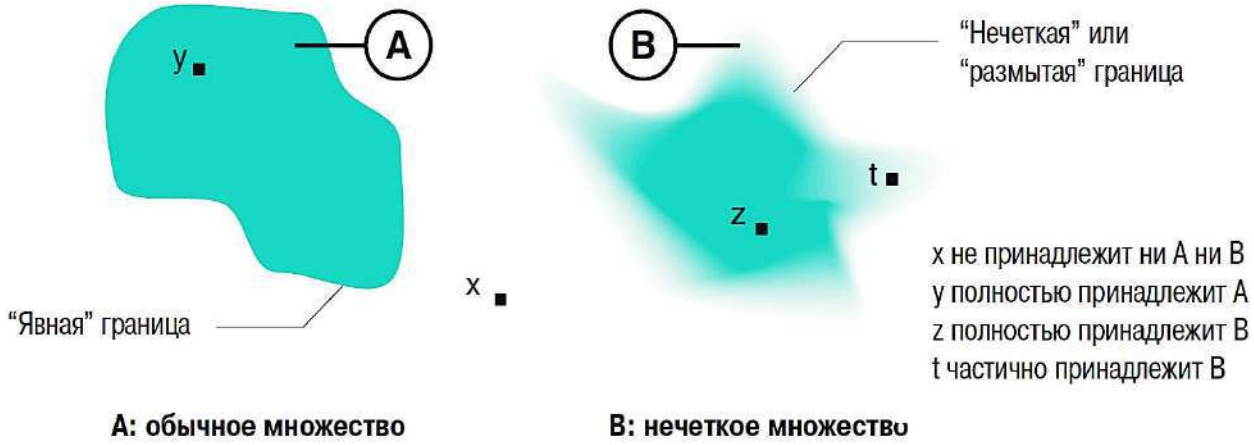


Рис. 5.5. Сравнение обычного и нечеткого множества

Нечеткое множество определяется при помощи «функции принадлежности», которая соответствует понятию «характеристическая функция» в классической логике.

Рис. 5.6. Характеристическая функция

Предположим, мы хотим определить множество людей «среднего роста». В классической логике мы должны определить, что средним ростом мы считаем рост между 1,60 м и 1,80 м. Характеристическая функция в этом случае (рис. 5.6) дает «0» для всех, отличных от заданного диапазона ростов и «1» если рост соответствует заданному. Нечеткое множество людей «среднего роста» за-

дается при помощи «функции принадлежности», которая в этом случае может принимать значения в диапазоне [0;1].

Каждый возможный рост в этом случае будет представлен в виде «функции принадлежности» к нечеткому множеству людей «среднего роста» (рис. 5.7), который принимает значения между 0 и 1.

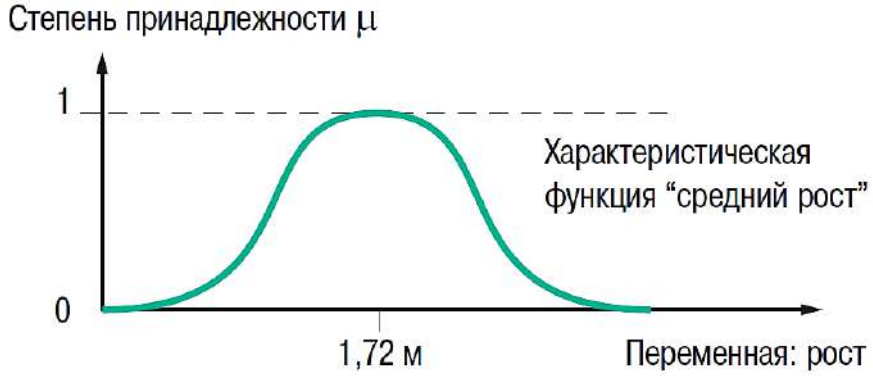


Рис. 5.7. Функция принадлежности

Несколько нечетких множеств можно определить через одну переменную, которая принимает несколько значений, например, множества «маленький рост», «средний рост» и «высокий рост». Каждое понятие описывает функция принадлежности (рис. 5.8).

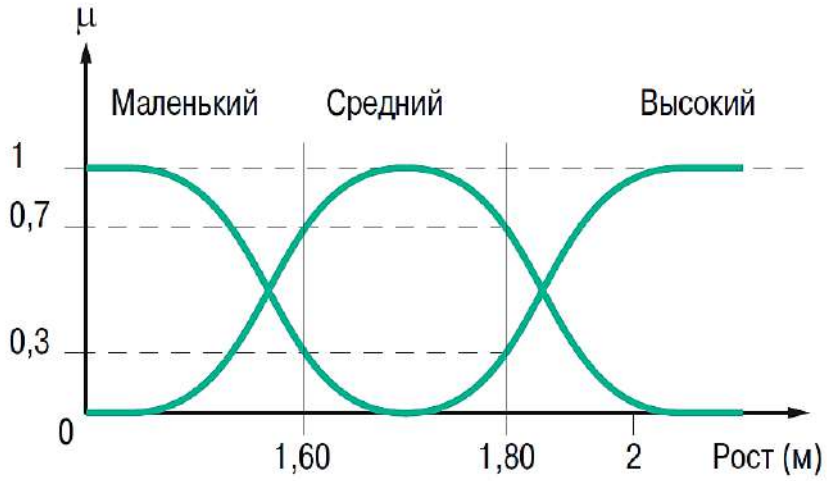


Рис. 5.8. Функция принадлежности, переменная и лингвистическое описание

Этот пример показывает возможность применения нечеткой логики. Человека ростом 1,80 м можно отнести к группе «высокий рост» со степенью принадлежности 0,3 и к группе «средний рост» со степенью принадлежности 0,7. В классической логике переход от среднего роста к высокому мгновенен. Человек с ростом 1,80 м скорее всего принадлежит группе среднего роста, а человек ростом

1,81 м скорее всего высокий. Переменная (рост) и ее описания (средний, высокий), определенные функцией принадлежности, являются лингвистической переменной и лингвистическими описаниями соответственно.

Для решения задач на практике используются базы правил нечеткой логики. Они представляют собой набор правил, которые обычно используются параллельно, но в некоторых приложениях могут быть объединены. Целью базовых правил нечеткой логики является формализация и применение человеческого умозаключения. Таким образом, нечеткая логика является частью искусственного интеллекта.

Процесс обработки нечеткой логики состоит из трех частей (рис. 5.9).

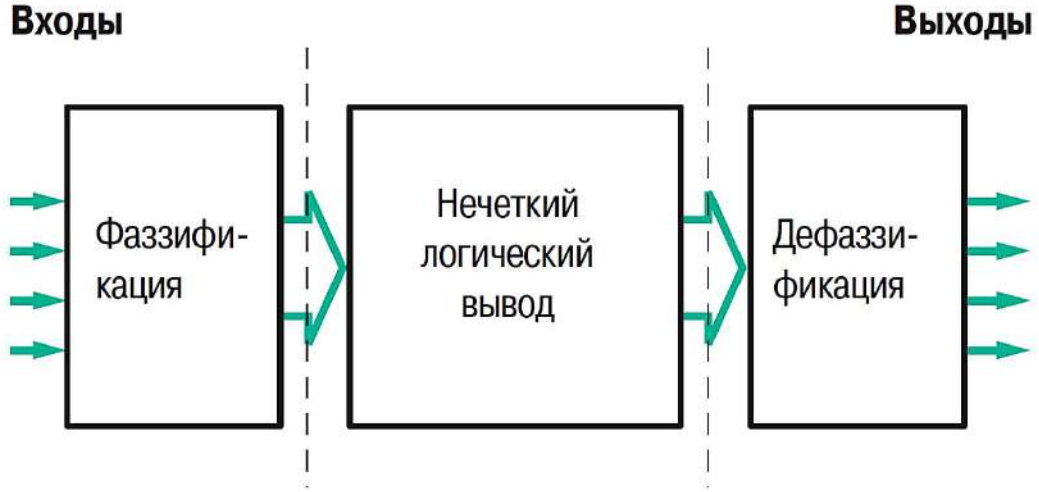


Рис. 5.9. Процесс обработки нечеткой логики

Фаззификация или подготовка задачи для решения методами нечеткой логики позволяет конвертировать реальные значения переменных в нечеткие.

В качестве механизма нечеткого логического вывода чаще всего используется механизм Мамдани [42]. Он представляет собой упрощение более общего механизма и основан на процедуре вывода с применением базы нечетких правил, а также на применении обобщенного правиле дедукции (новое положение выводится в процессе рассуждения от общего к частному).

На последнем этапе нечеткого логического вывода, выходное нечеткое множество уже определено, но оно не может быть напрямую использовано для предоставления оператору точной информа-

ции или для управления исполнительным механизмом. Необходимо выполнить переход из «мира нечеткой логики» в «реальный мир»: этот этап называется дефаззификация.

Большое количество приложений доступно широкой публике, особенно в Японии. Например, компактные портативные цифровые видеокамеры, обладающие повышенной чувствительностью к перемещениям. В данных устройствах нечеткая логика управляет системой стабилизации изображения.

В качестве ещё одного примера использования нечёткой логики приведём пример от автомобильных компаний *Renault* и *Peugeot* (*PSA*). Последние наладили выпуск автоматических коробок передач, в которых применяется нечеткая логика для адаптации к стилю вождения человека, находящегося за рулем, а также подстройки под дорожные условия.

6. ЛУЧШИЕ ПОЛЕЗНЫЕ ДОСТИЖЕНИЯ

За последнее время ИИ развивается такими шагами, что теперь не проходит и месяца без сообщений о прорывах в сфере ИИ. В самых разных областях человеческой деятельности компьютер все чаще начинает превосходить человека.

Однако вспомним о том, что любая технология полезна лишь тогда, когда она вносит существенный вклад в жизнь общества. Помимо этого, такой вклад должен сопровождаться мощным финансовым стимулом, иначе инвесторы не будут ему способствовать. Именно поэтому регулярно возникающие опасения о том, что машины оставят человека без работы, до сих пор не нашли своего практического подтверждения.

Но прогресс не стоит на месте, и многие технологии уже активно используются. Ставку на ИИ с каждым годом делают всё больше и больше игроков рынка – начиная от крупных ИТ-гигантов и заканчивая небольшими стартапами. Стоит отметить, что среди них не только такие известные игроки ИТ-рынка, как *Google*, *Facebook*, *Microsoft*, *IBM*, но и компании, которые, на первый взгляд, должны быть далеки от этой темы. Даже *General Motors* и *Boeing* создали совместную лабораторию, связанную с исследованием ИИ. Иными словами, искусственный интеллект становится своеобразным мейнстримом нашего времени.

В данной главе рассматриваются некоторые лучшие достижения ИИ, которые полезны уже сегодня, достижения ИИ непосредственно за 2020 год и перспективные направления развития ИИ на ближайшее будущее.

6.1. Наиболее яркие достижения разработчиков и создателей ИИ

Возможно, что о технологиях, рассматриваемых в следующих разделах, вы не видели сообщений по телевизору, но их уровень – высочайший. Проведем обзор лишь некоторых, наиболее ярких достижений человека в деле создания решений на основе ИИ.

Одиннадцатого апреля 2018 года Управление по санитарному надзору за качеством пищевых продуктов и медикаментов США (*FDA*) впервые в мировой истории приняло решение, разрешающее

продавать устройство, диагностирующее под управлением ИИ проблемы со здоровьем в клиниках первичной медицинской помощи без участия живого специалиста [43]. Программное обеспечение устройства, называемого *IDx-DR*, использует алгоритм ИИ для анализа изображений сетчатки глаз пациентов. При этом фотографии выполняются с помощью специального аппарата *Topcon NW*400. Далее оператор отправляет цифровое изображение на облачный сервер для обработки, который возвращает результат, указывающий на наличие или отсутствие признаков потери зрения, связанной с диабетом. Прежде чем выдать разрешение, *FDA* оценило результаты клинических испытаний устройства на 900 больных диабетом пациентах из десяти клиник. Система показала точность 87,4% в идентификации пациентов, у которых незначительная степень потери зрения, и точность 89,5% для пациентов с осложнениями, требующими лечения. Очевидно, что такое устройство может быть весьма полезным для людей, проживающих в удалённых районах или районах, где отсутствуют специалисты-офтальмологи.

Одна из новейших разработок ИИ позволяет конкурирующим фармацевтическим компаниям обмениваться информацией, не раскрывая собственных секретов. Разработанная безопасная система ориентирована на побуждение фармацевтических компаний объединить свои ресурсы, создавая большие библиотеки обучающих данных для создания более эффективных решений. Программисты обучили алгоритмы системы предсказывать, с какими белками определённые лекарства будут взаимодействовать в организме человека. Помимо этого, ИИ системы также может использоваться для анализа конфиденциальных медицинских записей в больницах, разработки планов лечения пациентов и составления прогнозов.

Не так давно была разработана специальная программа картографирования лунной поверхности. Она использует алгоритмы ИИ, которые изучили треть поверхности Луны и научились распознавать кратеры. Затем эти алгоритмы ИИ начали тренироваться находить кратеры на другой трети поверхности Луны. Здесь они сумели обнаружить 92% уже известных кратеров, и помимо этого – около 6000 очечных пятен, которые пропустили аналитики-люди. Если эту программу направить на обследование каменистых планет и ледяных лун, она может дать новое представление об истории Солнечной системы.

Аналогичным образом ИИ научился предсказывать, где потенциально могут произойти подземные толчки, с целью помочь людям в районах повышенного риска эффективнее подготовиться к опасным сейсмическим событиям. Программа, которая изучала характеристики более 130000 землетрясений и повторных толчков, может предсказывать места повторных толчков гораздо точнее традиционных методов.

Приведём также отрицательный пример использования ИИ. Он связан с кибербезопасностью, а точнее – с разработкой и использованием дипфейков. Дипфейк – это аудио или видеоконтент, сгенерированный при помощи машинного обучения. Популярный пример: берется лицо политика или артиста/актрисы и формируется компрометирующее видео (рис. 6.1) или просто порнография [44]. Годом пробы пера стал 2019 год, а сегодня создание таких роликов уже доступно широкому кругу лиц. Однако большинство людей пока не выработали иммунитет: мы сначала верим видеоролику, а только потом задумываемся, а настоящий ли он.

Рис. 6.1. Пример кадра из рекламного дипфейка

Продолжим обзор положительных примеров применения ИИ. Так, при фотографировании мегафауны интеллектуальные камеры-ловушки, снимающие животных в их естественной среде обитания, помогают исследователям и специалистам по охране природы отслеживать поведение животных. Эти системы наблюдения за дикой природой делают больше фотографий, чем способен любой человек во время наблюдения. Программа ИИ научилась распознавать ди-

кую природу, изучив почти полтора миллиона изображений с ручной маркировкой, собранных научным проектом *Snapshot Serengeti* [45]. Интеллектуальные алгоритмы фиксирует количество, вид и активность животных на каждом новом изображении.

6.2. Результаты ИИ за последнее десятилетие

Теперь рассмотрим некоторые конкретные результате работы технологий ИИ, полученные за последние годы.

Программные роботы, или, как их называют – «цифровые работники» стали одним из главных технологических трендов 2018 года. Аналитики прогнозируют, что в ближайшие два года количество проектов в области роботизации увеличится на 70%. В рамках стартапа по роботизации бизнес-процессов *RPA* (*Robotic Process Automation*) комапания *UiPath* привлекла инвестиций на сумму 120 миллионов долларов. Впоследствии капитализация компании превысила 1 миллиард долларов. Такой рост определяется тем, что платформу *UiPath RPA* можно внедрить за несколько месяцев, а окупается она за полгода [46]. Программные роботы экономят миллионы часов рабочего времени: наполняют клиентскую базу, обрабатывают несложные финансовые транзакции или отвечают на простые запросы в техническую поддержку. Так, например, «Альфа-банк» доверил роботам обработку платежей, разбор внутренней почты, изменение данных клиента и правки кредитных договоров по заявлениям. Компания планирует экономить на этих процессах по 85 миллионов рублей в год. При этом RPA становятся в разы производительнее, если дополнить их технологиями интеллектуальной обработки информации. На базе таких решений роботы уже определяют тип документа или анализируют смысл текста, извлекают из него важные факты и отправляют их в различные информационные системы. Такие навыки делают роботов полноценными цифровыми помощниками бизнеса.

Не так давно компания *Microsoft* организовала комитет по этике ИИ. Вслед за ней подобные подразделения создали и другие корпорации – *Facebook, Google* и *SAP*. Игрокам есть чего бояться. ИИ не только кардинально меняет нашу жизнь, но и привносит в неё новые вызовы. Вместе с автоматизацией появляется риск, что часть людей потеряет рабочие места. ИИ даёт огромное конкурентное преимуще-

ство крупному бизнесу, а это, в свою очередь, может усилить экономическое неравенство. ИИ зависит от качества и количества данных, которые используются для обучения. Их недостаток приводит к ошибкам или даже предвзятости в работе технологий. Известны примеры, когда роботы-рекрутеры принимали на работу только мужчин, отказывали в кредите людям определённой национальности или даже видели в них потенциальных преступников. В постановке медицинских диагнозов ИИ сегодня тоже недостаточно точен: по некоторым оценкам, машина не ошибается только в 60% случаев. Люди не могут во всем доверять ИИ, и для спорных ситуаций корпорациям и государствам нужно выработать общие принципы взаимодействия с технологиями: не нарушать права человека, повышать прозрачность работы ИИ, соблюдать стандарты безопасности, защищать персональные данные и не вредить.

Искусственный интеллект победил катастрофическую забывчивость. Аналитики компании *Gartner* опубликовали исследование о том, что к концу 2018 года прибыль компаний от ИИ достигла 1,2 триллиона долларов, что на 70% больше, чем в 2017 году. А в ближайшие три года ежегодный рост доходов от ИИ составит 60%. Увеличивается количество сценариев применения технологий. Это заметно и по российскому рынку. Наиболее активны были заказчики проектов с применением ИИ в банках, нефтегазовой сфере и энергетике, появилось больше проектов в промышленности. Банк ВТБ роботизировал открытие счета для юридических лиц, а НПО «Энергомаш», крупнейший производитель ракетных двигателей, использует ИИ для интеллектуального поиска по нескольким миллионам внутренних документов компании.

Искусственный интеллект научился распознавать объекты на фоне среды. Новая нейросеть отделяет распознанные объекты от окружающей их виртуальной среды, чтобы затем «представить» их в другой обстановке. Условный кактус в пустыне она распознает и в комнатном горшке. Система понимает, как объект выглядит под другим углом и освещением. Это ещё одна попытка преодолеть проблему ИИ – катастрофическую забывчивость. Традиционные нейронные сети не способны последовательно учиться новому и не забывать при этом старое. Подобные технологии будут особенно полезны в работе с изображениями: например, позволят лучше распознавать лица людей с разными причёсками или цветом глаз.

Google запустила поиск по открытым наборам данных (датасетам). В выдаче *Dataset Search* указывается информация о ресурсе, на котором опубликован набор данных, авторы, лицензия, дата обновления, описание и доступные для скачивания форматы. Тематика наборов не ограничена. Конечно, это не первая подобная инициатива: порталы с открытыми данными есть у многих городов, государственных и научных организаций. Но возможность искать такие наборы и найти нужный может упростить жизнь разработчикам технологий. Данные, особенно по специфической отраслевой теме, стоят дорого, их сложно раздобыть, к тому же они быстро устаревают. Возможность использовать открытые датасеты позволит удешевить и ускорить разработку технологий, особенно если речь идёт о стартапах.

Команда ботов *OpenAI* впервые проиграла людям в чемпионате по *Dota* 2, что удивительно, ведь в последнее время алгоритмы все чаще побеждают человека в различных играх: шахматы, Го и другие. А ещё год назад алгоритм, созданный компанией *OpenAI*, выиграл у человека в ту же *Dota* 2 один на один. Относительно недавнее сражение года показало, что машины по-настоящему сильны в индивидуальном зачёте, а вот работа в команде, коммуникация, распределение обязанностей и работа в изменчивых условиях – не самые сильные стороны ИИ. С одной стороны, это яркий пример того, какие профессиональные навыки будут востребованы у людей в ближайшем будущем. С другой стороны, индивидуализм характерен для человека, а это значит, что технологии больше становятся похожими на нас самих.

Человекоподобные роботы *Boston Dynamics* научились бегать и перепрыгивать через препятствия. Теперь они обрабатывают видео в реальном времени, а специальная программа помогает балансировать конечностям и торсу машины. За последние пять лет робот научился ходить по снегу, стоять на одной ноге и делать сальто. ИИ помогает роботам лучше ориентироваться в пространстве и работать в необычных, иногда даже экстремальных ситуациях. В ближайшие несколько лет подобных роботов будут активно использовать в условиях, где человеку опасно находиться: при производстве автомобилей, в металлургии и химической промышленности, а ещё для спасения людей при чрезвычайных ситуациях.

Искусственный интеллект научился прогнозировать болезнь Альцгеймера на ранних стадиях: распознавать изменения в участках нервной ткани, вызванные обменом веществ в определённых отделах мозга. В отличие от томографии, алгоритмы ИИ способны определить симптомы заболевания на шесть лет раньше. С его помощью у врачей появится возможность замедлять или вообще останавливать слабоумие.

6.3. Решения и проекты ИИ 2020 года

Прошедший год отметился множеством разработок, использующих технологии машинного обучения и системы ИИ на базе нейронных сетей. В 2020 году ИИ научился не только нюхать, писать музыку, стихи и картины, но и просто читать мысли человека.

Первым примером интенсивного развития технологий ИИ в 2020 году является созданный специалистами *Intel Labs* и Корнелльского университета программно-аппаратный ИИ-комплекс, способный различать запахи и имитировать работу обонятельной нервной системы человека. В основу разработки легли нейроморфные процессоры *Intel Loihi* (рис. 6.2), сочетающие процессы обучения, тренировки и принятия решений в одном чипе и позволяющие системе быть автономной и «сообразительной» без подключения к облаку (к базе данных).

Рис. 6.2. Нейроморфный процессор *Intel Loihi*

В ходе экспериментов спроектированный и оснащённый химическими датчиками комплекс продемонстрировал высокую эффективность при распознавании в воздухе запахов опасных веществ даже в условиях сильных помех. Подобного рода решения, уверены в *Intel*, помогут в развитии робототехники, когда роботы смогут сами сортировать продукты, ориентируясь на запах, подтолкнут развитие систем слежения за состоянием окружающей среды, приведут к повышению безопасности труда на производстве и в целом дадут толчок к развитию когнитивных способностей кремниевых процессоров.

В конце ноября 2020 года компания *DeepMind*, входящая в состав исследовательских подразделений *Google*, объявила о решении фундаментальной научной проблемы, которая была впервые сформулирована около полувека назад – по меркам современной биологии, почти в доисторические времена. Проблема, решению которой были посвящены многотомные монографии и работа целых институтов формулируется так: «Как предсказать трехмерную структуру любого белка по его аминокислотной последовательности, где эта структура однозначно закодирована?». Много лет среди ученых-биологов считалось, что её решение откроет для человечества невиданные перспективы в биотехнологиях и даст поток новых лекарств, разрабатывать которые станет возможно прямо на компьютере, не прикасаясь к пипеткам. Сейчас, когда решение знаменитой проблемы неожиданно пришло из недр технологической компании, создавшей для этого систему ИИ *AlphaFold* [47], эксперты из академического мира встречают его со смесью восторга и недоверия, а иногда и скепсиса. А *DeepMind* тем временем перешла к решению вопроса о том, как лучше всего обеспечить более широкий доступ к системе *AlphaFold* с возможностью её масштабирования.

На шаг вперёд в направлении чтения человеческих мыслей удалось продвинуться группе учёных из Калифорнийского университета в Сан-Франциско, экспериментально доказавшим возможность распознавания нервных сигналов в головном мозге человека и их трансляции в понятные слова с помощью рекуррентной нейросети и вживлённых в мозг электродов. В эксперименте приняли участие пациенты с эпилепсией, электроды которым были вживлены для борьбы с неврологическим заболеванием и отслеживания приступов. Так получилось, что часть электродов оказались в зонах мозга, в ко-

торых происходит подбор слов, составление выражений и осуществляется обратная связь с участками мозга, воспринимающими собственную речь человека. Испытуемым было предложено мысленно, а затем вслух произнести несколько предложений с ограниченным набором слов. Одновременно снимались сигналы с имплантированных в мозг датчиков. Полученные данные были переданы в нейронную сеть для обучения, а промежуточный результат был отдан для анализа другой искусственной нейронной сети. Вероятность ошибочного определения слов составила всего 3 процента.

Ещё одно интересное применение ИИ нашли программисты компании *OpenAI,* разработавшие программу ИИ на основе нейронной сети *Jukebox*, сочиняющую музыку с осмысленными текстами и вокалом. Для обучения нейронной сети системы было использовано множество отрывков из песен самых разных жанров, от рока, джаза и блюза до хип-хопа с кантри и классическими произведениями. Такой подход позволил команде *OpenAI* расширить возможности проекта и добиться эффекта имитации музыкальных композиций исполнителей, на треках которых он обучался. Например, *Jukebox* может сочинить музыку в стиле кантри-певца Джонни Кэша, рэпера *Drake* и даже российской поп-группы «Тату». На создание одной минуты музыкального трека с вокальными партиями искусственному интеллекту требуется около 9 часов и огромный объём вычислительных ресурсов. Именно по этой причине компания не может предоставить открытый доступ к своей системе ИИ. Зато разработчики опубликовали результаты работы *Jukebox*. Послушать их можно на сайте прослушивания примеров (рис. 6.3) [48].

В середине 2020 года стало известно о создании специалистами Лаборатории искусственного ИИ Массачусетского технологического института системы машинного обучения *Timecraft*, позволяющей воссоздать процесс написания картин и нанесения мазков для произведений живописи известных художников, будь то Моне, Винсент Ван Гог или Сальвадор Дали. Сначала модель натренировали на двух сотнях видеороликов с записью акварельной и цифровой техник, а потом на её основе создали сверточную нейронную сеть, которая владеет знаниями о том, как именно была написана картина, и даже может имитировать движения кисти художника [49]. В результате система *Timecraft* сумела показать более высокую эффективность, чем существующие подобные проекты, более чем в 90 % случаев.

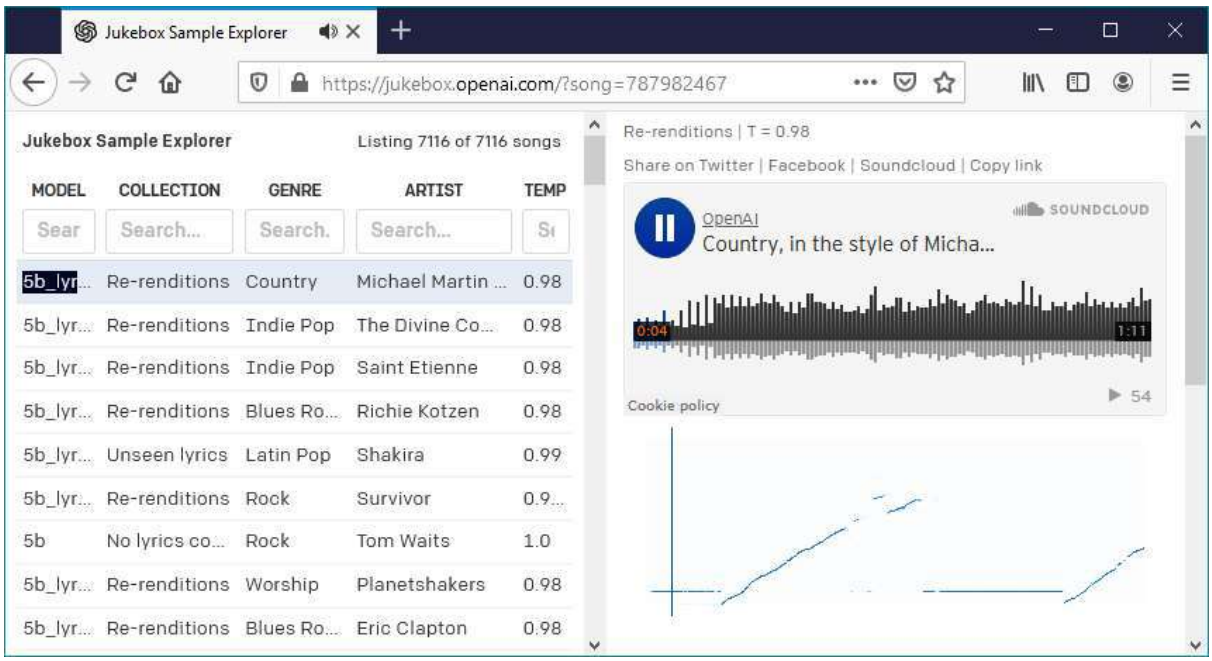


Рис. 6.3. Список композиций ИИ, доступных для прослушивания

Помимо этого, в 2020 году виртуальную галерею нейросетевого искусства, в которой представлены четыре тысячи уникальных картин, созданных ИИ, открыла российская компания «Яндекс». Галерея размещена на соответствующем сайте в Интернет [50] и разделена на четыре тематических зала: «Люди», «Природа», «Город» и «Настроение». Для обучения нейросети специалистами «Яндекс» были использованы произведения, принадлежащие к разным направлениям живописи: от фовизма и кубизма до минимализма и стрит-арта. В процессе обучения система ИИ изучила 40 тысяч картин, после чего взялась за создание собственных произведений. Для отбора картин по разным категориям использовалась другая нейросеть, которая применяется в сервисе «Яндекс.Картинки» для поиска изображений по запросам. Именно она смогла увидеть на картинах людей, природу, город и разные настроения, отсортировав имеющиеся в наличии работы по категориям.

Ещё одну разработку в области ИИ подготовила *Mail.ru Group*. Она представила в 2020 году платформу [51], которая позволяет в несколько кликов создавать новостные и репортажные видео студийного качества. Чтобы создать видео, достаточно загрузить в систему текст новости – и виртуальный ведущий его зачитает. Дикторы выглядят и разговаривают, как живые люди: при чтении новостей они реалистично воспроизводят мимику, эмоционально реаги-

руют и расставляют смысловые акценты. Внешность диктора выбирает пользователь: в компании создали несколько моделей цифровых ведущих, прототипами для которых послужили реальные люди (рис.6.4).

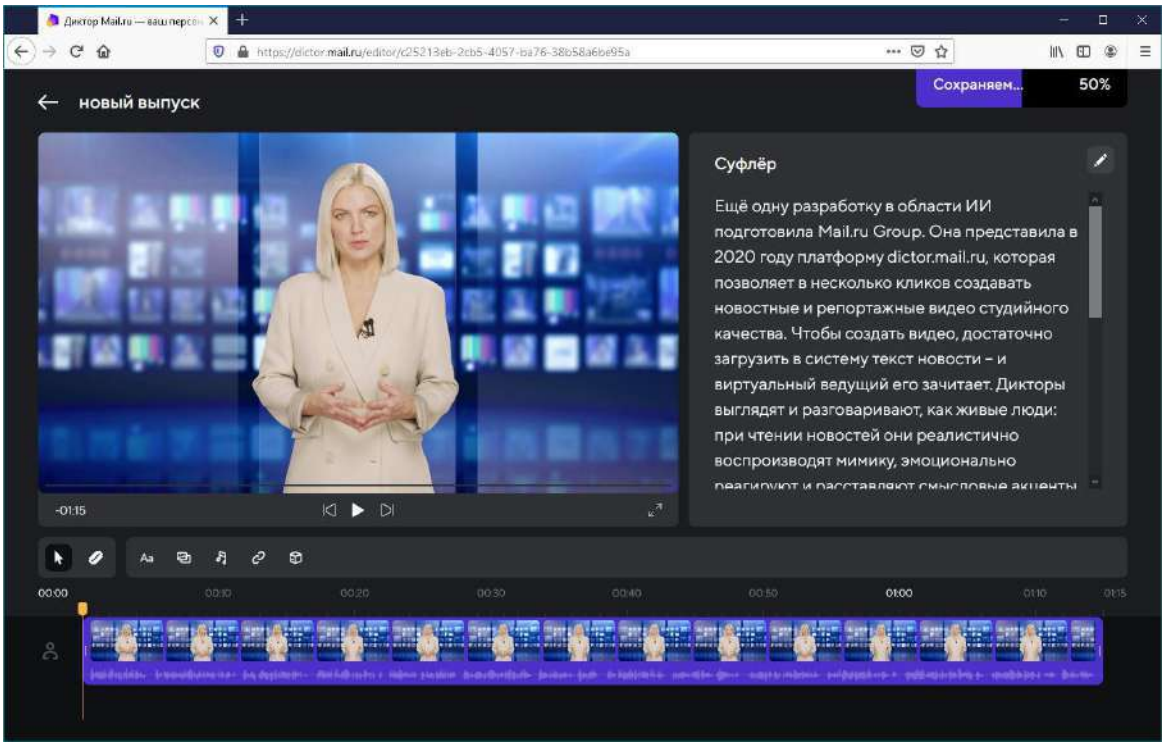


Рис. 6.4. Новостной выпуск виртуального диктора, читающего данный абзац

В *Mail.ru Group* подчёркивают, что при создании виртуальных ведущих использовались методы машинного обучения. Синтез речи построен на основе наработок голосового ассистента Маруси. А видеоизображение синхронизируется с речью в режиме реального времени с помощью системы компьютерного зрения *Vision* от той же *Mail.ru Group*, обученной на реальных прототипах и видеозаписях.

6.4. Что будет в ближайшие 10 лет?

По мнению ряда ведущих учёных, в ближайшее десятилетие в области ИИ возможны следующие достижения [52].

1.Создание интеллектуальных медико-диагностических систем, превосходящих по точности постановки диагнозов естественных врачей. Выявление с помощью интеллектуальных систем новых неизвестных ранее медицинских знаний и использование этих знаний для улучшения качества медицинской практики.

Одна из таких интеллектуальных систем – «Система диагностики и прогнозирования заболеваний сердечно-сосудистой системы» размещена на сайте *KardioNet* [52]. С помощью нее выявлены новые неизвестные ранее знания, которые используются для совершенствования существующей практики лечения и профилактики заболеваний сердца.

2. Создание интеллектуальных систем диагностики сложных технических устройств, превосходящих по своим возможностям и точности известные инженерные методики. Их внедрение в инженерную практику позволит повысить надежность технических устройств и, таким образом, способствовать решению проблемы снижения техногенных аварий и катастроф – актуальной проблемы XXI в.

Имеющийся отечественный опыт разработки нейросетевых систем диагностики авиационных двигателей показал, что интеллектуальные системы позволяют выявлять такие дефекты, которые обычными инженерными методами выявлены быть не могут.

3. Создание интеллектуальных систем, способных получать точные аналитические решения краевых задач, что позволит преодолеть современный кризис прикладной математики. Реально внедрение таких интеллектуальных систем в инженерную практику, что также будет способствовать снижению числа техногенных аварий и катастроф.

С сайта Пермского отделения Научного Совета Российской Академии Наук [54] можно обратиться к демонстрационному прототипу интеллектуальной системы «Искусственный математик», способной получать точные аналитические решения краевых задач теории упругости, теплопроводности, термоупругости и др.

4. Создание интеллектуальных систем, прогнозирующих экономическое состояние предприятий, позволяющих разрабатывать рекомендации по оптимизации их деятельности.

В научной литературе уже описан опыт создания нейросетевой системы прогнозирования вероятности банкротства российских банков. Помимо прогнозов система позволяет получать полезные рекомендации по предотвращению банкротства конкретных банков.

5. Создание интеллектуальных систем, предназначенных для прогнозирования развития политических событий и влияния на эти события. Профессор кафедры прикладной математики и информатики пермского университета Л.Н. Ясницкий в своих работах опи-

сал случай успешного прогнозирования победы Д. Медведева на президентских выборах 2008 года, выполненного за полтора года до этих выборов. Он также привёл подтвердившиеся впоследствии результаты прогнозирования рейтингов известных политических деятелей и рекомендации по улучшению этих рейтингов.

6. Создание и применение интеллектуальных систем в области криминалистики. Всё тот же автор в более поздних работах описал опыт создания нейросетевого детектора лжи, значительно превосходящего известные штатные аналоги по точности заключений и изложил опыт создания интеллектуальной системы, предназначенной для изучения личности и выявления серийных убийц.

7. Реально создание интеллектуальных систем, предназначенных для прогнозирования результатов спортивных состязаний и для оптимизации программы подготовки спортсменов с целью получения ими наивысших спортивных результатов.

В практике нейронные сети уже применялись для прогнозирования результатов олимпийских игр 2014 года, чемпионата мира по легкой атлетике 2015 года, чемпионата Европы по футболу 2016 года, а также при разработке рекомендаций для известных спортсменов: фигуриста Е. Плющенко и ряда спортсменов-бегунов.

8. Реально создание и широкое применение интеллектуальных систем в психологии.

В заключение следует отметить, что, несмотря на радужные перспективы развития ИИ, обусловленные результатами современных исследований, его внедрение в жизнь социума предполагает решение довольно непростых организационных задач. По мнению авторов, решение этих задач должно, прежде всего, опираться на первоначальное внедрение ИИ в сферы промышленности, где труд без использования методов ИИ влечет большой риск здоровью человека и где невозможно решение производственных задач без ИИ. Если для густонаселенных государств внедрение в жизнь социума ИИ не имеет возражений, то для стран с малой плотностью населения и огромными расстояниями между населенными пунктами повсеместное использование ИИ представляется проблематичным.

7. ИНТЕЛЛЕКТУАЛЬНЫЕ ИНСТРУМЕНТЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

В интеллектуальной эволюции общего управления проектами сегодня можно выделить четыре этапа. Начальный этап, для которого характерно первичное решение задач по автоматизации процессов проектного управления. Этап появления виртуальных помощников, позволивших автоматизировать процедуры распознавания речи и текста, организации встреч и контроля прогрессов, подготовки отчетов и актуализации данных в системах. Этап интеллектуальных систем управления проектами, предоставивший такие новые инструменты, как прогнозная аналитика проекта, оптимизация планирования, предупреждения о рисках и подбор оптимальной команды. И, наконец, этап ближайшего будущего, который можно охарактеризовать формированием гибридных команд типа «люди + роботы», созданием новых типов ИИ-проектов, появлением новых компетенций и, возможно, инструментов автономного управления проектами.

Сейчас мы находимся на стыке второго и третьего этапов. Поэтому в рамках этой главы попытаемся разобраться с плюсами и минусами будущего ИИ в управлении проектами (в большей степени относящихся к области программной инженерии и реализации проектов программного обеспечения (ПО)).

7.1. ИИ в управлении проектами программного обеспечения

Одним из современных трендов повышения эффективности процессов разработки (*Development*) и эксплуатации (*Operation*) проектов ПО стала методология (иногда говорят «набор практик»), получившая устоявшееся название *DevOps* (*DEVelopment OPeration*). Повышение эффективности в рамках *DevOps* достигается за счет непрерывной интеграции и активного взаимодействия профильных специалистов – разработчиков, тестировщиков и эксплуатационников на основе инструментов автоматизации (рис. 7.1). *DevOps* распределяет ответственность на каждого участника команды путем коммуникаций и взаимодействий между ними, что позволяет избежать внутренних конфликтов в организации. Сего-

дня данная методология уже стала применяться для реализации проектов в различных предметных областях и позволяет выполнять проекты быстрее и качественнее, а главное – обеспечивать более качественный сервис.

Рис. 7.1. Иллюстрация, демонстрирующая вариант *DevOps*, как пересечения разработки, эксплуатации и тестирования

DevOps-инженер – это специалист, который занимается внедрением методологии *DevOps* в проекты. Он должен обладать знаниями из разных областей, он и разработчик, и менеджер, и тестировщик и специалист технической поддержки [55].

Команда *DevOps* на предприятии, как правило, отвечает за управление проектами. Помимо прочего сотрудники команды во многом полагаются на анализ информации, но если раньше они задействовали для этих целей специализированные инструменты, то в последнее время наметилась тенденция к применению ИИ, который может проводить аналитические расчеты без участия человека.

Крупный аналитический портал *InformationWeek*, посвящённый информационным технологиям для бизнеса, в 2019 году выполнил анализ плюсов и минусов использования инструментов ИИ в управлении проектами для сферы информационных технологий (ИТ) и предоставил пользователям следующую информацию.

Первый плюс ИИ заключается в том, что он идеально подходит для обработки больших объемов данных. ИИ превосходит людей во многих сферах применения, но он особенно эффективен в обработке огромных объемов данных. Рынок ИТ знает немало примеров, когда ИИ помогал *DevOps*-командам сократить до нескольких дней

расчеты, на проведение которых в противном случае потребовались бы месяцы. Анализ данных с привлечением ИИ и машинного обучения поможет руководителям отточить планы своих проектов с учетом прошлых результатов. Он позволяет выявлять риски, которые могут угрожать успешной реализации проектов. Однако ИИ ни при каких обстоятельствах не должен заменять фактического менеджера проекта, а выступать в роли доверенного помощника. Применять ИИ целесообразно, если *DevOps*-команда работает над долгосрочным проектом или же ей приходится сталкиваться с повышенным объемом данных. В этом случае ПО на базе ИИ может применять для анализа информации, получения выводов и принятия решений. Не исключено, что в ходе анализа ИИ сможет извлечь из данных неочевидные, но сулящие выгоду решения. Менеджерам проекта желательно регулярно формулировать вопросы, на которые ИИ в состоянии ответить. Для этого нужно соответствующим образом настроить программный интерфейс ИИ. Если процедура настройки будет выполнена некорректно, анализ данных вряд ли увенчается успехом.

Второй плюс – ИИ помогает снизить стоимость проекта. Руководители по управлению проектами не понаслышке знают, как не просто проконтролировать, чтобы их команды не выходили за рамки бюджета. Исследования показывают, что внедрение ИИ-интерфейсов для управления проектами помогает управляемо контролировать затраты. По данным международной консалтинговой компании *Accenture*, административные задачи занимают 54% рабочего времени руководителя проекта, но, как считают аналитики компании, ИИ вдвое сократит эту цифру. ИИ умеет визуализировать данные, освещая в ходе реализации проекта узкие места, которые непросто обнаружить без комплексной оценки. Иной способ его применения: ввод данных для вычисления средних затрат времени на выполнение отдельных частей проекта по командам и вычисление приблизительного времени, которое потребуется на завершение проекта. Обычно перерасход бюджета возникает тогда, когда его меньше всего ждут и он связан с проблемами административного характера. Внедрение ИИ подскажет руководителям проектов способы минимизации проблем, прежде чем они выйдут из-под контроля.

Первый минус ИИ – это то, что его приложения для управления проектами находятся на ранних стадиях разработки. Руководители предприятий далеко не всегда внимают советам или рекомендациям ИТ-профессионалов по инвестициям в те или иные программные решения. В нашем случае они объясняют свой отказ незрелостью инструментов ИИ для управления проектами, которые уступают традиционным приложениям.

Тем временем новозеландская фирма *Psoda* разработала приложение для синхронизации физических и цифровых Канбан-досок [56]. ИИ-приложение *PsodaVision*, работающее на базе технологии машинного зрения, умеет запоминать изображения на физических картах Канбан и затем воспроизводить их цифровую проекцию. *Psoda* утверждает, что ее инструменты для управления проектами используются в крупнейших проектах, которые оцениваются в 10 и более миллиардов долларов. В настоящее время *DevOps*-командам нужно определиться с выбором: или встать на путь тестировщиков новых ИИ-инструментов для управления проектами, или дождаться того времени, когда они достигнут стадии готовности и их работу не будет сопровождать череда непредвиденных ошибок. Одновременно с этим нужно отметить растущую тенденцию применения ИИ в бизнес-среде. По данным консалтинговой компании *Gartner*, за последние четыре года спрос на него повысился на 270%.

Второй минус ИИ состоит в том, что если он использует неподготовленные данные, то это всегда приводит к ошибочным выводам. Генеральный директор *DevOps Institute* Джейн Гролл считает: «Всецело полагаться на принятие решений ИИ не стоит, иначе это приведет к искажению действительности и снижению уровня критического мышления, который (по крайней мере, на сегодняшний день) уникален для людей. Управление проектами – это не только искусство, но и опыт, поэтому в нем должны быть выдержаны пропорции между быстрой и умной аналитикой, которую может предоставить ИИ, и наставничеством менеджеров по проектам и продуктам». Говоря об ИИ, многие часто упускают из виду такой важный аспект, как подготовка данных. Компаниям, которые рассматривают возможность внедрения ИИ для управления проектами, следует иметь в виду, что им придется потратить немало времени на очистку и обучение данных. Наиболее трудоемкой частью обучения является подготовка данных для работы алгоритмов. Проигнорировав этап обу-

чения данных, компания рискует заполучить сырую систему, которая с высокой степенью вероятности будет выдавать неполные или не заслуживающие доверия результаты. Примечательно, что некачественно очищенные данные могут привести к сбоям даже самых дорогих и продвинутых систем ИИ для управления проектами.

Главное, что отмечают аналитики *InformationWeek* – управление проектами сегодня не может обходиться без участия человека. ИИ может упростить выполнение некоторых постоянно повторяющихся процессов и администрирование. Подобные задачи отвлекают руководителей проектов от решения стратегических вопросов. Тем не менее, применение ИИ для управления проектами – это пока что относительно новое начинание, и оно несет определенные риски. Нужно учитывать, что даже наиболее качественно обученные алгоритмы в ИИ-решениях для управления проектами не гарантируют их бесшовную работу, поэтому сотрудникам нужно регулярно следить за данными, которые «потребляет» ИИ, чтобы предотвратить возможное искажение результатов.

Однако ряд виртуальных помощников и инструментов ИИ сегодня уже прочно вошли в деятельность руководителей проектов. Рассмотрим наиболее популярные среди них.

7.2. Виртуальные помощники и интеллектуальные инструменты

Виртуальным помощником (или ассистентом) принято считать программного агента (бота), который может выполнять задачи для пользователя на основе информации, введенной пользователем, данных о его местонахождении, а также информации, полученной из различных Интернет-ресурсов. Типовой функционал виртуальных помощников определяется следующим образом:

- встраивание в мессенджеры или информационные системы;
- распознавание команд, текста и речи;
- ответы на типовые вопросы;
- контроль исполнения задач;
- подготовка отчетов;
- организация встреч и собраний;
- напоминания о мероприятиях и событиях проекта;
- актуализация данных в системах управления проектами.

Наиболее популярными виртуальными помощниками руководителей проектов сегодня являются следующие разработки.

1. Бот Дина. Является отечественным решением, подключается через мессенджеры, включая *WhatsApp* и *Telegram* (рис. 7.2). Дина может оказывать помощь в контроле исполнения задач проекта, проводить опросы и стендапы исполнителей, подготавливать отчёты для руководителя проекта [57].

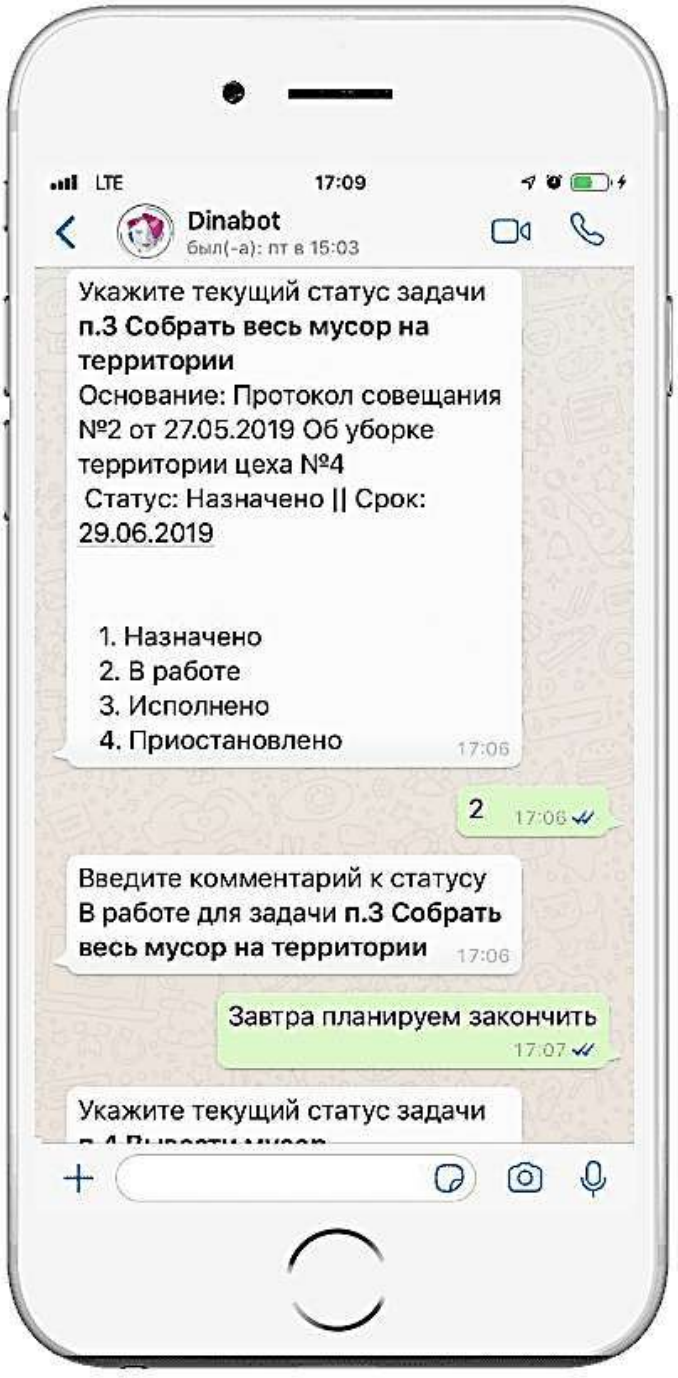


Рис. 7.2. Пример использования бота Дина

2. *PMOtto.ai* Представляет собой сервис персонального виртуального помощника руководителя проекта, объединяющий в себе функции чат-бота и интерфейс взаимодействия с системами управления проектами и портфелями проектов. Может давать рекомендации по реализации проекта [58].

3. *Stratejos*. Является помощником для проектных команд, использующих корпоративный мессенджер *Slack* и системы управления проектами типа *Jira* или *Trello*. Позволяет автоматически напоминать о таких событиях, как отставание от графика, необходимость оценки задачи [59] и др.

4. *Lili.ai* Французский стартап, созданный с целью внедрения методов ИИ для оптимизации бюджетов проектов и повышения эффективности проектного управления. Проект представляет Лили – виртуального помощника по проектному управлению. Лили ведет в Интернет профессиональный блог [60].

5. Чат-бот «Иван из Проектной ПРАКТИКИ». Эта разработка Центра дистанционного обучения Группы компаний «Проектная ПРАКТИКА» – прототип виртуального помощника руководителя проекта. Работает через *Telegram* [61].

6. *Autodesk Construction IQ*. Этот интеллектуальный помощник предназначен для проектов строительства, в которых применяется платформа *Autodesk BIM* 360. Использующий методы машинного обучения, *Construction IQ* собирает и анализирует данные о качестве и безопасности строительных объектов, возможных рисках проекта. Результаты анализа рисков предоставляются пользователям в *Project Home* – едином окне, отображающем ключевую информацию о проекте, включая прогресс по работам, интерактивную модель объекта, данные с камер [62] и т.п.

7. Битрикс24. Решение Битрикс24 позволяет управлять проектами с помощью программных роботов. Они могут создать новые задачи проекта, назначать или менять ответственных исполнителей, актуализировать статус исполнения задач, отвечать на необходимые письма и выполнять прочие поручения. На платформе Битрикс24 также был разработан Битрикс24.Ассистент [63], который позволяет работать с популярными голосовыми помощниками (Яндекс Алиса, Google Assistant). Битрикс24.Ассисетнт может раздавать задания сотрудникам и коллегам, назначать встречи в календа-

ре, общаться в Битрикс24.Чат. В 2020 году на платформе Битрикс24 был разработан ассистент от Сбербанка [64]

Помимо указанных решений следует также упомянуть платформу *Azure Bot Service*. Она является одной из наиболее продвинутых платформ для создания виртуальных помощников руководителей проектов и других чат-ботов и предоставляет исчерпывающий инструментарий для создания решений корпоративного уровня.

Если же речь заходит об инструментах ИИ в управлении проектами, то, как правило, речь идёт уже о более сложных, чем виртуальные помощники, системах, включающих в себя более широкий набор функций. Рассмотрим их.

1. Отечественная интеллектуальная система управления проектами *Smart Projects*. Представляет собой семейство продуктов, обеспечивающих поддержку полного цикла управления проектами. *Smart Projects* использует такие технологии, как мультиагентные системы, онтологии (базы знаний) и сетецентрический подход для построения сложных систем планирования и управления [65].

2. Система *Aurora* ориентирована на создание оптимальных календарно-сетевых графиков проектов. Она использует технологии ИИ для сохранения и применения правил и знаний планирования. Пользователи могут расширять знания системы правилами, специфичными для конкретной предметной области. Первоначально система разрабатывалась для *NASA*, с целью помощи в решении критичных задач планирования со сложными ограничениями. В последующем *Aurora* нашла широкое применение и в других организациях. Она особенно эффективна применительно к крупным проектам со сложными ограничениями и требованиями к ресурсам [66].

3. Интеллектуальный динамический планировщик *Liquid Planner* автоматически корректирует ожидаемые даты завершения задач проекта, в случае изменения ожиданий, а также, когда ресурсы перенаправляются на другие проекты или когда другие работы получают более высокий приоритет [67].

4. Платформа искусственного интеллекта *Infosys Nia Contracts Analysis* для автоматизации бизнес-процессов и ИТ-процессов. Позволяет клиентам использовать технологии ИИ для самого широкого круга задач. Входящий в ее состав модуль *Nia Contracts Analysis* использует машинное обучение и семантическое модели-

рование для помощи в управлении контрактами, ускорении проверки контрактов и снижения рисков их заключения. Применение такого интеллектуального приложения становится особенно эффективно в случае сложных проектных договоров, состоящих из десятков и даже сотен страниц с множественными оговорками, различными ценовыми схемами и условиями поставки, наличием дополнительных соглашений, которые могут изменяться с течением времени [68].

5. Приложение *Cloverleaf*, помогающее руководителю проекта сформировать оптимальную команду проекта или развить людей для результативной совместной работы и успешной реализации проекта. *Cloverleaf* сопоставляет личные качества и навыки участников команды проекта, чтобы помочь принять правильные решения по ее работе, дополняя функции *HR*-партнера [69].

6. Интеллектуальный инструмент *PineStem*, ориентированный на проекты по разработке ПО, помогающий руководителю проекта сформировать оптимальную команду программистов. *PineStem* подсказывает, какие сотрудники лучше всего подходят для конкретного проекта, основываясь на прошлом опыте их работы. Эффективность использования *PineStem* возрастает по мере того, как все больше проектов и задач записываются в его базу данных. Основываясь на анализе текущих данных и прошлом опыте, *PineStem* определяет производительность команды и возможное превышение сроков, что дает руководителю проекта возможность своевременно принять корректирующие меры, чтобы вернуть проект в нужное русло [70].

7. Платформа ИИ *TARA AI*, использующая машинное обучение, ориентирована помогать организациям и руководителям проектов в планировании и реализации проектов. Система предлагает варианты реализации проекта в рамках выделенного бюджета, помогает с подбором членов команды проекта. В ходе реализации оказывает содействие в перепланировании проекта, например, в случае изменения функционального объема [70].

Приведенный перечень примеров далеко не исчерпывает все возрастающее влияние ИИ на проектное управление. Между тем, вопрос о замене руководителя проекта ИИ едва ли может быть актуальным. ИИ высвобождает время от рутинных и технических задач. Позволяет руководителю проекта сконцентрироваться на

сложно формализуемых областях, например, управлении коммуникациями и ожиданиями, разрешении конфликтов, стратегическом планировании и других. Кроме того, внедрение ИИ требует достаточно высокого уровня культуры и дисциплины проектного управления. ИИ работает с данными, и эти данные должны быть достоверными и своевременно предоставляться.

В 1969 году был основан Институт управления проектами (*Project Management Institute, PMI*), который является ведущей глобальной ассоциацией для тех, кто считает управление проектами, программами или портфолио своей профессией. *PMI* представляет ценность для более чем трёх миллионов специалистов по проектам, работающих почти в каждой стране мира в условиях защиты, сотрудничества, обучения и исследований глобального уровня.

В 2019 г. *PMI* впервые описал новую компетенцию руководителя проектов *PMTQ* (*Project management technology quotient*). *PMTQ* трактуется как навык адаптации, способность управления и интеграции инновационных технологий, увязанных с проектным менеджментом. Примером высокого *PMTQ* является проект, в котором у руководителя проекта есть виртуальные помощники и/или часть задач проекта выполняют роботы. Поэтому для вступления в этап ближайшего будущего управления проектами руководителю проекта и команде проекта нужно быть готовыми к созданию и взаимодействию с роботами.

7.3. Чат-боты

В данном параграфе рассмотрим, как сделать и управлять собственным роботом на примере чат-бота.

Для начала обратимся к некоторым статистическим сведениям о чат-ботах. Во-первых, по сведениям издания *Chatbots Magazine*, уже сегодня в работе ряда компаний чат-боты экономят до 30% затрат на работе службы поддержки. Во-вторых, более 50% клиентов ожидают, что служба поддержки работает 24/7 (и чат-боты могут это обеспечить!). В-третьих, например, в социальной сети *Facebook* прекрасно работают более 300 000 активных чат-ботов.

Теперь рассмотрим некоторые потенциальные возможности чат-ботов.

Пример 1. Ваш бот может неустанно работать 24/7 и выполнять следующие задачи:

– оформлять заказ клиента прямо в окне диалога соцсети/мессенджера;

– там же принимать оплату банковской картой;

– сохранять всю требуемую информацию и присылать вам ее на почту/мобильный или выгружать в *Google*-таблицу.

После оформления заказа бот оповещает менеджера через мессенджер или по *Email* и выгружает в *Google*-таблицу необходимые данные клиента: ФИО, название пиццы, телефон, адрес доставки, информацию об оплате и т.д.

Если в процессе заказа участие человека не требуется, чат-бот справится с задачей сам. В то же время менеджер может в любой момент «перехватить» канал общения у бота в чате или связаться с клиентом по телефону, если, например, нужно уточнить информацию или подтвердить заказ.

Пример 2. Организация продаж через чат-бота в отличие от продаж через веб-сайт, не носит разовый характер. Для повторных продаж вам не надо вкладывать деньги в ремаркетинг и возвращать клиентов на сайт, снова и снова оплачивая рекламу.

Как только потенциальный клиент начал взаимодействовать с вашим ботом (просто подписался или вступил в диалог), вы тут же получили возможность писать ему все, что пожелаете (и в любой момент времени). Разумеется, у него есть возможность отписаться или заблокировать бота. Поэтому тревожить сообщениями клиента по 5 раз в день ни в коем случае не стоит. Однако можно поздравлять с праздниками, объявлять об акциях и спецпредложениях, просто желать хорошего настроения. И это будет гораздо эффективнее, чем *Email*-рассылка или постинг в социальных сетях.

Пример 3. Накопление собственной базы подписчиков, с которой вы можете взаимодействовать в любой удобный момент. Благодаря сегментации по широкому количеству параметров можно существенно повысить эффективность общения.

Впоследствии аудитория, сформированная и классифицированная чат-ботом, будет расширяться и получать информацию, которую вы хотите донести, и при этом не будет считать её неуместной или бесполезной, особенно, если повторные продажи стараться

проводить через интерактивные опросы, полезный контент, игровые механики и т.п.

На схеме (рис. 7.3) показан демонстрационный пример сегментации базы подписчиков. На основе данных, собранных чат-ботом, можно создавать уникальные подборки пользователей. Например, можно сделать рассылку только для клиентов, которые:

– брали пиццу «Маргарита» больше двух раз за последние полгода;

– заказывали ее с доставкой и покупали при этом напиток.

В несколько кликов можно сформировать список целевых (потенциально заинтересованных) клиентов и отправить им, например, такое сообщение: «Сегодня большой напиток в подарок к вашей любимой пицце «Маргарита»! Для заказа нажмите кнопку ниже».

Далее разберёмся, как создать собственного чат-бота.

Разработка чат-бота начинается с определения цели его использования. Как вариант, это может быть:

– оформление заявок на товар и передача их менеджеру;

– отправка серии сообщений для прогрева потенциальных клиентов (рассылка, формирование воронки продаж);

– консультационная поддержка (справочник, сборник ответов продавца и т.п.).

На следующем этапе выбирается площадка для размещения бота. Здесь работают, как правило, четыре альтернативы: окно обратной связи на веб-сайте, социальная сеть, мессенджер или программное приложение.

Робота можно самостоятельно запрограммировать или воспользоваться сервисом чат-ботов и за 1÷2 дня сделать алгоритм с интерфейсом. Основная задача при этом – продумать и правильно организовать логику ответов на вопросы.

Простого чат-бота можно создать самостоятельно на основе конструктора или заказать у фрилансеров за 400÷500 долларов. Создание продвинутого чат-бота у специализированных студий и профессиональных разработчиков может стоить от 1000÷1500 до десятков тысяч долларов. Содержание бота (абонентская плата) составляет от 10 до 100 долларов в месяц, в зависимости от пакета и количества подписчиков. У многих платформ-конструкторов есть бесплатные пакеты или пробные периоды.

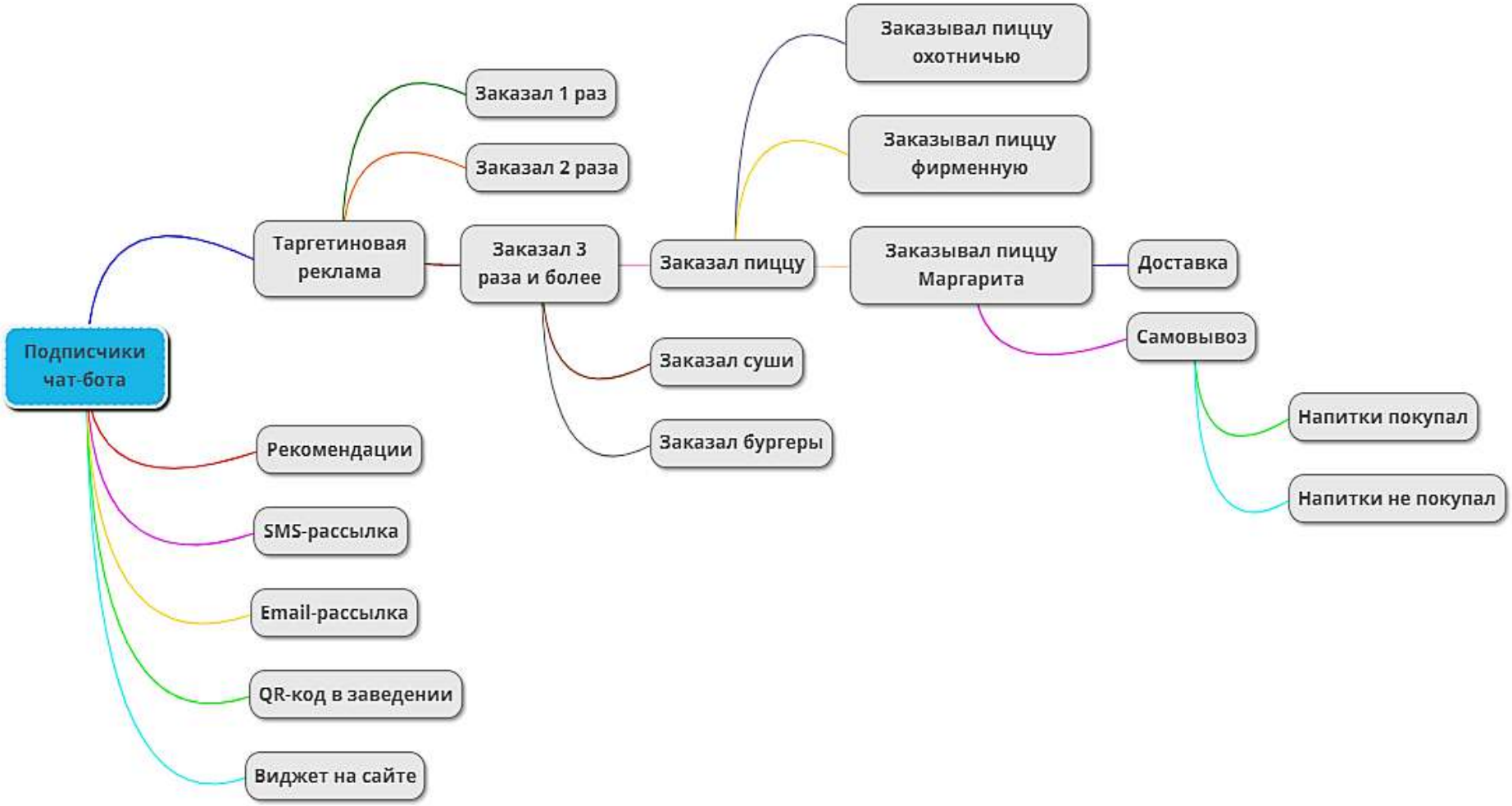


Рис. 7.3. Пример схемы сегментации подписчиков чат-бота

В заключение параграфа рассмотрим некоторые популярные конструкторы чат-ботов:

1. *Botmother*. Первого бота предлагает сделать бесплатно. Поддерживаются русский и английский языки [71].

2. *Bot Kit*. Боты конструируются на основе принципа блок-схем для любых площадок. Только на русском. Цена – от 10 долларов в месяц. Поддерживается обработка голосовых сообщений [72].

3. *Aimylogic*. Позволяет создать алгоритмы роботов для сайтов, приложений, соцсетей и мессенджеров. Поддерживаются русский и английский языки, а также обработка голосовых сообщений. Есть бесплатная версия [73].

4. *Botsify*. Констуктор коммуникационных алгоритмов для сайтов, *Facebook* и корпоративного мессенджера *Slack*. Поддерживается только английский язык. Цена – от 50 долларов в месяц. Можно создавать ботов, которые используют машинное обучение [74].

8. БУДУЩЕЕ ПРОФЕССИЙ И ПРОФЕССИИ БУДУЩЕГО

Прошедший 2020 год вошёл в историю как уникальный. Телевидение, Интернет и иные СМИ не прекращают нам сообщать о пандемии, начавшейся в этом году, как о крупнейшем кризисе современности. Так, недавно, дочерняя компания мирового лидера в области ИТ и сетей *Cisco*, разработчик систем управления производительностью программных приложений *AppDynamics* провела опрос сотрудников ИТ-компаний и представила следующие результаты:

1)81% из опрошенных заявили о том, что ощутили на своей организации самое большое давление, которое они когда-либо испытывали;

2)66% говорят, что пандемия выявила слабые места в их цифровой стратегии, вызвав необходимость в реализации инициатив, которые когда-то были частью многолетних программ цифровой трансформации;

3)71% указывают на проекты цифровой трансформации, которые были реализованы в течение недель, а не месяцев или лет, которые потребовались бы до пандемии;

4)65% сообщают, что они уже реализовали проекты цифровой трансформации во время пандемии, которые ранее считались ненужными.

Многие новшества, основанные на решениях ИИ, плотно вошли в жизнь ведущих ИТ-компаний ещё до начала пандемии. Однако помимо ИТ-компаний ИИ уверенно входит в проектное управление. Это подтверждается прогнозом аналитиков исследовательской компании *Gartner* о том, что к 2024 году технологии ИИ избавят менеджеров от 69% нагрузки. В дополнение к этому результаты опроса, выполненного Институтом управления проектами *PMI*, позволили выявить три технологии ИИ, оказывающие сегодня влияние на подавляющее число компаний: 1) системы, основанные на знаниях [34], 2) машинное обучение [76] и 3) управление принятием решений.

Также продекларированы технологии, влияние которых набирает обороты и будет расти в ближайшие годы: 1) экспертные системы, 2) глубокое обучение и 3) роботизированная автоматизация.

В связи с указанным трендом главным опасением менеджеров становится их боязнь стать сегодня невостребованными. Но не стоит забывать и то, что уровень зрелости интеллектуальных инструментов для управления проектами достаточно низкий. И это уже отмечалось в предыдущих главах данного пособия.

Стоит также отметить, что внедрение ИИ и его использование требует высокого уровня проектной дисциплины. Как минимум, ИИ работает с данными, которые должны быть достоверными и своевременно предоставленными. Поэтому говорить о скором переходе на технологии ИИ без участия человека пока рано.

В Интернет существуют прогнозные списки многих профессий (помимо тех, что относятся к управлению проектами), которые исчезнут до 2030 года [76, 77]. В эти списки вносят: юристов, журналистов, переводчиков, логистов, юрисконсультов, водителей такси, дальнобойщиков, страховых агентов и т.д. Но слово «исчезнут» здесь не совсем к месту: скорее всего, большинство профессий просто трансформируется, отдав монотонные операции роботам. Также есть ряд профессий, которые ИИ заменить не сможет: есть сферы деятельности, где он бессилен. Также существуют и профессии, на появление которых повлиял именно ИИ. Поговорим об этом подробнее в рамках последующих параграфов.

8.1. Профессии, которые не заменит ИИ

Тайваньский предприниматель в области разработок программного обеспечения и ИИ, занимавший высшие руководящие посты в китайских отделениях крупнейших американских фирм и ведущий широкомасштабные инвестиционные проекты в Китае Кай Фу Ли в своей книге «Сверхдержавы ИИ: Китай, Кремниевая Долина и новый мировой порядок» предсказывает, что в течение следующих 15 лет примерно 50% рабочих мест будут либо автоматизированы, либо переданы под управление искусственному разуму.

По утверждению М. Лаанпере, руководителя программы повышения квалификации для студентов профтехучилищ *Samsung Digi Pass* и старшего научного сотрудника Института цифровых технологий Таллиннского университета, ИИ – мощная и способная к адаптации сила, но все же она не сможет справляться со всеми задачами так, как это делает человек. Кроме того, у ИИ есть и свои

слабости. М. Лаанпере в ответ на предсказания Кай Фу Ли отмечает, что ИИ не может управлять сложным стратегическим процессом планирования, он не сможет справиться с неизвестными и неструктурированными пространствами и, в отличие от людей, у него нет чувства эмпатии. Существуют рабочие места, где человека невозможно заменить ИИ. По мнению Марта Лаанпере, ИИ не сможет заменить людей, работающих в следующих областях.

1. Психиатрия, медицина, социальная работа и, например, работа консультанта по семейным отношениям. Всё это требует наличия хороших коммуникативных навыков, эмпатии и умения заслужить доверие клиента. Для этих ролей необходимы эмоциональная интеллигентность, способность профессионально общаться с пациентами, утешать травмированных людей и предлагать им долгосрочную поддержку. Специалисты в области здравоохранения, будь то врачи или медсёстры, находятся рядом с пациентом и для того, чтобы при необходимости посочувствовать его проблемам, поддержать и приободрить.

2. Наука. Как известно, наука – наивысшее выражение творческих способностей человека. Сегодня ИИ ничего не создает самостоятельно, он всего лишь может оптимизировать цели, заданные человеком. Но несмотря на то, что ИИ не заменит учёных, он является прекрасным помощником. Например, при тестировании лекарств ИИ можно применять для имитации воздействия существующих лекарств на лечение всех известных заболеваний.

3. Информационные и инфокоммуникационные технологии. В рапорте международного консалтингового агентства *McKinsey* о профессиях будущего указывается, что к 2030 году количество инженеров в области информационных технологий возрастет с 20 до 50 миллионов. Специальности в сфере инфокоммуникационных технологий постоянно меняются, требуя от работников соответствия технологическим инновациям и трендам развития сектора, и распространяются на те сферы, которые пока не автоматизированы. ИИ не настолько гибок и не способен настолько же адаптироваться к изменениям, как человек.

4. Управление. Хорошего руководителя характеризуют превосходные коммуникативные навыки, включая умение убеждать, мотивировать сотрудников и успешно вести переговоры. Ещё важнее то, что самые талантливые руководители посредством своих слов и

поступков способны создать прочную систему ценностей и культуру труда. Хотя ИИ может успешно использоваться для измерения эффективности, другие управленческие обязанности остаются за человеком. В то же время, если руководитель – всего-навсего бюрократ, который просто сидит за столом и раздаёт приказы подчинённым, его, вероятно, заменят (но другим человеком).

5. Обучение. ИИ – отличный помощник для учителей и учебных заведений, поскольку технологии, например, помогают понять, как адаптировать учебные программы к компетенциям каждого учащегося. Однако роль учителя как наставника заключается и в том, чтобы выявить личные интересы студента, научить его учиться самостоятельно. С этими задачами сможет справиться лишь человек. Поэтому в будущем останется потребность в учителях и в контакте с человеком.

6. Спорт. Роботофутболом, битвами и гонками роботов уже никого не удивить. Такие состязания нашли своего зрителя. Однако соревнования людей гораздо популярнее, ведь кроме силы и возможностей соперников наблюдателей и самих спортсменов, прежде всего, в спорте «цепляют» именно азарт, переживания и эмоции. К тому же роботы-футболисты никогда не смогут так же эффектно симулировать боль от падения на поле, как это умеют знаменитые профессиональные спортсмены.

7. Поварское дело. Ритм современной жизни оставляет человеку все меньше времени на полноценное домашнее питание, потому рестораны и кафе вряд ли останутся без посетителей, а хороший повар, как залог успеха заведения, без работы. И если повар-робот уже может ускорить и облегчить процесс приготовления блюд, то улучшить их вкусовые свойства без помощи человека ему не под силу. А если вспомнить, что необычные рецепты появляются путем поварских проб и ошибок, то новых блюд от дисциплинированных роботов нам не дождаться.

8. Юриспруденция и адвокатура. Должности прокуроров, адвокатов и судей тоже не получится заменить компьютерами. Вместе с новыми законами увеличивается и количество сложных ситуаций и споров во всех сферах. Кому-то со всем этим нужно будет разбираться, и абсолютно точно это будет человек. И уж совсем невозможно представить, кто будет распутывать сложные клубки интриг и событий в конце детективов, если не человек-адвокат.

9. Парикмахерское дело. Первые роботы-парикмахеры уже прошли испытания: не совсем блестяще, но смогли помыть волосы и побрить клиентов. Однако сложные модельные стрижки, требующие кропотливой ручной работы, вряд ли в ближайшем будущем будут доступны компьютерам. А довериться роботу с ножницами ИИ – слишком уж рисковое испытание для потенциальной модели.

10. Розыск домашних животных. Знаменитый вымышленный персонаж Эйс Вентура – представитель вполне реальной профессии частного детектива по розыску домашних животных. Спрос на их услуги растет, и предложений от детективов все больше. Роботам в сложных делах с похищениями или пропажей животных не получится эффективно проявить себя, особенно если учесть не всегда предсказуемую психологию самих разыскиваемых.

8.2. Новые профессии от ИИ

Исследование международной консалтинговой компании *Accenture* предсказывает повышение производительности труда и рост прибыльности бизнеса на 40% за счет внедрения ИИ-решений в практике компаний.

Существует ряд рутинных рабочих задач, которые с развитием машин будут полностью переданы им на откуп, что вызовет исчезновение ряда профессий. С другой стороны, по мере развития ИИ-решений пропорционально будет расти и окно возможностей для профессионального развития людей в этой области. И речь здесь идёт о профессиях и специальностях, ранее не существовавших. Работы в ИИ хватит всем.

Для начала рассмотрим только некоторые профессии в области ИИ, которые сегодня считаются перспективными.

1. Когнитивный копирайтер. Специалист в данной области помогает боту создавать грамотные тексты, обучает его анализу содержания и эмоциональной окраске текстов.

2. Юрист (адвокат) по вопросам интеллектуальной собственности, созданной с применением ИИ. Деятельность такого юриста направлена на выявление и расследование случаев нарушения прав интеллектуальной собственности, предупреждение и пресечение таких случаев, предоставление консультаций по вопросам интел-

лектуальной собственности и экспертную оценку продуктов, созданных с помощью ИИ. Кроме того, как юрист в любой организации, он должен обеспечивать юридическое сопровождение деятельности компании, оформлять документы и разрабатывать варианты решения юридических проблем.

3. Онтоинженер. Этот специалист проектирует и создаёт экспертные системы – вычислительные системы, способные принимать решения, схожие с решениями экспертов в определённой области. Онтоинженер, выступая посредником между экспертом и программистом, структурирует знания, полученные от эксперта, и программирует операции, которые будут использоваться в продуктах с экспертными системами.

4. Проектировщик роботов. Такой специалист занимается разработкой и программированием роботов для разных сфер деятельности. В его компетенции также входит проектирование систем управления роботами с помощью различных интерфейсов, в том числе нейроинтерфейсов (обеспечивающих связь между мозгом и компьютером).

5. Нейромаркетолог. Специалист, который анализирует поведение потребителя и его реакцию на рекламу товара и предлагает эффективные стратегии продвижения товаров и услуг с точки зрения нейробиологии.

Многие профессии не исчезнут, но им придётся изменяться и переходить в цифру. Например, к 2033 году специалисты по дизайну могут стать дизайнерами виртуальных миров, экскурсоводы смогут вести экскурсии по таким мирам, а журналисты и редакторы – создавать и запускать сценарии приключений героев таких цифровых вселенных. Спасатели и военные будут управлять роботами дистанционно, а врачи освоят телемедицину, что в совокупности с диагностическими алгоритмами и 3D печатью искусственных органов ускорит эффективность лечения в десятки раз. Важно то, что нынешним специалистам надо начать изучать цифровые науки и автоматизацию как можно скорее, а студентам – осваивать информационные технологии. Главным вызовом для человека будет не только изучение интерфейсов программных роботов и платформ автоматизации, но и умение мыслить, как ИИ. Без этого управлять им не получится.

Вот несколько наиболее очевидных новых профессий будущего, которые возникнут на стыке человеческого и машинного участия.

1. Специалист по эмпатии. Уже существуют системы машинного обучения, позволяющие чат-ботам типа *Siri* и *Alexa* реагировать на вопросы людей с глубоким пониманием, состраданием и даже юмором. Это могут быть, к примеру, проблемы, связанные с потерей багажа, покупкой некачественного продукта или неисправностью в работе бытовой техники. Однако эмпатии нужно учить: машина не может сгенерировать ее самостоятельно.

2. Специалист по обучению мировоззрению и локализации. Чат-боты будущего должны считывать локальную специфику региона и общества, в которых они работают.

3. Специалист по интерпретации алгоритмов. Это ещё одна перспективная профессия, связанная с ИИ. Если система совершает ошибку, такому специалисту необходимо определить алгоритм, который к ней привел, и внести необходимые коррективы.

4. Специалист по безопасности систем ИИ. Его задача сделать все возможное, чтобы предвидеть непреднамеренные действия ИИ с возможными негативными последствиями.

5. Специалист по обеспечению устойчивости. Работает над тем, чтобы системы ИИ функционировали надлежащим образом, не выходя за пределы этических нормативов по отношению к человеку и обществу.

Нам давно понятно, что уже в среднесрочной перспективе ИИ радикально преобразит процесс профессионального обучения, придав ему перманентный характер. Объясняется это просто: за ростом возможностей и перспектив ИИ человеку нужно будет поспевать ровно с той скоростью, которую он сам и задает развитию этой технологии. Обратной дороги нет: исследование американской компании по оказанию профессиональных услуг *Genpact* показало, что 88% топ-менеджеров в технологических компаниях считают, что ИИ начнет радикально преобразовывать повседневную жизнь в ближайшие 3 года.

Более того: чем чаще люди используют ИИ-решения в своей жизни, тем больше они склонны им доверять. 41% опрошенных заявили, что ИИ-решения изменили некоторые важные аспекты их

повседневности, 35% оценили эти изменения как положительные. Негативно к ИИ настроены лишь 11%.

В каждой индустрии будут требоваться свои особые навыки, но, вероятнее всего, вектор ключевых профессий будет определяться следующей тройкой [78]:

1. Архитекторы автоматизации. Для них важны системное, последовательное мышление и опыт создания алгоритмов процессов (сценариев автоматизации). Эти алгоритмы будут загружаться в программных роботов, которые на их основе будут самообучаться и создавать комплексные, масштабируемые процессы автоматизации.

2. Аналитики *Big Data*. Важны быстрый поиск и обработка больших данных, находящихся в онлайн доступе, а также умение на основе извлеченных инсайтов определять тренды будущего. Именно аналитики будут формировать спрос на развитие прикладных и узкоспециализированных систем ИИ в области *Big Data* для каждой индустрии.

3. Пользователи прикладных систем ИИ. Для людей этой категории важно умение взаимодействовать с интерфейсами программных роботов и цифровых платформ в своей индустрии.

8.3. Там, где ИИ бессилен…

Стоит уделить внимание и тем областям жизнедеятельности и знания человека, в которых ИИ не смог подтвердить людские ожидания. – Одной из ключевых проблем, которая преследует ИИ сегодня, является то, что люди продолжают наделять ИИ человеческими качествами и считать его тем, чем он не является. ИИ получает на входе очищенные данные, анализирует их, находит шаблоны и предоставляет требуемый вывод. Однако ИИ ничего не понимает. Он не может ни создать, ни обнаружить ничего нового. У него нет никакого внутриличностного знания. Поэтому ИИ никому и ни в чем не может сочувствовать и ведёт себя так, как заложено разработавшим его человеком: его интеллект – это лишь объединение программного обеспечения и данных, анализируемых определенным образом.

У людей способность постигать является врожденной, а у ИИ она полностью отсутствует. Глядя на яблоко, человек видит боль-

ше, чем просто набор свойств, связанных с изображением объекта. Он понимает, что яблоко съедобно и содержит определенные питательные вещества. ИИ, глядя на яблоко видит объект, с которым связаны некие свойства, значения которых он не понимает, он только манипулирует ими. В следующих разделах описано, как отсутствие понимания заставляет ИИ в целом терпеть неудачу.

ИИ использует алгоритмы, чтобы манипулировать исходными данными и создать вывод. Акцент делается на выполнении анализа данных. Но направление этого анализа контролирует человек, и он же впоследствии должен интерпретировать результаты. Например, ИИ может выполнить анализ рентгеновского снимка, представляющего потенциальную опухоль. В полученном результате он может подчеркнуть часть снимка, вероятно, содержащую опухоли, чтобы врач мог ее заметить. В противном случае врач мог бы не заметить ее – таким образом, ИИ, несомненно, оказывает важную услугу. Но даже в этом случае врач все еще должен сделать обзор результата и определить, действительно ли на рентгеновском снимке опухоль.

Интерпретация подразумевает также способность видеть поверх данных. Это – не способность создавать новые данные, а понимание того, что данные могут указывать и не только на то, что и так очевидно. Например, люди нередко говорят, что некие данные неверны или сфальсифицированы, даже при том что сами данные не представляют тому доказательств. ИИ принимает как истинные все данные, а человек знает, что они могут и не быть такими. Точная формализация того, как люди решают эту задачу, в настоящее время невозможна, поскольку люди фактически этого не понимают.

Вопреки любому внешнему виду ИИ работает только с числами. Он не может понимать смысл слов, когда вы, например, с ним говорите. Он просто ищет соответствие шаблону после преобразования вашей речи в числовую форму. Теряется сущность сказанного. Даже если бы ИИ был в состоянии понять слова, он не мог бы сделать так потому, что слова исчезли в процессе лексического анализа. Невозможность ИИ понять нечто столь простое, как слова, означает, что перевод ИИ с одного языка на другой всегда будет иметь недостатки, и это бесспорно: кто-то должен перевести чув-

ства, кроющиеся за словами, а также сами слова. Слова, прежде всего, выражают чувства, а ИИ этого сделать не может.

Компьютер преобразует взгляд, звук, запах, вкус и касания в числовые представления, а затем находит соответствие шаблону, чтобы создать набор данных, который моделирует реальную практику. Дальше дело куда сложнее, поскольку разные люди обычно переживают одни и те же вещи по-разному. Например, каждый человек воспринимает цвет индивидуально. Для ИИ каждый компьютер видит цвет совершенно одинаково, а значит, ИИ не может воспринимать цвета индивидуально. Однако из-за преобразований ИИ фактически не ощущает цвет вообще.

ИИ может анализировать данные, но не принимать моральные или этические решения. Если вы попросите ИИ сделать выбор, то он всегда выберет вариант с самой высокой вероятностью успеха, если вы не включите также своего рода функцию случайности.

ИИ может интерполировать существующее знание, но не может его экстраполировать, чтобы создать новое знание. Столкнувшись с новой ситуацией, он обычно пытается решить ее исходя из существующих знаний, вместо того чтобы придумать что-то новое. Фактически у ИИ нет никакого способа создавать что-то новое или видеть нечто как-то уникально. Только человеческие чувства помогают нам открывать новые вещи, работать с ними, изобретать методы для взаимодействия с ними и создавать новые методы для того, чтобы использовать их для выполнения новых задач или лучшего решения прежних задач.

В настоящее время ИИ способен заметить в данных те шаблоны, которые не очевидны для людей. Именно возможность видеть эти шаблоны делает ИИ настолько ценным. Манипулирование данными и их анализ являются трудоемким, сложным и монотонным делом, но ИИ легко может выполнить эту задачу. Однако шаблоны данных – это просто вывод и необязательно решение. Люди полагаются на пять органов чувств, сочувствие, творческий потенциал и интуицию, чтобы видеть вне шаблонов и находить потенциальное решение вне рамок, обусловленных данными.

Проще всего понять человеческую способность видеть вне шаблонов – это посмотреть на небо. В облачный день люди могут видеть шаблоны в облаках, но ИИ видит облака и только облака. Кроме того, два человека могут видеть различные вещи в одном и

том же наборе облаков. При творческом взгляде на шаблоны в облаках один человек может видеть овцу, а другой – фонтан. То же самое относится к звездам и другим видам шаблонов. ИИ представляет шаблон как результат, но не понимает его, поскольку отсутствие творческого потенциала не позволяет ему сделать с шаблоном ничего, кроме как указать в отчете, что шаблон существует.

Впоследствии люди узнали о таких вариациях в человеческих органах чувств, которые фактически нереально корректно передать искусственному интеллекту, поскольку репликация этих чувств на аппаратных средствах сейчас действительно невозможна. Например, способность использовать разные органы чувств для обследования одного и того же ввода [79] находится вне возможностей ИИ.

Хотя общеизвестно, что у людей есть пять органов чувств, некоторые источники сейчас утверждают, что фактически люди имеют намного больше пяти обычных чувств [80]. Некоторые из этих дополнительных чувств не очень хорошо поняты и едва доказуемы, такие как магниторецепция [81]. Это чувство позволяет людям указывать направление, подобно птицам, но в меньшей степени. Поскольку нет никакого способа даже определения этого чувства, его репликация, как элемента ИИ, невозможна.

Идея войти в чужое положение означает взгляд на вещи с точки зрения другого человека, когда он испытает чувства, подобные испытываемым другим человеком. Никто действительно не чувствует то же самое, что и кто-то еще, но благодаря сочувствию люди могут себе это представить. Эта форма сочувствия требует сильного внутриличностного интеллекта в качестве отправной точки, чего никогда не будет иметь ИИ, если он не выработает самоощущение. Кроме того, ИИ должен быть в состоянии чувствовать нечто, что в настоящее время для него невозможно, и быть открытым для того, чтобы разделять чувства с некой другой сущностью (обычно – с человеком), что также невозможно. Текущее состояние технологии ИИ не позволяет ему чувствовать или понимать любой вид эмоции, что делает сочувствие невозможным.

Когда речь заходит о взаимоотношениях, люди должны учитывать как интеллектуальную привязанность, так и чувства. Источником интеллектуальной привязанности зачастую является общая вы-

года для двух сущностей. К сожалению, у ИИ и человека (или любого другого субъекта) нет никакой общей выгоды. ИИ просто обрабатывает данные, используя определенный алгоритм. Нечто не может претендовать на то, чтобы любить нечто другое, даже если ему это приказано. Эмоциональная привязанность несет с собой риск отказа, что подразумевает самооценку.

Уверенность в истинности чего-то без доказывающих фактов называется верой. Как правило, вера имеет форму доверия, т.е. верой в искренность другого человека безо всяких доказательств тому, что этот человек заслуживает доверия. ИИ не может продемонстрировать ни веры, ни доверия. Поскольку эта способность у ИИ отсутствует, он не может продемонстрировать способность проникновения в суть – требование, необходимое для шаблонов мышления, подобных человеческим. За примерами веры далеко ходить не нужно – на её основе были сделаны многие изобретения, создавшие нечто новое. Одним из самых заметных изобретателей был Томас Эдисон. Например, он предпринял тысячу (а возможно, и больше) попыток создать лампу накаливания. ИИ при этом, вероятно, сдался бы после определенного количества попыток...

ИТОГОВЫЙ ТЕСТ ПО ТЕОРЕТИЧЕСКОЙ ЧАСТИ

1. Какие два основных направления сформировались в ИИ после его признания отдельной областью науки?

(1) анализ данных и машинное обучение
(2) нейрокибернетика и «кибернетика черного ящика»
(3) нейрокибернетика и нечёткая логика
(4) робототехника и нейрокибернетика

2. Как можно охарактеризовать *NP*-полные задачи?

(1) для решения этих задач требуются алгоритмы построения логического вывода
(2) они отличаются от других тем, что поиск их решения основан на методе проб и ошибок
(3) они отличаются от других тем, что поиск их решения за разумный период времени всё еще невозможен
(4) они решаются на основе последовательности операций, гарантирующей нахождение правильного решения за конечный промежуток времени

3. Что отличает графические процессоры?

(1) они обычно содержат сотни вычислительных ядер
(2) они способны заменить множество компьютерных систем, снабженных процессорами с архитектурой фон Неймана
(3) они способны напрямую обращаться к оперативной памяти компьютера
(4) все ответы верны

4. Какой голосовых переводчиков является лидером в области динамического перевода?

(1) Яндекс Переводчик
(2) *Google* Переводчик
(3) *Bing Microsoft Translator*
(4) *Skype Translator*

5. Что такое «экспертная система»?

(1) это сложный программный комплекс, аккумулирующий знания специалистов в конкретной предметной области и тиражирующий этот эмпирический опыт для консультаций менее квалифицированных пользователей
(2) это программа, имитирующая на компьютере мышление человека
(3) это программный комплекс, предоставляющий эффективные способы хранения, представления и извлечения данных
(4) это система, которая выполняет частную задачу управления (например, поддержания параметров промышленного объекта на заданном уровне)

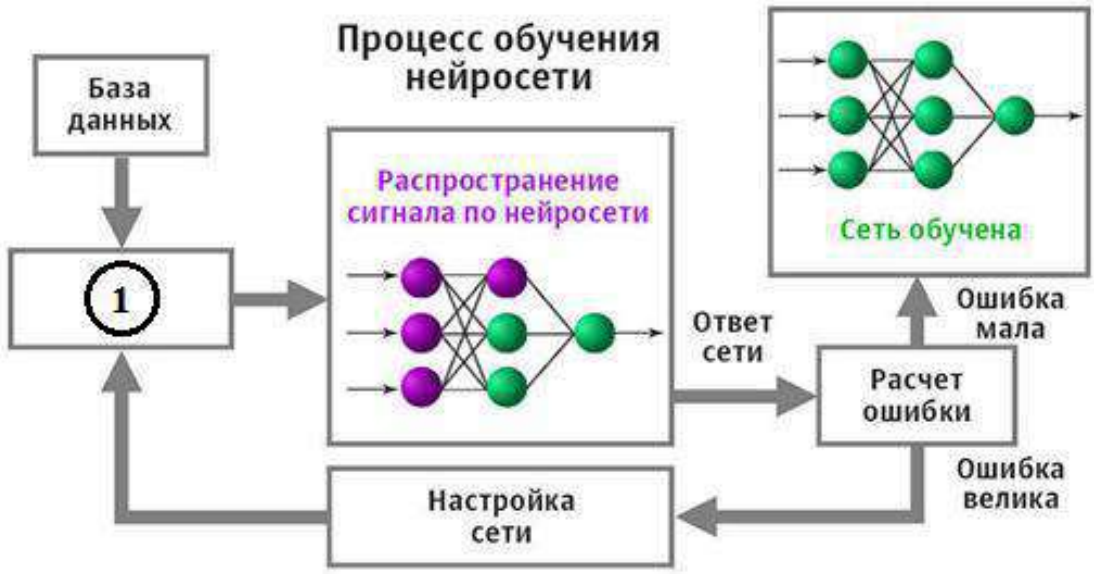
6. Укажите правильную последовательность выполнения преобразований, составляющих основу анализа данных.

(1) проверка, чистка, преобразование, моделирование
(2) проверка, преобразование, чистка, моделирование
(3) преобразование, проверка, чистка, моделирование
(4) преобразование, чистка, проверка, моделирование

7. К какой группе машинного обучения относятся алгоритмы, которые учатся на основе заданных примеров и относящихся к цели ответов?

(1) обучение с подкреплением
(2) обучение с учителем
(3) обучение без учителя
(4) ни к одной из указанных групп

8. Укажите назначение блока в составе схемы процесса обучения нейросети, который обозначен цифрой «1».

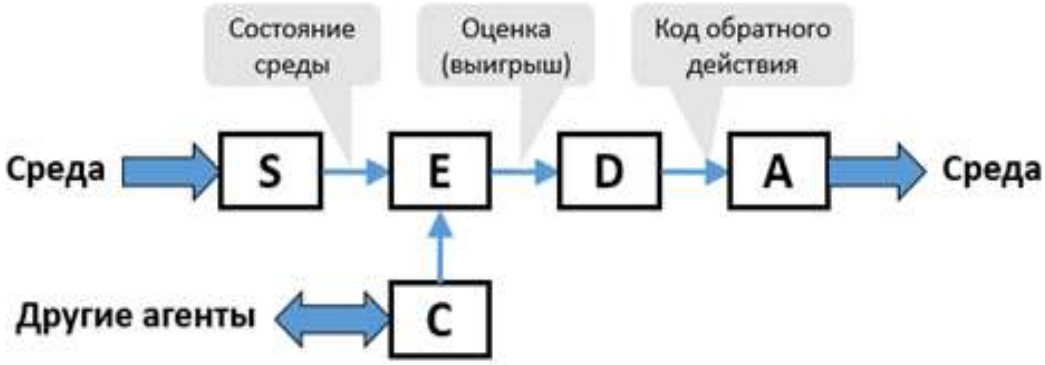


(1) ответ сети
(2) выбор примера
(3) анализ ошибки
(4) настройка сети

9. Что означает характеристика «проактивность» в отношении интеллектуального агента?

(1) способность ощущать внешнюю среду и реагировать на изменения в ней, совершая действия, направленные на достижение целей
(2) способность показывать управляемое целями поведение, проявляя инициативу, совершая действия, направленные на достижение целей
(3) способность взаимодействовать с другими сущностями внешней среды (агентами, людьми и т.д.) для достижения целей
(4) способность функционирования без прямого вмешательства кого-либо и обладание определенной способностью контролировать свои действия и внутреннее состояние

10. Какой из блоков функциональной структуры интеллектуального агента отвечает за выбор им следующего действия?



(1) S
(2) E
(3) D
(4) C
(5) A

11. Какой из приборов беспилотного автомобиля создает основу трёхмерной карты окружающего пространства, сканируя его с полным обзором в 360 градусов?

(1) лидар
(2) радар
(3) одометр
(4) гиростабилизатор

12. Совокупность каких элементов определяет онтологическую модель предметной области?

(1) множество концептов, классов, понятий, терминов
(2) множество отношений
(3) множество концептов и отношений между ними
(4) множество концептов, отношений между ними и множество их функций интерпретации

13. Кто впервые описал и систематизировал инструментарий интеллект-карт?

(1) американский математик, один из основоположников кибернетики и теории ИИ Норберт Винер
(2) американский математик и логик Лотфи Заде
(3) американский учёный в области психологии, нейрофизиологии и ИИ Фрэнк Рознблатт
(4) английский писатель и психолог Тони Бьюзен

14. Как называется подготовка задачи для решения методами нечеткой логики, позволяющая конвертировать реальные значения переменных в нечеткие?

(1) логическое преобразование
(2) логический вывод
(3) фаззификация
(4) дефаззификация

15. Что такое «дипфейк»?

(1) это алгоритм ИИ для анализа изображений сетчатки глаз пациентов
(2) это смешной аудио или видеоконтент, сгенерированный при помощи машинного обучения
(3) это компрометирующий аудио или видеоконтент, сгенерированный при помощи машинного обучения
(4) это способ обучения искусственных нейронных сетей, основанный на использовании заведомо ложных данных

16. В 2020 г. была разработана система ИИ, позволяющая воссоздать процесс написания картин и нанесения мазков для произведений живописи известных художников. Что лежит в основе этой системы?

(1) нейроморфный процессор
(2) свёрточная нейронная сеть
(3) механизм логического вывода
(4) многоядерный графический процессор

17. Какие из интеллектуальных инструментов позволяют руководителям управлять проектами с помощью программных роботов?

(1) *Lili.ai*
(2) Битрикс24
(3) Бот Дина
(4) *Stratejos*

18. Что из перечисленного не относится к потенциальным возможностям чат-ботов?

(1) выполнение работы в режиме 24/7
(2) организация продаж в повторяющемся режиме
(3) возможность быстрого анализа сложных запросов
(4) накопление и сегментация клиентской базы

19. Как называется специалист, структурирующий знания, полученные от эксперта, и программирующий операции, которые будут использоваться в продуктах с экспертными системами?

(1) онтоинженер
(2) когнитивный копирайтер
(3) нейромаркетолог
(4) инженер-конструктор

20. Какие из указанных позиций входят в состав вектора ключевых профессий ближайшего будущего?

(1) архитекторы автоматизации
(2) аналитики *Big Data*
(3) пользователи прикладных систем ИИ
(4) все ответы верны

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

«Релевантный информационный поиск»

Цели тестового задания:

– анализ знаний и навыков в отношении Интернет-поиска целевой информации;

– оценка эффективности поиска и производительности испытуемого.

Время выполнения

Обсуждение задачи с преподавателем лучше выполнить в конце недели (четверг/пятница). Для выполнения задания следует использовать время в течение выходных дней. Предоставить результаты преподавателю следует в понедельник.

Краткие теоретические сведения

Сегодня объем информации растет быстрее возможности человека адекватно её обрабатывать. Быстрый поиск необходимой для работы информации может существенно повысить качество её результатов и ускорить выполнение. Поэтому информационный поиск следует рассматривать, как элемент компетентностного подхода, создающего необходимые условия для повышения эффективности работы сотрудников.

Важно отметить, что практически все современные поисковые системы сегодня построены на основе ИИ. Так, например, популярные системы Яндекс и *Google*, используя искусственные нейронные сети, реализуют огромное число сервисных функций. И первая из них – фильтрация поискового спама. Вторая функция – ранжирование результатов запроса по релевантности.

Релева́нтность (лат. *relevo* – поднимать, облегчать) в информационном поиске – семантическое соответствие поискового запроса и поискового образа документа. В более общем смысле, одно из наиболее близких понятию качества «релевантности» – «адекватность», то есть не только оценка степени соответствия, но и степени практической применимости результата.

Лучшими русскоязычными поисковыми системами принято считать *Google*, Яндекс и *Nigma.ru* (научный проект «Интеллекту-

альная поисковая система *Nigma.ru*» МГУ им. М.В. Ломоносова [82]). Среди научных поисковых систем выделяют научную электронную библиотеку *eLIBRARY.RU* [83], Академию *Google* [84], *Scopus*/*ScienceDirect* [85] и *Web of Science* [86]. С дополнительными сведениями тематического классификатора поисковых систем предлагается ознакомится самостоятельно [87].

Задание

1. Используя 5 поисковых систем (*Google*, Яндекс, *Nigma.ru* плюс две системы тематического поиска на выбор – в соответствии с вариантом задания), осуществите поиск информации, указанной в Вашем варианте. Отобразите в отчёте для каждой поисковой системы результат поиска (первые 5 ресурсов). Выполните сравнительный анализ результатов (% соответствующих содержанию запроса (да, нет, частично), % совпавших (да, нет)). На основе полученных данных постройте в *Microsoft Excel* круговые диаграммы и оформите отчёт.

2. Используя аналогичный запрос, воспользуйтесь функцией расширенного поиска. Максимально уточните запрос (окно параметров и результат представить в отчёте). Постройте аналогичные графики.

3. Используя образец темы Вашего варианта, воспользуйтесь функцией поиска с элементами условий для поиска: *and*, *or*, &, +, -, вложенные условия через скобки. Оптимально подберите условия запроса (окно с запросом и результат представить в отчёте). Постройте аналогичные графики.

4. Найдите несколько определений понятия «Релевантность».

Вопросы

1. Какой из методов поиска дал наилучшие результаты?

2. Что, на Ваш взгляд, является причиной того, что разные сайты на один и тот же запрос выдают различный набор найденных ресурсов?

3. Как релевантность влияет на возможность информации быть найденной?

Варианты заданий:

№	Информация для поиска	Тематика
1	Эволюционное программирование	Информационные технологии
2	Медицинский экзоскелет	Медицина
3	Интеллектуальный анализ текстов	Искусственный интеллект
4	Искусственная социальность	Социология
5	Регрессионный анализ	Математика
6	Фитнес-браслет	Спорт, здоровье
7	Последние достижения дронов	Робототехника
8	Бесплатное создание чат-бота	Бизнес
9	Колоборативное обучение	Образование
10	Целесообразность использования роботов-пылесосов	Бытовая техника
11	Инструменты роботизации бизнес-процессов	Управление, аналитика
12	Перспективные профессии от искусственного интеллекта	Карьера, образование

Представление результатов

Результаты поиска для поисковых систем представьте в виде таблицы следующего вида:

№	Поисковая система 1	Степень совпадения 1	…	Поисковая система 5	Степень совпадения 5
1	Адрес сайта 1	Да	…	Адрес сайта 1	Нет
…	…		…	…	…
5	Адрес сайта 5	Частично	…	Адрес сайта 5	Да

На основании анализа информации из таблицы обоснуйте выбор поисковой системы, предоставившей лучшие результаты по интеллектуальному поиску целевой информации.

ГРУППОВЫЕ КЕЙСЫ

Кейс 1. Интеллект-карта востребованных услуг

Цель выполнения кейса:

– развитие у студентов групповых навыков аналитической работы и умений по составлению интеллект-карт.

Требования к выполнению:

– задание выполняется в группах по 5÷7 человек в течение фиксированного времени (1 час 30 минут) на основании предложенного описания ситуации для указанной задачи;

– результаты выполнения кейса необходимо сохранить для выполнения следующего кейса (Кейса 2).

Описание ситуации

В последние десятилетия в РФ и, в Ростовской области в частности, стали появляться и развиваться коттеджные поселки, предназначенные под малоэтажную застройку. На текущий момент таких поселений только на прилегающих к г. Ростов-на-Дону землях уже около пятидесяти [88, 89, 90].

Управление посёлком, например, на основе ТСЖ, формируется далеко не везде и не всегда сразу. При этом число проблем, с которыми сталкиваются в ходе строительства владельцы участков, от этого не уменьшается.

Ситуация осложняется тем, что новые технологические тренды и технологии постоянно вносят свои изменения в окружающую обстановку. И о них, как минимум, стоит иметь представление [91] Так, например, в течение последнего полугодия коттеджный посёлок «Радужный» (Большелогское сельское поселение) был разделен на две части. Около 20 его домов были выкуплены и снесены.

Возникающие проблемы требуют решения. В связи с этим очевидна необходимость в привлечении подрядчиков, исполнителей и отдельных специалистов, оказывающих всевозможные услуги: начиная от рытья траншей и доставки обедов до предоставления услуг провайдера Интернет, и консультирования по правовым вопросам.

В сложившейся ситуации формируется потенциал для создания и развития рынка востребованных услуг на территориях коттеджных посёлков. При этом каждый из посёлков имеет ряд специфических характеристик:

– удалённость от города с развитой инфраструктурой;
– численность проживающих;
– престижность района и стоимость земельных участков;
– подверженность плановым технологическим изменениям;
– наличие дорог;
– закрытость/доступность территории;
– наличие коммерческой недвижимости под аренду и т.д.

Задача

Сформируйте интеллект-карту с перечнем из 5÷7 наиболее перспективных услуг с целью последующей подготовки и реализации сервисных чат-ботов для жителей и застройщиков коттеджных посёлков. Предварительно выберите и обоснуйте основные направления карты трендов и технологий [91], которые вы будете учитывать в своей разработке. Детализируйте полученную интеллект-карту.

Кейс 2. Проектирование сервисных чат-ботов

Цель выполнения кейса:

– развитие у студентов навыков и умений по проектированию чат-ботов и выбору платформ для их реализации.

Требования к выполнению:

– задание выполняется в группах по 2÷3 человека в течение фиксированного времени (1 час 30 минут) на основании описания ситуации для указанной задачи;

– исходными данными для данного кейса являются результаты выполнения предыдущего кейса (Кейс 1).

Описание ситуации

После разработки интеллект-карты перспективных услуг для оказания жителям коттеджных поселков Ростовской области возникает вопрос в программной реализации соответствующего про-

екта. Перед Вами стоит задача выполнения проектирования чат-ботов, последующее использование которых позволит:

– автоматизировать сбор заявок на оказание сервисных услуг от клиентов;

– сформировать и сегментировать базу их контактов и предпочтений;

– обеспечить информационную поддержку сервисов в режиме 24/7;

– предоставить заказчикам возможность ознакомиться с конкретными услугами без выполнения телефонных звонков;

– сделать процедуру оформления заказа быстрой, простой и интуитивно понятной.

На текущий момент уже существует множество реализованных примеров сервисных чат-ботов [92], как для обращения в центры по ремонту электроники [93], так и для заказа бутилированной воды [94].

Основная задача при создании чат-бота – продумать и правильно организовать логику ответов на вопросы потенциальных заказчиков. В качестве примера рассмотрим логику работы последнего из упомянутых ботов. Она описывается на уровне сценария общения с клиентом и карты работы.

Сценарий общения клиента с чат-ботом:

1. После того, как потенциальный клиент написал в чат компании, бот приветствует его и предлагает либо сразу сделать заказ, либо оформить заявку после ознакомления с ассортиментом.

2. Если человек хочет сразу сделать заказ, бот задает ему несколько уточняющих вопросов:

– о виде воды (питьевая, минеральная);

– для кого осуществляется заказ (предприятие или частное лицо);

– об объеме воды в заказе.

3. Затем пользователю нужно подтвердить номер телефона, который бот получает автоматически из электронной базы, и сообщить адрес доставки.

При выборе на первом этапе варианта «Ознакомиться», клиент получает от бота информацию об ассортименте в виде преимуществ, а после этого бот предлагает сделать заказ.

Описанный чат-бот реализован и доступен для ознакомления [95] на платформе *Facebook*. С картой его работы предлагается ознакомиться самостоятельно [94].

Задача

Разработайте общий сценарий для работы сервисных чат-ботов и детализируйте его на уровне карт работы для каждого из сервисов, вошедших в состав интеллект-карты перспективных услуг. Подумайте и обоснуйте выбор вариантов платформы и конструктора для реализации спроектированных чат-ботов.

КЛЮЧИ к итоговому тесту

1(2)	2(3)	3(4)	4(4)	5(1)	6(4)	7(2)	8(2)	9(2)	10(3)
11(1)	12(4)	13(4)	14(3)	15(2)	16(2)	17(2)	18(3)	19(1)	20(4)

ЛИТЕРАТУРА

1. Мохов В.А., Кузнецова А.В. Интеллектуальные технологии и представление знаний: учебное пособие / Юж.-Рос. гос. техн. ун-т (НПИ). - Новочеркасск: ЮРГТН(НПИ), 2012. - 104 с.
2. Павлюкевич В. Закат архитектуры фон Неймана, о котором вы еще не слышали и что будет дальше? [Электронный ресурс] // OCTADERO: информ. портал. 2020. 2 июня. URL: https://www.octadero.com/2020/06/02/закат-архитектуры-фон-неймана-о-котор/ (дата обращения: 12.05.2021).
3. Google Brain [Электронный ресурс] // Википедия: информ.-справочный портал. URL: https://ru.wikipedia.org/wiki/Google_Brain (дата обращения: 12.05.2021).
4. Evanson N. Explainer: What Are Tensor Cores? [Электронный ресурс] // techspot.com: информ. портал. 2020. 2 июля. URL: https://www.techspot.com/article/2049-what-are-tensor-cores/ (дата обращения: 12.05.2021).
5. ИИ в режиме реального времени: Microsoft представляет предварительную версию Project Brainwave [Электронный ресурс] // news.microsoft.com: информ. портал. 2018. 17 мая. URL: https://news.microsoft.com/ru-ru/features/project-brainwave/ (дата обращения: 12.05.2021).
6. Азимов Айзек. Я, робот / Предисл. И. Ефремова. – М.: Знание, 1964.
7. Been E.A. Asimov’s Three Laws Helped Shape A.I. and Robotics. We Need Four More. [Электронный ресурс] // medium.com: информ. портал. 2020. 17 ноября. URL: https://onezero.medium.com/its-time-to-add-4-new-laws-of-robotics-8791139cdb11 (дата обращения: 12.05.2021).
8. Робот Атлас от Boston Dynamics первым научился делать сальто. BBC News - Русская служба [Электронный ресурс] // youtube.com: информ. видеопортал. 2017. 17 ноября. URL: https://www.youtube.com/watch?v=KJ0E5vB7ABs (дата обращения: 12.05.2021).
9. Танцующие роботы Boston Dynamics [Электронный ресурс] // youtube.com: информ. видеопортал. 2020. 30 декабря. URL:

https://www.youtube.com/watch?v=OPAXkv4Bob8 (дата обращения: 12.05.2021).
10. Chow B. New Exoskeleton Does the Heavy Lifting for Factory Workers [Электронный ресурс] // nbcnews.com: информ. портал. 2017. 9 ноября. URL: https://www.nbcnews.com/mach/science/new-exoskeleton-does-heavy-liftingfactory-workers-ncna819291 (дата обращения: 12.05.2021).
11. Matchar E. This Robotic Exoskeleton Helps Kids With Cerebral Palsy Walk Upright [Электронный ресурс] // smithsonianmag.com: информ. портал. 2017. 7 сентября. URL: https://www.smithsonianmag.com/innovation/this-robotic-exoskeleton-helps-kids-cerebral-palsy-walk-upright-180964750/ (дата обращения: 12.05.2021).
12. Словения. Научиться ходить после инсульта [Электронный ресурс] // euronews.com: информ. портал. 2018. 15 мая. URL: https://ru.euronews.com/2017/05/15/the-robotics-helping-patients-learn-how-to-walk-again (дата обращения: 12.05.2021).
13. Shaer M. Is This the Future of Robotic Legs? [Электронный ресурс] // smithsonianmag.com: информ. портал. 2014. 1 ноября. URL: https://www.smithsonianmag.com/innovation/future-robotic-legs-180953040/ (дата обращения: 12.05.2021).
14. Павлов В. Н. и др. Применение хирургической робот-системы при удалении опухоли орбиты (случай из практики) // Опухоли головы и шеи. – 2019. - Т.9. - № 4. - С.80-84.
15. Материалы официального сайта инновационной компании в области фитнеса Moov Inc. [Электронный ресурс]. 2020. URL: https://welcome.moov.cc/ (дата обращения: 12.05.2021).
16. Материалы официального сайта инновационной компании в области фитнеса K'Watch Inc. [Электронный ресурс]. 2020. URL: http://www.pkvitality.com/ktrack-glucose/ (дата обращения: 12.05.2021).
17. Watson S. Wearable Defibrillator Vest: Pros and Cons [Электронный ресурс] // healthcentral.com: информ. портал. 2014. 19 мая. URL: https://www.healthcentral.com/article/wearable-defibrillator-vest-pros-and-cons (дата обращения: 12.05.2021).
18. Qardio Announces FDA Clearance for QardioCore, a self-fit continuous ambulatory ECG to view and annotate ECG [Электронный ресурс] // prnewswire.com: информ. портал. 2021. 11 марта. URL:

https://www.prnewswire.com/news-releases/qardio-announces-fda-clearance-for-qardiocore-a-self-fit-continuous-ambulatory-ecg-to-view-and-annotate-ecg-301244960.html (дата обращения: 12.05.2021).
19. Wilson A.D., Baietto M. Applications and advances in electronic-nose technologies // Sensors. – 2009. – Т. 9. – №. 7. – С. 5099-5148.
20. Таблица символов Юникода: Эмотиконы. [Электронный ресурс] // unicode-table.com: информ. портал. 2021. URL: https://unicode-table.com/ru/blocks/emoticons/ (дата обращения: 12.05.2021).
21. Эмоджи Фильм [Электронный ресурс] // youtube.com: информ. видеопортал. 2017. 7 ноября. URL: https://www.youtube.com/watch?v=kNnYqgJ8dL4 (дата обращения: 12.05.2021).
22. Exclusive: Google’s New AI Tool Turns Your Selfies Into Emoji [Электронный ресурс] // fastcompany.com: информ. портал. 2017. 5 ноября. URL: https://www.fastcompany.com/90124964/exclusive-new-google-tool-uses-ai-to-create-custom-emoji-of-you-from-a-selfie (дата обращения: 12.05.2021).
23. Материалы сервиса статистического анализа медицинских карт Health InfoScape [Электронный ресурс] // Информ. портал. Массачусетского технологического института, Senseable City Lab, 2011. URL: http://senseable.mit.edu/healthinfoscape/ (дата обращения: 12.05.2021).
24. Невала К. Машинное обучение для чайников (и не только) [Электронный ресурс] // SAS Best Practices. 2021. URL: https://www.sas.com/ru_ru/insights/articles/analytics/machine-learning-for-beginners-and-beyond.html (дата обращения: 12.05.2021).
25. Хенрик Б., Джозеф Р., Марк Ф. Машинное обучение // Издательский дом «Питер», 2017. - 336 с.
26. «Если без хайпа, то ИИ повышает производительность труда»: 15 тезисов о машинном обучении профессора МФТИ Константина Воронцова [Электронный ресурс]// Новая газета. 2019. 25 февраля. URL: https://novayagazeta.ru/articles/2019/02/25/79685-esli-bez-haypa-to-ii-povyshaet-proizvoditelnost-truda (дата обращения: 12.05.2021).
27. Решения глубокого обучения и машинного обучения [Электронный ресурс] // Информ. портал Hewlett Packard Enterprise. 2021. URL: https://www.hpe.com/ru/ru/compute/hpc/deep-learning.html (дата обращения: 12.05.2021).

28. UNIMATE the first programmable industrial robot [Электронный ресурс] // youtube.com: информ. видеопортал. 2020. 6 января. URL: https://www.youtube.com/watch?v=qiFOgeFzueA (дата обращения: 12.05.2021).
29. Материалы первого независимого российского онлайн-издания 3DNews Daily Digital Digest, посвященного цифровым технологиям: Теги - роботы [Электронный ресурс] // 3dnews.ru: информ. портал. 2021. 12 мая. URL: https://3dnews.ru/tags/роботы/page-2.html (дата обращения: 12.05.2021).
30. Гаврилова Т.А. Как из менеджера сделать аналитика: опыт подготовки инженеров по знаниям [Электронный ресурс] // youtube.com: информ. видеопортал. 2020. 8 января. URL: https://www.youtube.com/watch?v=PgyX4lkTwhA (дата обращения: 12.05.2021).
31. Wu Z., Chen H., Jiang X. (ed.). Modern Computational Approaches to Traditional Chinese Medicine. – Elsevier, 2012. URL: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/unified-medical-language-system (дата обращения: 12.05.2021).
32. United Nations Standard Products and Services Code (UNSPSC) [Электронный ресурс] // unspsc.org: информ.-справочный портал. 2021. URL: https://www.unspsc.org (дата обращения: 12.05.2021).
33. Гаврилова Т.А. Дискуссия на 5 канале «Искусственный разум – фантастика или реальность будущего?» [Электронный ресурс] // youtube.com: информ. видеопортал. 2016. 19 августа. URL: https://www.youtube.com/watch?v=hbkumztxNM0 (дата обращения: 12.05.2021).
34. Protege. A free, open-source ontology editor and framework for building intelligent systems [Электронный ресурс] // protege.stanford.edu: информ.-сервисный портал. Stanford. 2020. URL: https://protege.stanford.edu (дата обращения: 12.05.2021).
35. Дикий Д.Г. Стандарт ASD S1000D. Эффективные методики разработки технической документации. Бумажное прошлое или электронное настоящее? [Электронный ресурс] // Журнал «ИСУП». 2017. - №6(72). URL: https://isup.ru/articles/1/12266/ (дата обращения: 12.05.2021).
36. S1000D [Электронный ресурс] // ASD (Европейская ассоциация авиационно-космической и оборонной промышленности) :

[сайт]. URL: www.asd-europe.org/s1000d (дата обращения: 12.05.2021).
37. Интерактивная электронная техническая документация [Электронный ресурс] // itorum.ru: информ. портал. 2021. URL: https://itorum.ru/articles/interaktivnaya-elektronnaya-tehnicheskaya-dokumentacziya/ (дата обращения: 12.05.2021).
38. Бьюзен Т. Интеллект-карты. Полное руководство по мощному инструменту мышления» // М: Манн, Иванов и Фербер, 2019. - 199 с.
39. Материалы сервиса iMindMap [Электронный ресурс] // imindmap.com: информ. портал. 2011. URL: https://app.imindmap.com (дата обращения: 12.05.2021).
40. Информационная страница Гавриловой Т.А. [Электронный ресурс] // Software Engineering Conference Russia: информ. портал. 2019. 15 ноября. URL: https://2019.secrus.org/program/invited-speakers/tatiana-gavrilova/ (дата обращения: 12.05.2021).
41. Шеври Ф., Гели Ф. Нечеткая логика // Техническая коллекция Schneider Electric. 2009. №31. - 30 с.
42. Алгоритм Мамдани в системах нечеткого вывода [Электронный ресурс] // habr.com: информ. портал. 2021. URL: https://habr.com/ru/post/113020/ (дата обращения: 12.05.2021).
43. FDA permits marketing of artificial intelligence-based device to detect certain diabetes-related eye problems [Электронный ресурс] // fda.gov: информ.-справочный портал. 2018. 11 апреля. URL: https://www.fda.gov/news-events/press-announcements/fda-permits-marketing-artificial-intelligence-based-device-detect-certain-diabetes-related-eye (дата обращения: 12.05.2021).
44. Чайников Ю. ИИ-тренды: чего ждать в сфере искусственного интеллекта в 2020 году [Электронный ресурс] // vc.ru: информ. портал. 2020. 4 марта. URL: https://vc.ru/ml/110478-ii-trendy-chego-zhdat-v-sfere-iskusstvennogo-intellekta-v-2020-godu (дата обращения: 12.05.2021).
45. AI помогает изучать животных Африки [Электронный ресурс] // habr.com: информ. портал. 2019. 19 августа. URL: https://habr.com/en/company/cloud4y/blog/464155/ (дата обращения: 12.05.2021).
46. Welcome to UiPath Marketplace [Электронный ресурс] // uipath.com: информ. портал. 2021. URL: https://marketplace.uipath.com (дата обращения: 12.05.2021).

47. Алгоритм AlphaFold от DeepMind решил 50-летнюю задачу фолдинга белка [Электронный ресурс] // habr.com: информ. портал. 2020. 1 декабря. URL: https://habr.com/ru/news/t/530718/ (дата обращения: 12.05.2021).
48. Jukebox Sample Explorer [Электронный ресурс] // openai.com: информ. портал. 2020. 1 декабря. URL: https://jukebox.openai.com/ (дата обращения: 12.05.2021).
49. Timecraft. A learning-based method for synthesizing time-lapse videos of paintings [Электронный ресурс] // github.com: информ. портал. 2020. URL: https://xamyzhao.github.io/timecraft/ (дата обращения: 12.05.2021).
50. Галерея нейросетевого искусства [Электронный ресурс] // yandex.ru: информ. портал. 2020. URL: https://yandex.ru/lab/ganart (дата обращения: 12.05.2021).
51. Смотри mail диктор [Электронный ресурс] // mail.ru: информ. портал. 2021. URL: https://dictor.mail.ru/ (дата обращения: 12.05.2021).
52. Фиговский О. Некоторые достижения искусственного интеллекта и варианты его влияния на социум [Электронный ресурс] // Наука и жизнь Израиля. Технические науки. 2020. 15 апреля. URL: http://nizinew.co.il/nauka/texnicheskie-nauki/nekotorye-dostizheniya-iskusstvennogo-intellekta-i-varianty-ego-vliyaniya-na-socium.html (дата обращения: 12.05.2021).
53. Ясницкий Л.Н. Нейро-экспертная система «KardioNet» [Электронный ресурс] // kardionet.ru: информ.-сервисный портал. 2021. URL: https://www.kardionet.ru/ (дата обращения: 12.05.2021).
54. Пермское отделение Научного Совета Российской Академии Наук по методологии искусственного интеллекта [Электронный ресурс] // permai.ru: информ. портал. 2019. URL: http://www.permai.ru/ (дата обращения: 12.05.2021).
55. Новые продукты с экспертным опытом. Что такое DevOps? [Электронный ресурс] // itexpert.ru: информ. портал. 2021. URL: https://www.itexpert.ru/rus/biblio/detail.php?ID=16167 (дата обращения: 12.05.2021).
56. Азаренко Н. Kanban: принципы и возможности в управлении проектами [Электронный ресурс] // unisender.com: информ. портал. 2020. 7 сентября. URL: https://www.unisender.com/ru/support/about/glossary/kanban/ (дата обращения: 12.05.2021).

57. Повысьте эффективность работы сотрудников с помощью интеллектуального ассистента [Электронный ресурс] // dinabot.com: информ.-сервисный портал. 2020. URL: https://dinabot.com/ (дата обращения: 12.05.2021).
58. Personalized Project Management through AI [Электронный ресурс] // facebook.com: информ. портал. 2018. URL: https://www.facebook.com/pmottoai/ (дата обращения: 12.05.2021).
59. Assistant for tech teams using Slack [Электронный ресурс] // stratejos.ai: информ.-сервисный портал. 2018. URL: https://stratejos.ai/ (дата обращения: 12.05.2021).
60. Artificial Intelligence reinvents Project Management: Lili at Moniteur Innovation Day 19 October [Электронный ресурс] // wordpress.com: информ.-сервисный портал. 2020. 15 октября. URL: https://liliai.wordpress.com/ (дата обращения: 12.05.2021).
61. Знакомьтесь! Чат-бот «Иван из Проектной ПРАКТИКИ» [Электронный ресурс] // pmpractice.ru: информ. портал. 2018. 9 февраля. URL: https://blog.pmpractice.ru/2018/02/09/znakomtes-chat-bot-ivan-iz-proektno/ (дата обращения: 12.05.2021).
62. Искусственный интеллект от Autodesk поможет бороться с рисками строительных проектов [Электронный ресурс] // autodesk.ru: информ.-сервисный портал. 2019. 25 марта. URL: https://www.autodesk.ru/press-releases/2019-03-25 (дата обращения: 12.05.2021).
63. Битрикс24 Ассистент [Электронный ресурс] // bitrix24.ru: информ.-сервисный портал. 2021. URL: https://www.bitrix24.ru/apps/?app=bitrix.assistant (дата обращения: 12.05.2021).
64. Бизнес Ассистент от Сбербанка [Электронный ресурс] // bitrix24.ru: информ.-сервисный портал. 2021. URL: https://www.bitrix24.ru/apps/?%20app=pao_sberbank.app1 (дата обращения: 12.05.2021).
65. Smart Projects Интеллектуальная система управления проектами [Электронный ресурс] // kg.ru: информ.-сервисный портал. 2021. URL: http://www.kg.ru/solutions/smart-projects/ (дата обращения: 12.05.2021).
66. Aurora is the most advanced Intelligent Planning and Scheduling Solution available on the market today [Электронный ресурс] // stottlerhenke.com: информ.-сервисный портал. 2021. URL:

https://www.stottlerhenke.com/products/aurora/ (дата обращения: 12.05.2021).
67. LiquidPlanner - Planning intelligence for smart projects [Электронный ресурс] // liquidplanner.com: информ.-сервисный портал. 2021. URL: https://www.liquidplanner.com/ (дата обращения: 12.05.2021).
68. Nia Contracts Analysis [Электронный ресурс] // edgeverve.com: информ.-сервисный портал. 2021. URL: https://www.edgeverve.com/artificial-intelligence/nia/nia-contracts-analysis/ (дата обращения: 12.05.2021).
69. Think of the best team you've ever been on. How do you create that magic? [Электронный ресурс] // cloverleaf.me: информ.-сервисный портал. 2021. URL: https://cloverleaf.me/ (дата обращения: 12.05.2021).
70. Stay tuned with your team for better collaboration [Электронный ресурс] // pinestem.com: информ.-сервисный портал. 2019. URL: https://pinestem.com/ (дата обращения: 22.11.2020).
71. Botmother – конструктор чат-ботов для Telegram, WhatsApp, Viber, Facebook, ВКонтакте и Одноклассников [Электронный ресурс] // botmother.com: информ.-сервисный портал. 2021. URL: https://botmother.com/ru (дата обращения: 12.05.2021).
72. Botkit - Building Blocks for Building Bots [Электронный ресурс] // botkit.ai: информ.-сервисный портал. 2021. URL: https://botkit.ai/ (дата обращения: 12.05.2021).
73. Aimylogic: powerful and easy-to-use AI bot builder [Электронный ресурс] // aimylogic.com: информ.-сервисный портал. 2021. URL: https://aimylogic.com/ (дата обращения: 12.05.2021).
74. Botsify: Fully Managed Chatbot Platform [Электронный ресурс] // botsify.com: информ.-сервисный портал. 2021. URL: https://botsify.com/ (дата обращения: 12.05.2021).
75. Бесплатный онлайн-курс «Машинное обучение» [Электронный ресурс] // Академия искусственного интеллекта для школьников. 2021. URL: https://ml.ai-academy.ru/ (дата обращения: 12.05.2021).
76. Топ-10 профессий, где нас заменят роботы [Электронный ресурс] // Журнал для абитуриентов: старшеклассникам и поступающим в вузы. Поступи онлайн. 2016. 23 июня. URL: https://postupi.online/journal/kem-stat/top-10-professiy-gde-nas-zamenniat-roboti/ (дата обращения: 12.05.2021).

77. 9 профессий, в которых искусственный интеллект и роботы скоро заменят людей [Электронный ресурс] // ferra.ru (наука и технологии): информ. портал. 2017. 4 апреля. URL: https://www.ferra.ru/review/techlife/jobs-that-soon-will-be-taken-by-ai-and-robots.htm (дата обращения: 12.05.2021).
78. Ракова Ю. Год 2033: Искусственный интеллект и мёртвые профессии [Электронный ресурс] // vc.ru: информ. портал. 2019. 6 августа. URL: https://vc.ru/future/78287-god-2033-iskusstvennyy-intellekt-i-mertvye-professii (дата обращения: 12.05.2021).
79. «Ревность пахнет носками»: как ощущают мир синестеты [Электронный ресурс] // afisha.ru: информ. портал. 2019. 6 ноября. URL: https://daily.afisha.ru/relationship/13437-revnost-pahnet-noskami-kak-oschuschayut-mir-sinestety/ (дата обращения: 12.05.2021).
80. Пироговская М. Курс №61. Антропология чувств [Электронный ресурс] // Европейский университет в Санкт-Петербурге. 2021. URL: https://arzamas.academy/courses/61/1 (дата обращения: 12.05.2021).
81. Ястребова С. Магниторецепция у человека и других животных: новые данные, новые сомнения [Электронный ресурс] // Элементы. 2019. 5 апреля URL: https://elementy.ru/novosti_nauki/433458/Magnitoretseptsiya_u_cheloveka_i_drugikh_zhivotnykh_novye_dannye_novye_somneniya (дата обращения: 12.05.2021).
82. Научный проект «Интеллектуальная поисковая система Nigma.ru» МГУ им. М.В. Ломоносова [Электронный ресурс] URL: https://www.nigma.net.ru (дата обращения: 12.05.2021).
83. Научная электронная библиотека eLIBRARY.RU [Электронный ресурс] URL: https://www.elibrary.ru (дата обращения: 12.05.2021).
84. Google Академия [Электронный ресурс] URL: https://scholar.google.ru (дата обращения: 12.05.2021).
85. ScienceDirect. Search for peer-reviewed journal articles and book chapters (including open access content) [Электронный ресурс] URL: https://www.sciencedirect.com (дата обращения: 12.05.2021).
86. Web of Science. Крупнейший в мире независимый от издательств указатель цитирования и интеллектуальная платформа для исследований [Электронный ресурс] URL: https://login.webofknowledge.com (дата обращения: 12.05.2021).

87. Лучшие поисковые системы [Электронный ресурс] // spbu.ru: информ. портал. URL: http://distolymp2.spbu.ru/www/search/engines_r.html (дата обращения: 12.05.2021).
88. Все коттеджные поселки Ростова-на-Дону на одном сайте [Электронный ресурс] // zemlebaza.ru: информ.-поисковый портал. 2021. URL: https://rostov.zemlebaza.ru/ (дата обращения: 12.05.2021).
89. Коттеджные поселки Ростова-на-Дону [Электронный ресурс] // rostov-dom.info: информ. портал. 2021. URL: http://rostov-dom.info/kottedzhnye-poselki-rostova-na-donu/ (дата обращения: 12.05.2021).
90. Коттеджные поселки в Ростове и Ростовской области [Электронный ресурс] // alfaplan.ru: информ. портал. 2018. 18 февраля. URL: https://rostov.alfaplan.ru/statii/v-rostove/kottedzhnye-poselki-v-rostove-i-rostovskojj-oblasti/ (дата обращения: 12.05.2021).
91. Уотсон Р. Таймлайн трендов и технологий 2010+ [Электронный ресурс] URL: https://cherepkov.com/files/trends_and_technology_timeline_2010.pdf (дата обращения: 12.05.2021).
92. Шуйский А. Telegram-боты в бизнесе: примеры использования [Электронный ресурс] // cossa.ru: информ. портал. 2018. 25 октября. URL: https://www.cossa.ru/trends/224036/ (дата обращения: 12.05.2021).
93. Чат-бот для сервисного центра по ремонту электроники [Электронный ресурс] // chatbullet.com: информ. портал. 2020. 1 июля. URL: https://chatbullet.com/tpost/un6vk2p8pg-chat-bot-dlya-servisnogo-tsentra-po-remo (дата обращения: 12.05.2021).
94. Создание чат-бота для службы доставки питьевой воды [Электронный ресурс] // 4limes.com: информ. портал. 2021. URL: https://4limes.com/sozdanie-chat-bota-dlya-sluzhby-dostavki-pitevoy-vody (дата обращения: 12.05.2021).
95. Holy water. Вода в офис и на дом в Харькове [Электронный ресурс] // facebook.com: информ. портал. 2019. 27 сентября. URL: https://www.facebook.com/Holy-water-Вода-в-офис-и-на-дом-в-Харькове-113852729999360/ (дата обращения: 12.05.2021).

ПРИЛОЖЕНИЕ 1
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 1

Официально считается, что рождение искусственного ИИ как научного направления, произошло в 40-х годах XX века после создания ЭВМ. В 1956 году был предложен сам термин «искусственный интеллект» – свойство интеллектуальных систем выполнять творческие функции, которые традиционно считаются прерогативой человека.

После признания ИИ отдельной областью науки в нём сформировались два основных направления: нейрокибернетика и «кибернетика черного ящика». Эти направления развивались практически независимо, существенно различаясь как в методологии, так и в технологии. И только в настоящее время части ИИ стали объединяться в единое целое.

Основной ориентир нейрокибернетики – программно-аппаратное моделирование структур, подобных структуре мозга. Физиологами давно установлено, что основой человеческого мозга является большое количество связанных между собой и взаимодействующих нервных клеток – нейронов. Поэтому усилия нейрокибернетики были сосредоточены на создании элементов, аналогичных нейронам, и их объединении в функционирующие системы. Эти системы принято называть нейронными сетями, или нейросетями.

На сегодняшний день можно выделить три подхода к созданию нейросетей: 1) аппаратный, 2) программный и 3) гибридный.

В основу подхода «черного ящика» положен принцип, противоположный нейрокибернетике, который ориентирован на поиски алгоритмов решения интеллектуальных задач на существующих моделях компьютеров. Кибернетики начали создавать собственные модели и разрабатывать собственные технологии. Так последовательно были созданы и опробованы следующие решения ИИ:

– в конце 50-х годов родилась модель лабиринтного поиска;

– начало 60-х годов стало эпохой эвристического программирования – разработки стратегии действий на основе известных, заранее заданных эвристик.

– в 1963–1970 годы к решению интеллектуальных задач стали подключать методы математической логики. Был разработан метод резолюций, который позволяет автоматически доказывать теоремы при наличии набора исходных аксиом. На основе метода резолюций был создан язык логического программирования «ПРОЛОГ».

Примерно в это же время существенный прорыв в развитии практических приложений ИИ произошел в США, когда на смену поискам универсального алгоритма мышления пришла идея моделировать конкретные знания специалистов-экспертов. Так появились первые коммерческие системы, основанные на знаниях, или экспертные системы (ЭС). Появилась новая технология решения задач ИИ – представление знаний. В это время были созданы две первые экспертные системы для медицины и химии – MYCIN и DENDRAL, которые впоследствии стали классическими.

Начиная с середины 1980-х годов, повсеместно происходит коммерциализация ИИ. Растут ежегодные капиталовложения, создаются промышленные экспертные системы. Растет интерес к самообучающимся системам

На практике ни одно решение ИИ невозможно реализовать без использования соответствующих данных и алгоритмов. В начале исторического развития ИИ больше внимания уделялось разработке новых алгоритмов. Сегодня же, когда объемы накопленных человеком данных стали превышать осязаемые границы, большую важность стали приобретать методы хранения, структурирования и обработки этих «больших данных».

ПРИЛОЖЕНИЕ 2
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 2

Большинство из нас уже давно пользуются ИИ, даже не осознавая этого. При взаимодействии со смартфоном ИИ занимается распознаванием речи. Он же предупреждает нас о входящих спам-звонках. ИИ фильтрует поступающие в наш почтовый ящик электронные письма, а также решает множество иных «незаметных» задач.

Несмотря на широкое разнообразие областей применения ИИ, он чаще всего используется на уровне исправлений и рекомендаций. Исправление не предполагает обязательного предоставления ответа на ошибку, а рекомендация – обязательного ответа на вопрос.

Исправления могут принимать разные формы и необязательно означать, что ошибка произошла или произойдет в будущем. Например, ИИ автомобильного навигатора способен вносить микро-коррекции, чтобы помочь водителю соблюдать границы безопасности. Однако, водитель может вести машину и самостоятельно.

Рекомендация – это идея, выдвинутая как возможное решение задачи. Рекомендация подразумевает возможность и других решений, а принятие рекомендации не означает её неизбежной реализации (она может даже не сработать). Наиболее распространенный способ использования ИИ для создания рекомендаций подразумевает сбор данных о действиях при предыдущих событиях и выработку новых рекомендаций на их основе.

Гибкость приложений ИИ означает то, что некоторые из них могут использоваться в тех сферах, для которых они первоначально даже не предназначались. Из множества технологий ИИ можно выделить те, которые в настоящее время вызывают наибольший интерес у практиков и исследователей. Среди них следующие:

– понимание естественного языка;
– машинный перевод;
– интеллектуальные базы данных;
– экспертные системы;
– автоматическое доказательство теорем;

– планирование действий робота;
– распознавание образов;
– интеллектуальные игры.

Помимо перечисленных интеллектуальных технологий существует множество зародившихся совсем недавно и сегодня активно развивающихся. Стоит лишь взглянуть на основные рубрикаторы конференций по ИИ, чтобы понять, насколько широко простирается область применения современных технологий ИИ: эволюционные алгоритмы, когнитивное моделирование, интеллектуальные интерфейсы, мультиагентные системы, менеджмент знаний, мягкие вычисления и многое другое.

По мере роста интеллекта ИИ немаловажным фактором становятся действия дружественного ИИ (ДИИ), совместимого с общим искусственным интеллектом (ОИИ), оказывающим положительное влияние на человечество. При этом важно отметить, что задачи ОИИ могут не иметь ничего общего с человеческой этикой. Это сегодня вызывает тревогу. Однако интеллект ДИИ обладает логикой, гарантирующей соответствие задач ИИ человеческим. Последнее определяется на уровне трёх законов робототехники А. Азимова:

1) робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинён вред;

2) робот должен повиноваться всем приказам, которые даёт человек, кроме тех случаев, когда эти приказы противоречат Первому Закону;

3) робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому или Второму Законам.

Однако многие утверждают, что эти три закона лишь хорошая отправная точка и что необходима перманентная разработка дополнительных гарантий.

ПРИЛОЖЕНИЕ 3
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 3

Сравнительно недавно (в 2006 году) британский математик Клайв Хамбли приравнял данные к нефти. При этом он отметил, что подобно тому, как нельзя непосредственно использовать неочищенную нефть, данные предварительно следует проанализировать и переработать, чтобы они приобрели значимость. Анализ данных, в зависимости от контекста, сводится к большому количеству возможных операций, иногда специфичных для конкретной отрасли или задачи. Однако, все преобразования можно отнести к четырем основным категориям.

1. Преобразование. Данные переводятся в матричный формат и подвергаются специальной числовой обработке (масштабирование, вычисление среднего, минимального и максимального значений и т.п.).

2. Чистка. Исправляются дефектные данные: устраняются проблемы с отсутствием информации, выбросами значений из диапазона или просто неправильными значениями.

3. Проверка данных. Предполагает выполнение множества статистических выкладок и графических представлений, позволяющих сразу ухватить информативное содержимое.

4. Моделирование. Основано на выявлении отношений между элементами в данных посредством статистических инструментов (например, корреляции, линейной регрессии и т.п.), позволяющих определить, наличие/отсутствие взаимосвязей между данными.

Апофеозом анализа данных является машинное обучение. Однако успешное применение машинного обучения возможно только после того, как выполнен анализ данных. Главная идея машинного обучения такова: «Можно представить действительность, используя математические функции, которые алгоритму заранее неизвестны, но которые он может предположить на основании наблюдения некоторых данных». Эта идея лежит в основе всех алгоритмов машинного обучения, которые подразделяются на три основные категории.

1. Обучение с учителем. Алгоритм учится на основе имеющихся примеров данных и относящихся к цели ответов так, чтобы впо-

следствии прогнозировать правильные ответы, при предъявлении ему новых (ранее неизвестных) примеров.

2. Обучение без учителя. Алгоритм учится на основе простых примеров без ассоциированных ответов, или явных указаний, что сними делать. Этот тип алгоритма пытается самостоятельно найти зависимости в данных, извлекая полезные признаки и анализируя их.

3. Обучение с подкреплением. Алгоритму предоставляют примеры без ответов, как при обучении без учителя. Но примеры сопровождаются положительной или отрицательной оценкой в зависимости от предлагаемого алгоритмом решения.

Сейчас стало принято приравнивать машинное обучение и нейронные сети и говорить, что традиционные методы постепенно вытесняются последними. Если говорить языком математиков, классическая искусственная нейронная сеть – это сложная композиция функций, в которой есть числовые параметры, настраиваемые на множестве обучающих примеров (обучающей выборке).

Самый современный подход к машинному обучению – нейронные сети и глубокое обучение. При этом следует понимать, что алгоритмы машинного обучения почти всегда требуют структурированных данных, в то время как сети глубокого обучения полагаются на свои слои (классические искусственные нейронные сети).

Глубина нейронной сети позволяет ей построить иерархию объектов с возрастающей абстракцией. При этом каждый последующий слой выступает в качестве фильтра для все более сложных функций, объединяющих функции предыдущего уровня. Эта иерархия функций и фильтры, которые моделируют значимость данных, создаются автоматически, когда глубокие сети учатся восстанавливать данные. Поскольку глубокие нейронные сети могут работать с неупорядоченными данными (которые составляют большинство данных в мире), они могут стать более точными, чем традиционные алгоритмы машинного обучения, которые не могут обрабатывать неупорядоченные/неструктурированные данные. Таким образом, нейронные сети глубоко обучения с доступом к большему количеству данных всегда выигрывают.

ПРИЛОЖЕНИЕ 4
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 4

Современный этап развития ИИ характеризуется широким применением терминов «агент» и «мультиагентная система».

Под агентами понимаются разумные сущности, взаимодействующие с окружающей средой, их поведение рационально – они способны к пониманию. Действия агентов всегда направлены на достижение какой-либо цели. Агент может быть, как роботом, так и встроенной программной системой, которая взаимодействует с окружающей средой примерно так же, как действовал бы человек. Поэтому агентов принято называть интеллектуальными.

Важно отметить, что совокупность из нескольких агентов может решать задачи, которые не под силу каждому из агентов в отдельности. В этом случае говорят, что начинает работать коллектив (рой) взаимодействующих агентов, который принято также называть мультиагентной системой.

Понятие агента на данный момент не является строго формализованным. В общем случае агент – всё то, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов. Агенту присуще обладать всеми или некоторыми из следующих свойств: автономность, способность к общению, реактивность, проактивность, обучаемость и интеллектуальное поведение.

С точки зрения разработчиков информационных систем агент – это модуль программного обеспечения, выполняющийся на определенной платформе, совершающий некоторые действия, такие как отображение и ввод информации, и обменивающийся сообщениями с другими агентами или человеком. Одна из важнейших характеристик агента – наличие ментального поведения.

Если говорить об аппаратном воплощении (разработке и реализации) агентов – робототехнике, то люди привыкли принимать её за ИИ. Однако, робототехника отличается от него. ИИ нацелен на поиск решения некоторых трудных задач, связанных с человеческими способностями (такими, как распознавание объектов, понимание речи и т.п.). Робототехника же стремится использовать машины для решения задач в физическом мире за счет частичной или пол-

ной автоматизации. Поэтому ИИ можно считать программным обеспечением (в виде агента) для решения задач, а робототехнику – аппаратными средствами для того, чтобы сделать эти решения реальностью.

Как было уже отмечено, коллектив интеллектуальных агентов, которые действуют совместно для достижения конкретной цели принято называть мультиагентной системой. Отдельный агент является её элементом (частью). Функциональная целостность мультиагентной системы предполагает такой тип внутреннего взаимодействия её элементов, при котором свойства целого (т.е. всей системы) нельзя свести к сумме свойств элементов. При этом проявляется эмерджентность – возникают качественно новые свойства при объединении элементов (агентов) в систему. Поэтому, считается, что коллективными действиями можно достичь больших результатов чем индивидуальными.

В качестве примера отдельного интеллектуального агента можно привести беспилотный автомобиль (робот-автомобиль) – транспортное средство, оборудованное системой автоматического управления, которое может передвигаться без участия человека. Разработкой беспилотных автомобилей и технологий для них сегодня занимаются более 50-ти компании. Среди наиболее известных такие как *Tesla*, Яндекс, *Mercedes*, *General Motors*, *Toyota*, Сбер, *BMW*, *Volvo*, *Hyundai* и ряд других.

В качестве примера использования мультиагентной системы возможно рассмотреть задачу аэрофотосъемки, выполняемую коллективом дронов (беспилотных летательных аппаратов). В ходе её решении каждый агент-дрон должен чётко согласовывать свои положение и действия с другими агентами-дронами, работающими в одной команде.

ПРИЛОЖЕНИЕ 5
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 5

Данные – совокупность зафиксированных фактов. Информация – сведения, уменьшающие неопределённость. Знания – это сведения, позволяющие действовать с прогнозируемым результатом. Поэтому инженерия знаний сегодня является краеугольным камнем ИИ.

В последние годы разработка онтологий – явное формальное описание терминов предметной области и отношений между ними – переходит из мира лабораторий по ИИ на рабочие столы экспертов по предметным областям. Во всемирной паутине онтологии стали обычным явлением. Во многих дисциплинах сейчас разрабатываются стандартные онтологии, которые могут применяться экспертами по предметным областям для совместного использования и аннотирования информации в своей области.

Обеспечение возможности совместного использования знаний предметной области стало одной из движущих сил недавнего всплеска в изучении онтологий. Например, для моделей многих различных предметных областей необходимо сформулировать понятие времени. Это представление включает понятие временных интервалов, моментов времени, относительных мер времени и т.д. Если одна группа ученых детально разработает такую онтологию, то другие могут просто повторно использовать её в своих предметных областях. Кроме того, если нам нужно создать большую онтологию, мы можем интегрировать несколько существующих онтологий, описывающих части большой предметной области.

В России существенный вклад в развитие идей онтологического моделирования внесен школой профессора Татьяны Альбертовны Гавриловой. Три этапа, лежащие в его основе, определяются следующим образом.

1. Онтологический анализа предметной области – составление словаря терминов, который используется при исследовании характеристик системы и процессов, составляющих рассматриваемую область, а также создания базы данных определений этих терминов.

2. Документирование основных логических взаимосвязей между терминами и понятиями.

3. Дополняется онтологической модели набором вариантов интерпретации совокупности терминов и связей между ними конкретными примерами.

Ещё одним современным средством обработки информации, признанным во всём мире, являются интеллект-карты. Интеллектуальная карта – это схема, наглядно представляющая главную мысль, её основные элементы и взаимосвязи между ними. Методика построения интеллект-карт основана на визуализации и структурировании мышления. Любая интеллект-карта – это дерево. Дерево имеет ствол и ветви, отходящие от него. Чем дальше от ствола, тем тоньше становятся ветви – этот простой принцип визуализации позволяет кратко отобразить ход мыслей в правильном порядке. При этом информация оформляется сразу в сжатом и структурированном виде, легко воспринимается, а основные понятия и категории сразу оказываются выделенными.

При формализации человеческих знаний исследователи зачастую сталкиваются с необходимостью количественного описания качественных характеристик объектов или отношений между ними. Такая интерпретация подобных параметров неоднозначна, а использование традиционного математического аппарата весьма затруднительно. Для преодоления этой проблемы в области ИИ были разработаны различные теории, в том числе математическая теория нечётких множеств и нечёткая логика, автором которых стал профессор калифорнийского университета (г. Беркли) Лотфи Заде. Он опубликовал две свои основополагающие работы: «Нечёткие множества» («*Fuzzy Sets*») в журнале «*Information and Control*» (США, 1965 год) и «Тени нечётких множеств» в журнале «Проблемы передачи информации» (СССР, 1966 год). Эти статьи заложили основы современного моделирования интеллектуальной деятельности человека и стали отправной точкой к развитию новой математической теории. Сегодня её результаты доступны широкой публике в виде большого количества приложений в самых различных предметных областях.

ПРИЛОЖЕНИЕ 6
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 6

В последнее время ИИ развивается такими шагами, что теперь не проходит и месяца без сообщений о прорывах в сфере ИИ. В самых разных областях человеческой деятельности компьютер все чаще начинает превосходить человека.

Прогресс не стоит на месте, и многие технологии уже активно используются. Ставку на ИИ с каждым годом делают всё больше и больше игроков рынка – начиная от крупных ИТ-гигантов и заканчивая небольшими стартапами. Возможно, что о некоторых технологиях вы не видели сообщений по телевизору, но их уровень – высочайший.

Так, в 2018 году в США впервые в мировой истории было приняло решение, разрешающее продавать устройство, диагностирующее под управлением ИИ проблемы со здоровьем в медицинских учреждениях без участия живого специалиста. ИИ разработанного устройства на основе интеллектуального анализа изображений сетчатки глаз пациентов указывал на наличие или отсутствие у них признаков потери зрения, связанных с диабетом.

Программные роботы или «цифровые работники» стали одним из главных технологических трендов последнего десятилетия. Аналитики прогнозируют, что в ближайшие два года количество проектов в области роботизации увеличится на 70%. Программные роботы эффективно экономят рабочее время: наполняют клиентскую базу, обрабатывают несложные финансовые транзакции или отвечают на простые запросы в техническую поддержку.

Человекоподобные роботы *Boston Dynamics* за последние десять научились ходить по снегу, стоять на одной ноге и делать сальто. В ближайшие несколько лет подобных роботов планируют активно использовать при производстве автомобилей, в металлургии и химической промышленности, а ещё для спасения людей при чрезвычайных ситуациях.

В конце ноября 2020 года компания *DeepMind*, входящая в состав исследовательских подразделений *Google*, объявила о решении фундаментальной научной проблемы предсказания трехмерной

структуры любого белка по его аминокислотной последовательности, где эта структура однозначно закодирована.

И это лишь немногие современные достижения ИИ. По мнению ряда ведущих учёных в ближайшее десятилетие в этой области возможны следующие достижения:

1. Создание диагностических систем ИИ, превосходящих по точности постановки диагнозов естественных врачей.

2. Создание систем ИИ для диагностики сложных технических устройств, превосходящих по своим возможностям и точности известные инженерные методики.

3. Создание систем ИИ, способных получать точные аналитические решения краевых задач, что позволит преодолеть современный кризис прикладной математики.

4. Создание систем ИИ, прогнозирующих экономическое состояние предприятий и позволяющих разрабатывать рекомендации по оптимизации их деятельности.

5. Создание систем ИИ, предназначенных для прогнозирования развития политических событий и влияния на эти события.

6. Создание и применение интеллектуальных систем в области криминалистики.

7. Создание систем ИИ, предназначенных для прогнозирования результатов спортивных состязаний и для оптимизации программы подготовки спортсменов с целью получения ими наивысших спортивных результатов.

8. Создание и широкое применение систем ИИ в психологии.

В заключении следует отметить, что, не смотря на радужные перспективы развития ИИ, обусловленные результатами современных исследований, его внедрение в жизнь социума предполагает решение довольно непростых организационных задач. Решение этих задач должно, прежде всего, опираться на первоначальное внедрение ИИ в сферы промышленности, где труд без использования методов ИИ влечет большой риск здоровью человека и где невозможно решение производственных задач без ИИ.

ПРИЛОЖЕНИЕ 7
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 7

В интеллектуальной эволюции общего управления проектами сегодня можно выделить четыре этапа. Начальный этап, для которого характерно первичное решение задач по автоматизации процессов проектного управления. Затем – этап появления виртуальных помощников, позволивших автоматизировать процедуры распознавания речи и текста, организации встреч и контроля прогрессов, подготовки отчетов и актуализации данных в системах. После него – этап интеллектуальных систем управления проектами, предоставивший такие новые инструменты, как прогнозная аналитика проекта, оптимизация планирования, предупреждения о рисках и подбор оптимальной команды. И, наконец, этап ближайшего будущего, который можно охарактеризовать формированием гибридных команд типа «люди + роботы», созданием новых типов интеллектуальных проектов, появлением новых компетенций и, возможно, инструментов автономного управления проектами.

Сейчас мы находимся стыке второго и третьего этапов. Сегодня управление проектами программного обеспечения не может обходиться без участия человека. ИИ может упростить выполнение некоторых постоянно повторяющихся процессов и администрирование. Подобные задачи отвлекают руководителей проектов от решения стратегических вопросов. Тем не менее, применение ИИ для управления проектами ПО – это пока что относительно новое начинание, и оно несет определенные риски. Нужно учитывать, что даже наиболее качественно обученные алгоритмы в интеллектуальных решениях для управления проектами не гарантируют их бесшовную работу, поэтому сотрудникам нужно регулярно следить за данными, которые «потребляет» ИИ, чтобы предотвратить возможное искажение результатов. Однако сегодня ряд виртуальных помощников и инструментов ИИ уже прочно вошли в деятельность руководителей проектов.

Виртуальным помощником принято считать программного агента, который может выполнять задачи для пользователя на основе информации, введенной им, данных о его местонахождении, а также информации, полученной из различных Интернет-ресурсов.

Интеллектуальные инструменты в управлении проектами ПО – это более сложные, чем виртуальные помощники системы, включающих в себя более широкий набор функций (например, формирование оптимальных календарно-сетевых графиков и команд исполнителей, автоматизация комплекса функций планирования и реализации проектов и т.п.).

Помимо этого, сегодня широкую популярность обрели роботы, реализованные в виде программных собеседников или чат-ботов. Чат-боты могут:

– выполнять функции клиентской поддержки в режиме 24/7;

– обеспечивать организацию повторных продаж без ремаркетинга;

– решать задачи накопления и сегментации базы клиентов.

Чат-бот создаётся в три этапа. На первом этапе определяются цели его использования. На втором – выбирается площадка для размещения бота. На третьем – выполняется проектирование и реализация чат-бота.

Простого чат-бота можно сделать самостоятельно на основе конструктора или заказать у фрилансеров. Создание продвинутого чат-бота заказывается у специализированных студий и профессиональных разработчиков. Стоимость разработки бота варьируется в пределах от 400 до 1500 долларов плюс 10÷100 долларов абонентской платы в месяц за его содержание.

В 2019 г. Институт управления проектами *PMI* впервые описал новую компетенцию руководителя проектов – *PMTQ* (*Project management technology quotient*). Она трактуется как навык адаптации, способность управления и интеграции инновационных технологий, увязанных с проектным менеджментом. Примером высокого *PMTQ* является проект, в котором у руководителя проекта есть виртуальные помощники и/или часть задач проекта выполняют роботы. Поэтому для вступления в этап ближайшего будущего в управлении проектами руководителю проекта и команде проекта нужно быть готовыми к созданию и взаимодействию с роботами.

ПРИЛОЖЕНИЕ 8
КРАТКИЙ КОНСПЕКТ ПО ТЕМЕ 8

Многие новшества, основанные на решениях ИИ, плотно вошли в жизнь ведущих ИТ-компаний ещё до начала пандемии. Однако помимо ИТ-компаний ИИ уверенно входит и в проектное управление. Это подтверждается прогнозом аналитиков исследовательской компании *Gartner* о том, что к 2024 году технологии ИИ избавят менеджеров от 69% нагрузки. В дополнение к этому результаты опроса, выполненного Институтом управления проектами PMI, позволили выявить три технологии ИИ, оказывающих сегодня влияние на подавляющее число компаний:

1) системы, основанные на знаниях;
2) машинное обучение;
3) управление принятием решений.

Также PMI продекларированы технологии, влияние которых набирает обороты и будет расти в ближайшие годы:

- экспертные системы;
- глубокое обучение;
- роботизированная автоматизация.

В Интернет существуют прогнозные списки многих профессий (помимо тех, что относятся к управлению проектами), которые исчезнут до 2030 года. Но слово «исчезнут» здесь не совсем к месту: скорее всего, большинство профессий просто трансформируется, отдав монотонные операции роботам.

Существует ряд профессий, которые ИИ заменить не сможет: есть сферы деятельности, где он бессилен (например, социальная сфера, наука и обучение).

Некоторые профессии в области ИИ уже сегодня считаются перспективными. Это когнитивные копирайтеры, онтоинженеры, проектировщики роботов и некоторые иные.

Также существуют и профессии, на появление которых повлиял именно ИИ. К ним относят специалистов по эмпатии, интерпретации алгоритмов, обеспечению устойчивости работы интеллектуальны систем, их локализации и ряд аналогичных.

В каждой индустрии будут требоваться свои особые навыки, но, вероятнее всего, вектор ключевых профессий будет определяться следующей тройкой.

1. Архитекторы автоматизации. Для них важны системное, последовательное мышление и опыт создания алгоритмов процессов (сценариев автоматизации). Эти алгоритмы будут загружаться в программных роботов, которые на их основе будут самообучаться и создавать комплексные, масштабируемые процессы автоматизации.

2. Аналитики *Big Data*. Уже сегодня важны быстрый поиск и обработка больших данных, находящихся в онлайн доступе, а также умение на основе извлеченных инсайтов определять тренды будущего. Именно аналитики будут формировать спрос на развитие прикладных и узкоспециализированных систем ИИ в области *Big Data* для каждой индустрии.

3. Пользователи прикладных систем ИИ. Для специалистов этой категории важно умение взаимодействовать с интерфейсами программных роботов и цифровых платформ в своей индустрии.

Стоит отметить, что внедрение ИИ и его использование требует высокого уровня проектной дисциплины. Как минимум, ИИ работает с данными, которые должны быть достоверными и своевременно предоставленными. Поэтому говорить о скором переходе на технологии ИИ без участия человека пока рано.

Учебное издание

Мохов Василий Александрович
Кузнецова Алла Витальевна

**СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА:
СОВРЕМЕННЫЕ МЕТОДЫ
ПРОГРАММНОЙ ИНЖЕНЕРИИ**

Учебное пособие

Редактор *Н.А. Юшко*

Подписано в печать 27.05.2021
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 8,83. Уч.-изд. л. 9,0. Тираж 300 экз. Заказ 46-0222.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации

Южно-Российский государственный политехнический
университет (НПИ) имени М. И. Платова

С.А. Левшин

АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к лабораторным работам и самостоятельной работе студентов бакалавриата направлений подготовки «Программная инженерия», «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника»

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 51:512(076.5)
ББК 22.1 я73

Рецензент
кандидат технических наук, доцент кафедры
«Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

Левшин С.А.

Архитектура программного обеспечения: методические указания к лабораторным работам и самостоятельной работе студентов бакалавриата направлений подготовки «Программная инженерия», «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника» / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 12с.

В данном издании представлены методические указания к лабораторным работам и самостоятельной работе студентов «Архитектура программного обеспечения». Методические указания содержат исходные данные для построения имитационных моделей работы программно-аппаратных систем различной производительности. Приведены исследуемые параметры, определяющие эффективность программной системы.

Для самостоятельной работы студентов определены вопросы, позволяющие получить дополнительную информацию по курсу. Для оценки знаний студентов по курсу приведен перечень экзаменационных вопросов.

Методические указания предназначены для студентов, обучающихся по программе бакалавриата направления 09.03.04 Программная инженерия, 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника.

УДК 51:512(076.5)

1. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Номер компетенции	Литература
1	Изучение выполнения функциональных задач в одноядерном микропроцессоре	9	Защита отчета	15-20.9	ПК-2, ПК-5, ПК-22	7 [1-4]
2	Изучение выполнения функциональных задач в многоядерном микропроцессоре	9	Защита отчета	15-20.10	ПК-2, ПК-5, ПК-22	7 [1-4]
3	Изучение выполнения функциональных задач в многопроцессорной вычислительной системе с SMP-архитектурой	9	Защита отчета	15-20.11	ПК-2, ПК-5, ПК-22	7 [1-4]
4	Изучение выполнения функциональных задач в многопроцессорной вычислительной системе с MPP-архитектурой	9	Защита отчета	15-20.12	ПК-2, ПК-5, ПК-22	7 [1-4]

Лабораторная работа №1

Изучение выполнения функциональных задач в одноядерном микропроцессоре

Цель работы: разработка имитационной модели работы программного обеспечения (ПО) в одноядерном микропроцессоре.

Программа работы

1. Разработать алгоритм работы имитационной модели ПО вычислительной системы (ВС).
2. Разработать тестовую версию имитационной модели, демонстрирующую работу ПО во всех режимах.
3. Выполнить программную реализацию имитационной модели.
4. Определить время простоя каждой задачи, расчетное и фактическое время выполнения каждой задачи.

Отчет по лабораторной работе должен содержать:

1.Цель работы и исходные данные.
2.Описание модельных переменных.
3.Описание алгоритма работы имитационной модели.
4.Описание интерфейса пользователя.
5.Результаты выполнения программы.
6.Листинг с необходимыми комментариями.
7.Вывод о проделанной работе.

Варианты заданий на лабораторную работу №1

Варианты тактовой частоты МП – 1,5 ГГц, 1,3 ГГц, 700мГц;
Варианты тактовой частоты системной шины – 800мГц, 600мГц, 233мГц;
Количество ядер микропроцессора –1.

Перечень задач выполняемых вычислительной системой:

Тип задачи	Вероят-ность появления задачи, %	Вероятность появления подзадачи, %	Длитель-ность выполнения подзадачи, такт	Приори-тет подзадачи	Вероятность синхрониза-ции данных, %	Кол-во подзадач в цикле
Задача 1	20	-	-	-	-	3
ПЗ 1.1		20	1-5	1	20	
ПЗ 1.2		20	2-10	1	40	
ПЗ 1.3		20	5-20	2	10	
ПЗ 1.4		20	2-15	2	20	
ПЗ 1.5		20	1-5	2	10	
Задача 2	40	-	-	-	-	5
ПЗ 2.1		10	1-25	2	30	
ПЗ 2.2		30	2-20	2	60	
ПЗ 2.3		20	5-30	1	50	
ПЗ 2.4		20	2-15	2	20	
ПЗ 2.5		20	1-25	1	40	
Задача 3	40	-	-	-	-	4
ПЗ 3.1		20	1-15	1	30	
ПЗ 3.2		20	2-20	1	40	
ПЗ 3.3		10	5-25	1	20	

Лабораторная работа №2

Изучение выполнения функциональных задач в многоядерном микропроцессоре

Цель работы: разработка имитационной модели работы программного обеспечения (ПО) в многоядерном микропроцессоре.

Программа работы

1. Разработать алгоритм работы имитационной модели ПО вычислительной системы (ВС).
2. Разработать тестовую версию имитационной модели, демонстрирующую работу ПО во всех режимах.
3. Выполнить программную реализацию имитационной модели.
4. Определить время простоя каждой задачи, расчетное и фактическое время выполнения каждой задачи.

Отчет по лабораторной работе должен содержать:

1.Цель работы и исходные данные.
2.Описание модельных переменных.
3.Описание алгоритма работы имитационной модели.
4.Описание интерфейса пользователя.
5.Результаты выполнения программы.
6.Листинг с необходимыми комментариями.
7.Вывод о проделанной работе.

Варианты заданий на лабораторную работу №2

Варианты тактовой частоты МП – 1,5 ГГц, 1,3 ГГц, 700мГц;
Варианты тактовой частоты системной шины – 800мГц, 600мГц, 233мГц;
Количество ядер микропроцессора до 4.

Перечень задач выполняемых вычислительной системой:

Тип задачи	Вероятность появления задачи, %	Вероятность появления подзадачи, %	Длительность выполнения подзадачи, такт	Приоритет подзадачи	Вероятность синхронизации данных, %	Кол-во подзадач в цикле
Задача 1	40	-	-	-	-	2
ПЗ 1.1		10	1-5	1	20	
ПЗ 1.2		40	2-10	1	40	
ПЗ 1.3		20	5-20	2	10	
ПЗ 1.4		10	2-15	2	20	
ПЗ 1.5		20	1-5	2	10	
Задача 2	30	-	-	-	-	4
ПЗ 2.1		10	1-25	2	30	
ПЗ 2.2		30	2-20	2	60	
ПЗ 2.3		40	5-30	1	50	
ПЗ 2.4		10	2-15	2	20	
ПЗ 2.5		10	1-25	1	40	
Задача 3	30	-	-	-	-	5
ПЗ 3.1		10	1-15	1	30	
ПЗ 3.2		50	2-20	1	40	
ПЗ 3.3		10	5-25	1	20	
ПЗ 3.4		15	2-35	2	30	
ПЗ 3.5		15	1-25	2	20	

Лабораторная работа №3

Изучение выполнения функциональных задач в многопроцессорной вычислительной системе с SMP-архитектурой

Цель работы: разработка имитационной модели работы программного обеспечения (ПО) в многопроцессорной вычислительной системе с SMP- архитектурой.

Программа работы

1. Разработать алгоритм работы имитационной модели ПО вычислительной системы (ВС).

2. Разработать тестовую версию имитационной модели, демонстрирующую работу ПО во всех режимах.

3. Выполнить программную реализацию имитационной модели.

4. Определить время простоя каждой задачи, расчетное и фактическое время выполнения каждой задачи.

Отчет по лабораторной работе должен содержать:

1.Цель работы и исходные данные.

2.Описание модельных переменных.

3.Описание алгоритма работы имитационной модели.

4.Описание интерфейса пользователя.

5.Результаты выполнения программы.

6.Листинг с необходимыми комментариями.

7.Вывод о проделанной работе.

Варианты заданий на лабораторную работу №3

Варианты тактовой частоты МП – 1,5 ГГц, 1,3 ГГц, 700мГц;

Варианты тактовой частоты системной шины – 800мГц, 600мГц, 233мГц;

Количество микропроцессоров до 10;

Перечень задач выполняемых вычислительной системой:

Тип задачи	Вероятность появления задачи, %	Вероятность появления подзадачи, %	Длительность выполнения подзадачи, такт	Приоритет подзадачи	Вероятность синхронизации данных, %	Кол-во подзадач в цикле
Задача 1	40	-	-	-	-	6
ПЗ 1.1		10	1-5	1	20	
ПЗ 1.2		40	2-10	1	40	
ПЗ 1.3		20	5-20	2	10	
ПЗ 1.4		10	2-15	2	20	
ПЗ 1.5		20	1-5	2	10	
Задача 2	30	-	-	-	-	4
ПЗ 2.1		10	1-25	2	30	
ПЗ 2.2		30	2-20	2	60	
ПЗ 2.3		40	5-30	1	50	
ПЗ 2.4		10	2-15	2	20	
ПЗ 2.5		10	1-25	1	40	
Задача 3	30	-	-	-	-	2
ПЗ 3.1		10	1-15	1	30	
ПЗ 3.2		50	2-20	1	40	
ПЗ 3.3		10	5-25	1	20	

ПЗ 3.4		15	2-35	2	30	
ПЗ 3.5		15	1-25	2	20	

Лабораторная работа №4

Изучение выполнения функциональных задач в многопроцессорной вычислительной системе с МРР-архитектурой.

Цель работы: разработка имитационной модели работы программного обеспечения (ПО) в многопроцессорной вычислительной системе с МРР- архитектурой.

Программа работы

1. Разработать алгоритм работы имитационной модели ПО вычислительной системы (ВС).
2. Разработать тестовую версию имитационной модели, демонстрирующую работу ПО во всех режимах.
3. Выполнить программную реализацию имитационной модели.
4. Определить время простоя каждой задачи, расчетное и фактическое время выполнения каждой задачи.

Отчет по лабораторной работе должен содержать:

1.Цель работы и исходные данные.
2.Описание модельных переменных.
3.Описание алгоритма работы имитационной модели.
4.Описание интерфейса пользователя.
5.Результаты выполнения программы.
6.Листинг с необходимыми комментариями.
7.Вывод о проделанной работе.

Варианты заданий на лабораторную работу №4

Варианты тактовой частоты МП – 1,5 ГГц, 1,3 ГГц, 700мГц;
Варианты тактовой частоты системной шины – 800мГц, 600мГц, 233мГц;
Количество микропроцессоров до 10;

Перечень задач выполняемых вычислительной системой:

Тип задачи	Вероятность появления задачи, %	Вероятность появления подзадачи, %	Длительность выполнения подзадачи, такт	Приоритет подзадачи	Вероятность синхронизации данных, %	Кол-во подзадач в цикле
Задача 1	40	-	-	-	-	6
ПЗ 1.1		10	1-5	1	20	
ПЗ 1.2		40	2-10	1	40	
ПЗ 1.3		20	5-20	2	10	

Тип задачи	Вероятность появления задачи, %	Вероятность появления подзадачи, %	Длительность выполнения подзадачи, такт	Приоритет подзадачи	Вероятность синхронизации данных, %	Кол-во подзадач в цикле
ПЗ 1.4		10	2-15	2	20	
ПЗ 1.5		20	1-5	2	10	
Задача 2	30	-	-	-	-	4
ПЗ 2.1		10	1-25	2	30	
ПЗ 2.2		30	2-20	2	60	
ПЗ 2.3		40	5-30	1	50	
ПЗ 2.4		10	2-15	2	20	
ПЗ 2.5		10	1-25	1	40	
Задача 3	30	-	-	-	-	2
ПЗ 3.1		10	1-15	1	30	
ПЗ 3.2		50	2-20	1	40	
ПЗ 3.3		10	5-25	1	20	
ПЗ 3.4		15	2-35	2	30	
ПЗ 3.5		15	1-25	2	20	

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование, – 51,1 часа.

№	Наименование тем	Кол-во часов	Номер компетенции	Литература
1	Системное ПО, прикладное ПО, инструментальное ПО.	9	ПК-2, ПК-5, ПК-22	7 [1-5]
2	Системное программное обеспечение для многопроцессорных вычислительных систем.	9	ПК-2, ПК-5, ПК-22	7 [1-5]
3	Прикладное программное обеспечение: общего назначения, мультимедиа, проблемно-ориентированные, методо - ориентированные, издательские.	9	ПК-2, ПК-5, ПК-22	7 [1-5]
4	Инструментальное программное обеспечение: языки и системы программирования, интегрированные среды программирования, программные комплексы.	9	ПК-2, ПК-5, ПК-22	7 [1-5]
5	Классификация операционных систем: графические и неграфические, однопользовательские и многопользовательские, однозадачные и многозадачные.	9	ПК-2, ПК-5, ПК-22	7 [1-5]
6	Промежуточное ПО	7	ПК-2, ПК-5, ПК-22	7 [1-5]

Задания для самостоятельного изучения и конспектирования

Тема 1. Рассмотреть следующие вопросы:
- системное ПО;
- прикладное ПО;
- инструментальное ПО.

Тема 2. Рассмотреть следующие вопросы:
Системное программное обеспечение для многопроцессорных ВС:
- базовое ПО;
- сервисное ПО.

Тема 3. Рассмотреть следующие вопросы:
Прикладное программное обеспечение:
- общего назначения;
- мультимедиа;
- проблемно-ориентированные;
- методо – ориентированные;
- издательские.

Тема 4. Рассмотреть следующие вопросы:
Инструментальное программное обеспечение:
- языки и системы программирования;
- интегрированные среды программирования;
- программные комплексы.

Тема 5. Рассмотреть следующие вопросы:
Классификация операционных систем:
- графические и неграфические;
- однопользовательские и многопользовательские;
- однозадачные и многозадачные.

Тема 6. Рассмотреть следующие вопросы:
- архитектура промежуточного ПО.

Контактная внеаудиторная работа

СРС – групповые консультации с преподавателем в течении семестра – 0,9 часа.

СРС экз. - групповая консультация перед экзаменом – 2 часа.

СРС экз. – сдача экзамена – 0,35 часа.

Экзаменационные вопросы по курсу

1. Виды современного программного обеспечения.
2. Характерные черты современных программных систем.
3. Классификация операционных систем.
4. Архитектура операционных систем.
5. Типы драйверов.
6. Типы антивирусных программ.
7. Типы архиваторов.
8. Классификация прикладных программ.
9. ПО общего назначения.
10. ПО системы визуализации.
11. Проблемно-ориентированное ПО.
12. Характеристики, свойства и качества программных продуктов.
13. Цель программирования.
14. Надежность программного средства.
15. Программная инженерия как технология разработки программных средств.
16. Жизненный цикл программного средства.
17. Особенности процесса разработки программного обеспечения.
18. Процедурная разработка.
19. Особенности этапа проектирования.
20. Модель хранилища данных.
21. Модель клиент-сервер.
22. Трехуровневая модель.
23. Модель абстрактной машины.
24. Модель централизованного управления.
25. Модель событийного управления.
26. Модель потока данных.
27. Модель объектов.
28. Модульность.
29. Информационная закрытость.
30. Определение связности модуля.
31. Правило параллельной цепи.
32. Правило последовательной цепи.
33. Сцепление модулей.

34.Степенью сцепления.
35.Сложность программной системы.
36.Характеристики иерархической структуры программной системы.
37.Качество ПО.
38.Функциональность ПО.
39.Лёгкость применения ПО.
40.Эффективность ПО.
41.Сопровождаемость ПО.
42.Мобильность ПО.
43.Обеспечение надёжности ПО.
44.Самообнаружение ошибки ПО.
45.Обеспечение независимости компонент ПО.
46.Контроль принимаемых решений.
47.Смежный контроль ПО.
48.Внешнего описания ПО.
49.Определение требований к программному средству.
50.Спецификация качества программного средства.
51.Функциональная спецификация программного средств.
52.Методы контроля внешнего описания программного средства.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Кузнецов А.С., Ченцов С.В., Царев Р.Ю. Многоэтапный анализ архитектурной надежности и синтез отказоустойчивого программного обеспечения сложных систем [Электронный ресурс]: Монография, Сибирский федеральный университет, - 2013, - 143 с - **режим доступа:** http://www.knigafund.ru.
2. Сафонов В.О. Основы современных операционных систем [Электронный ресурс]: Интернет-университет Информационных Технологий, 2011г.,584. - режим доступа: http://www.knigafund.ru/.

Дополнительная учебная литература

3. Гунько А.В. Системное программное обеспечение [Электронный ресурс]: Электрон. дан. – НГТУ 2011. – 138 с. – Режим доступа: http://www.knigafu/books/185306.

4. Курячий Г.В. Операционная система UNIX [Электронный ресурс]: Интернет-университет Информационных Технологий, 2004г.,288. - режим доступа http://www.knigafund.ru/
5. Операционная система Microsoft Windows XP [Электронный ресурс]: Национальный открытый университет «Интуит», 2016г., 375. - режим доступа http://www.knigafund.ru/

Учебно-методическое издание

Левшин Сергей Афанасьевич

Архитектура программного обеспечения

**Методические указания
к лабораторным работам
и самостоятельной работе студентов
бакалавриата направлений подготовки
«Программная инженерия»
«Математическое обеспечение и администрирование
информационных систем»
«Информатика и вычислительная техника»**

Редактор *Н.А. Юшко*
Подписано в печать 20.09.2016
Формат 60х84 1/16. Бумага офсетная. Печать цифровая.
Усл. печ. л.0,7. Уч.-изд.л.0,75. Тираж 50 экз. Заказ _____.

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166

idp-npi@mail.ru

Министерство образования и науки Российской Федерации

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова

Д.В. Гринченков, С.А. Селезнев, Д.Н. Кущий

ДИСКРЕТНАЯ МАТЕМАТИКА

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 517.8 (075.8)
ББК 22.17
Г35

Рецензенты:

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

кандидат технических наук, профессор, профессор кафедры
«Программное обеспечение вычислительной техники»
Иванченко Александр Николаевич

Гринченков Д.В., Селезнев С.А., Кущий Д.Н.

Г35 Дискретная математика: учебное пособие / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова.– Новочеркасск: ЮРГПУ (НПИ), 2016. – 80 с.

В пособии представлена информация о базывоых математических понятиях, таких как: множества, соответствия, функции и отношения на множествах. Описаны элементы общей алгебры. Приведены основные понятия теории графов. Рассмотрены деревья, Эйлеровы графы, транспортные сети и задача раскраски графов. Отдельный раздел посвящен решению задач нахождения кратчайших путей в графе с использованием различных алгоритмов.

УДК 517.8 (075.8)

1. ТЕОРИЯ МНОЖЕСТВ

1.1. Основные понятия теории множеств

1.1.1. Множества, способы задания множеств

Под ***множеством*** понимают объединение в одно целое объектов, хорошо различимых нашей интуицией или нашей мыслью. Другими словами, ***множество*** — это совокупность объектов любой природы, рассматриваемая как единое целое. Обычно множества обозначают прописными латинскими буквами $A, B, M, \ldots$.

Пример 1.1. $N = \{1, 2, 3, \ldots\}$ – множество натуральных чисел.

Объекты, образующие множество, называются ***элементами*** множества (обозначаются маленькими буквами). Если элемент a входит во множество A, то это обозначается так: $a \in A$. Запись $a \notin A$ означает, что элемент a не принадлежит множеству A. Множество, содержащее конечное число элементов, называется ***конечным*** (в противном случае – ***бесконечным***). Если множество A конечно, то число его элементов называется ***мощностью*** множества и обозначается $|A|$. Если множество не содержит ни одного элемента, то оно называется ***пустым*** множеством и обозначается символом $\varnothing$. Множество A является подмножеством множества B, если любой элемент A принадлежит также множеству B. Это свойство обозначается $A \subseteq B$ (читается: B включает или равно A).

Равенство множеств. Множества A и B равны тогда и только тогда, когда их элементы совпадают. В этом случае пишут $A = B$. Так как при равенстве множеств A и B во множестве A нет элементов, не принадлежащих B, а в B нет элементов не принадлежащих A, то признаком равенства множеств является одновременное выполнение двух условий: $A \subseteq B$ и $B \subseteq A$.

Если $A \subseteq B$ и $B \neq A$, то множество A называется ***собственным подмножеством*** множества B. Обозначается $A \subset B$ (строгое включение).

Одним из частных случаев является ситуация, когда элементами некоторого множества являются другие множества.

Например, пусть A– множество футболистов команды «Спартак», B– множество команд высшей лиги. Тогда $A \in B$.

Если в рамках некоторого класса задач рассматриваются различные множества, то полная совокупность всех элементов, из которых могут формироваться все множества и подмножества, образует универсальное множество – “Универсум” или полное пространство. Обозначается универсальное множество символом U (генеральная совокупность).

Множество может быть задано:

1. Перечислением всех его элементов. Например,

$$A = \{a, b, c, d\},\ B = \{0, 1, 3, 8, 9\}.$$

2. Порождающей процедурой. ***Порождающая процедура*** представляет собой правило получения элементов множества на основе уже имеющихся элементов либо из других объектов. Элементами множества считаются все объекты, которые получены с помощью этой процедуры.

3. Описанием характеристик и свойств, которыми обладают все элементы множества. Например, $A = \{x |\ x \in N, x \le 100\}$.

1.1.2. Основные операции над множествами и их свойства

1. ***Объединение множеств*** $(A \cup B)$ – это множество, состоящее из тех элементов, которые принадлежат хотя бы одному из исходных множеств: $A \cup B = \{x |\ x \in A$ или $x \in B\}$.

Пример 1.2. $A = \{a, b, c\},\ B = \{b, c, d, m\},\ A \cup B = \{a, b, c, d, m\}$.

Операции над множествами удобно представлять с помощью диаграммы Венна – замкнутой линии, ограничивающей элементы одного множества. Для операции объединения диаграмма представлена на рис.1.1.

2. ***Пересечение множеств*** $(A \cap B)$ – это множество тех и только тех элементов, которые принадлежат и множеству A, и множеству B(рис.1.2): $A \cap B = \{x |\ x \in A$ и $x \in B\}$.

Пример 1.3. $A = \{a, b, c\},\ B = \{b, c, d, m\},\ A \cap B = \{b, c,\}$.

3. ***Разность множеств*** $(A \setminus B)$ – это множество, состоящее из тех и только тех элементов множества A, которые не содержатся во множестве B (рис.1.3): $A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}$. Если $A \setminus B = \varnothing$, *то* $A \subseteq B$ (рис.1.4).

4. ***Симметричная разность*** $(A \div B)$ – это множество элементов, принадлежащих множествам A или B за исключением их общих элементов (рис.1.5): $A \div B = \{x \mid (x \in A \text{ и } x \notin B) \text{ или } (x \in B \text{ и } x \notin A)\}$

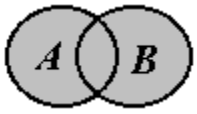


Рис.1.1

Рис.1.2

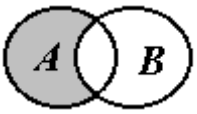


Рис.1.3

Рис.1.4

Рис.1.5

Рис.1.6

5. ***Дополнением*** множества A до множества U (обозначается $\overline{A}$) называется множество всех элементов U, не принадлежащих множеству A (рис.1.6).

Основные свойства операций над множествами. Для всех множеств A, B, C и универсального множества U справедливы следующие равенства:

1. $A \cup B = B \cup A$
2. $A \cup (B \cup C) = (A \cup B) \cup C$
3. $A \cap B = B \cap A$
4. $A \cap (B \cap C) = (A \cap B) \cap C$
5. $A \cap A = A$
6. $A \cup A = A$
7. $A \cup \varnothing = A$
8. $A \cap \varnothing = \varnothing$
9. $A \cup U = U$
10. $A \cap U = A$
11. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
12. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
13. $\overline{\overline{A}} = A$
14. $\overline{\varnothing} = U$
15. $\overline{U} = \varnothing$
16. $A \cup \overline{A} = U$
17. $A \cap \overline{A} = \varnothing$
18. $\overline{A \cup B} = \overline{A} \cap \overline{B}$
19. $\overline{A \cap B} = \overline{A} \cup \overline{B}$

1.2. Прямое произведение множеств

Вектором *(кортежем)* называется упорядоченный набор элементов. Элементы, образующие вектор, называются ***координатами*** или ***компонентами***. Координаты нумеруются слева направо. Число координат называется ***длиной*** или ***размерностью*** вектора. В отличие от элементов множества координаты вектора могут совпадать. Обозначение вектора: (a,b,c), где a,b,c – координаты вектора. ***Два вектора равны,*** если они имеют одинаковую длину и равны их соответствующие координаты.

Прямым или ***декартовым произведением*** множеств A и B (обозначается $A\times B$), называется множество всех упорядоченных пар (a,b) таких, что $a\in A$ и $b\in B$, т.е. $A\times B=\{(a,b)\mid a\in A; b\in B\}$.

Частный случай: $A=B$. Тогда $A\times B=A\times A$. $A\times A$ обозначается A^2.

Прямым произведением $A_1\times A_2\times A_3\times\ldots\times A_n$ называется множество всех векторов $(a_1,a_2,a_3,\ldots,a_n)$ длины n таких, что $a_1\in A_1;\ a_2\in A_2;\ldots;\ a_n\in A_n$.

Пример 1.4. 1) $A_1=\{0,1\},\ A_2=\{x,y,z\}$

$$A_1\times A_2=\{(0,x),(0,y),(0,z),(1,x),(1,y),(1,z)\};$$

2) $A=\{a,b,c,d,e,f,g,h\},\ B=\{1,2,3,4,5,6,7,8\}$

$$A\times B=\{(a,1),(a,2),\ldots,(h,7),(h,8)\}.$$

Пусть имеется множество A, элементы которого являются символами (буквы, цифры, знаки). Такое множество называется ***алфавитом***. Элементы множества A^n – слова. Множество $R\times R=R^2$ – это множество точек (пар координат) плоскости (здесь R – множество всех действительных чисел).

Теорема 1.1. Пусть A_1, A_2, ... , A_n – конечные множества и их мощности известны: $|A_1|=m_1, |A_2|=m_2,\ldots,|A_n|=m_n$. Тогда

$$|A_1\times A_2\times\ldots\times A_n|=m_1\cdot m_2\cdot\ldots\cdot m_n.$$

Частный случай: $|A^n|=|A|^n$.

Проекцией вектора $A=(a_1, a_2, ..., a_n)$ на ось i (обозначается $пр_i A$) называется его компонента a_i. ***Проекцией вектора*** A на оси $i_1 ... i_k$ $(пр_i A)$ называется вектор $(a_{i_1}, a_{i_2}, ..., a_{i_k})$ длины k.

Если $V-$ множество векторов A_j одинаковой длины, то проекцией V на *i-ю* ось называется *множество проекций* всех A_j *на эту ось*:
$пр_i V = \{пр_i A_j \mid A_j \in V\}$.

1.3. Соответствия

1.3.1. Понятие соответствия. Типы соответствий

Соответствием между множествами A и B называется подмножество декартового произведения A и B, включающее совокупность тех пар элементов множеств A и B, которые связаны между собой (элементы одного множества соответствуют элементам другого множества). Обозначение соответствия: $G \subseteq A \times B$.

Если вектор $(a, b) \in G$, то говорят, что b соответствует a при соответствии G.

Частным случаем соответствия является функция. Подмножество $F \subset M_x \times M_y$ называется ***функцией,*** если для каждого $x \in M_x$ находится не более одного значения $y \in M_y$, при этом вектор $(x, y) \in F$.

Область определения и область значения. Совокупность всех элементов первого множества, входящих в соответствие $(пр_1 G)$, называется ***областью определения*** соответствия.

Совокупность всех элементов второго множества, входящих в соответствие $(пр_2 G)$, называется ***областью значений*** соответствия.

Множество всех $b \in B$, соответствующих элементам $a \in A$, называется ***образом*** A в B. Множество всех $a \in A$, соответствующих $b \in B$, называется ***прообразом*** B в A при соответствии G.

Типы соответствий. Если $пр_1 G = A$, то такое соответствие называется ***полностью определенным,*** в противном случае соответствие называется ***частично определенным***.

Если $пр_2 G = B$, то такое соответствие называется ***сюръективным***.

Если образом любого элемента a_i из $пр_1 G$ является единственный элемент b_i из $пр_2 G$, то соответствие G называется ***однозначным*** или ***функциональным***. Пример не функционального соответствия – множество точек круга: $G = \left\{ (x, y) \middle| (x-3)^2 + (y-3)^2 \le 1 \right\}$.

Соответствие называется ***взаимно однозначным***, если:

1) оно полностью определено;

2) оно сюръективно;

3) прообразом b_i является единственный элемент a_i .

1.3.2. Связь мощности множеств, входящих в соответствие

Утверждение. Если между двумя множествами A и B имеется возможность установить взаимно однозначное соответствие, то их мощности равны, т.е. $|A| = |B|$. Такие множества называются равномощными. Справедливо и обратное: если множества равномощны, то между их элементами можно установить взаимно однозначное соответствие.

С учетом указанного выше утверждения можно уточнить введенное выше понятие конечного множества. ***Конечное множество*** – это пустое множество или множество, равномощное с множеством целых положительных чисел {1, 2, 3, …, m}, где m не превышает некоторого натурального числа n. Любое множество, которое не является конечным, называют ***бесконечным множеством***. Доказывается, что в каждом бесконечном множестве A имеется собственное подмножество, равномощное всему A. Наличие правильной части, равномощной целому, можно принять за определение бесконечного множества.

Множество, равномощное с множеством всех натуральных чисел, называются ***счетным***. Объединение конечного числа

счетных множеств тоже является счетным множеством. Существует теорема Кантора, в которой утверждается, что множество всех действительных чисел на отрезке [0,1] несчетно.

Мощность множеств, равномощных с множеством всех действительных чисел, называется ***мощностью континуума***. Например, мощность континуума имеет множество всех замкнутых подмножеств n – мерного евклидова пространства. Возможность построить иерархию мощностей дает следующая теорема Кантора: мощность множества всех подмножеств любого непустого множества A больше мощности множества A.

Теорема (о числе всех подмножеств во множестве). Если конечное множество A содержит n элементов (то есть $|A| = n$), то число всех его подмножеств (включая само множество A и его пустое подмножество) равно 2^n.

Доказательство:

1. Пронумеруем все элементы множества A: $A = \{a_1, a_2, ..., a_n\}$.

2. Пусть A_j – некоторое подмножество A. Рассмотрим множество A^*, элементами которого являются все подмножества множества A.

3. Введем множество B^* – множество двоичных векторов (то есть векторов, содержащих только символы *0* и *1*) длиной n, где n – количество координат вектора, определяемых следующим образом: каждому A_j поставим в соответствие вектор $B_j = (v_1, v_2, ..., v_i, ..., v_n)$, где

$$v_i = \begin{cases} 1, & \text{если } a_i \in A_j \\ 0, & \text{если } a_i \notin A_j \end{cases}.$$

Например, если $A = \{a, b, c, d, e, f\}$, то

подмножеству $A_1 = \{a,b,c\}$ соответствует вектор $B_1 = (1, 1, 1, 0, 0, 0)$;

подмножеству $A_2 = \{b,d\}$ соответствует вектор $B_2 = (0, 1, 0, 1, 0, 0)$;

подмножеству $A_3 = \{d\}$ соответствует вектор $B_3 = (0, 0, 0, 1, 0, 0)$.

Очевидно, что между множеством A^* и множеством B^* имеет место

взаимно однозначное соответствие, поэтому $|A^*| = |B^*|$.

Множество B^* может быть представлено как декартово произведение:

$B^* = B \times B \times ... \times B = B^n$, где $B = \{0,1\}$ – двухэлементное множество. $|B| = 2$. Тогда мощность множества B^*

$$|B^*| = |B^n| = |B|^n = 2^n.$$

Отсюда получаем, что $|A^*| = 2^n$, и теорема доказана.

1.4. Функции

1.4.1. Функции и отображения

Функция представляет собой однозначное (или функциональное) соответствие. Если функция f устанавливает соответствие между элементами множеств A и B, то пишут $f: A \to B$ и говорят, что функция имеет тип $A \to B$ или $f(a) = b$.

Каждому элементу из области определения A соответствует единственный элемент из области значений B (a – аргумент, b – значение функции на a).

Задать функцию $f: A \to B$ - это значит:

1) определить и описать множества A и B;

2) задать вычислительную процедуру (действие, которое по любому значению аргумента выдает соответствующее значение функции). Процедура должна однозначно приводить к результату.

Полностью определенная функция типа $f: A \to B$ называется ***отображением*** A в B.

Если при этом каждый элемент B имеет свой прообраз в A (то есть соответствие, образующее функцию, сюръективно), то имеет место отображение A на B (сюръективное отображение). Если $f(A)$ состоит из одного единственного элемента, то эта функция называется ***константой***. Отображение типа $A \to A$ называется ***преобразованием*** множества A.

Функция типа $f: A_1 \times A_2 \times ... \times A_n \to B$ называется *n-местной* функцией. Эта функция имеет n аргументов: $f(a_1, a_2, ..., a_n) = b$.

Равенство функций. Обратная функция. Функции f и q *равны*, если их области определения есть одно и то же множество A и для любого элемента $a \in A$ имеет место равенство $f(a) = q(a)$.

Пусть дано соответствие $G \subseteq A \times B$. Если соответствие $H \subseteq B \times A$ таково, что $(b, a) \in H$ тогда и только тогда, когда $(a, b) \in G$, то соответствие H называется ***обратным*** к G и обозначается $H = G^{-1}$.

Если соответствие H является функциональным, то оно называется ***функцией, обратной*** к f, и обозначается f^{-1}.

Обратная функция существует тогда и только тогда, когда имеет место взаимно однозначное соответствие между областью определения и областью значений функции.

Композиция функций. Суперпозиция функций. Пусть даны две функции: $f: A \to B$ и $q: B \to C$. Функция $h: A \to C$ называется ***композицией*** функций f и q, если для всех $x \in A$ имеет место $h(x) = q[f(x)]$. Говорят, что h получено подстановкой f в q.

Функция, полученная из $f_1, f_2, ..., f_n$ некоторой подстановкой их друг в друга и переименованием аргументов, называется ***суперпозицией функций***.

1.4.2. Способы задания функции

Существуют четыре основных способа задания функций:

1. **Задание функции с помощью таблиц** (таблица – конечный список пар элементов).

Вычислительная процедура заключается в поиске нужной строки таблицы. Такой способ применяется, если множество конечно. Для функций, определенных на бесконечных множествах, таблицы задают их значения в конечном числе точек; для вычисления промежуточных точек используются правила интерполяции.

2. **Задание функции как суперпозиции** других (исходных) функций. Процесс вычисления значения функции осуществляется

последовательным вычислением одних функций и подстановкой их результатов в другие функции.

3. **Задание функции с помощью рекурсивной процедуры.** Рекурсивная процедура задает функцию f следующим образом:

а) задается начальное значение $f(0)$ или $f(1)$;

б) $f(n+1)$ определяется как суперпозиция $f(n)$ и других известных функций.

Пример 1.5. $(n+1)! = n!(n+1)$.

4. **Логический способ описания функции.**

Пример 1.6. $f(x)=\begin{cases}0, & \text{если } x \notin M \\ x+5, & \text{если } x \in M\end{cases}$

1.5. Отношения на множествах

1.5.1. Понятие отношения

Если в декартовом произведении $A \times A$ некоторого множества A выделено какое-либо подмножество R, то говорят, что на A задано ***отношение*** (или бинарное отношение). Если для некоторых элементов $u, v \in A$ имеет место включение $(u,v) \in R$, то говорят, что u,v находятся в отношении R . В содержательном смысле R рассматривается, как правило, как свойство или как признак, по которому выделяются элементы множества.

Можно ввести обобщенное понятие ***n-местного (n-арного) отношения*** R – это множество упорядоченных наборов:

$$R \subset A_1 \times A_2 \times .... \times A_n = \{(a_1, a_2, ...a_n) \mid a_1 \in A_1, a_2 \in A_2, ..., a_n \in A_n\}.$$

Множества A_i могут совпадать или различаться. Если $(a_1, a_2, ..., a_n) \in R$, то говорят, что эти элементы находятся в отношении R.

Пусть дано отношение R на множестве M. Тогда для любого $M_1 \subset M$ существует отношение $R_1 \in R$, называемое ***сужением*** отношения R на M_1. R_1 получают из R удалением из него всех векторов, содержащих элементы, которые не принадлежат M_1: $R_1 = R \cap M_1^n$.

Частным случаем (при $n=1$) является одноместное отношение; это свойство одиночных элементов. Такие отношения

$R \subseteq M$ называются ***признаками***. Если элемент a обладает этим признаком, то $a \in R$.

Наиболее часто встречаются и хорошо изучены бинарные отношения, т.е. двухместные отношения $(n=2)$. Если a и b находятся в отношении R, это обозначают следующим образом: $(a,b) \in R$ или $aR\,b$. Обычно бинарные отношения на конечных множествах задаются либо списком пар, для которых эти отношения выполняются, либо матрицей бинарных отношений.

Матрица бинарного отношения на множестве $M = \{a_1, a_2, ..., a_n\}$ – это квадратная матрица C, в которой каждый элемент c_{ij} определяется следующим образом:

$$c_{ij} = \begin{cases} 1, \text{ если } a_i R\, a_j; \\ 0, \text{ если } (a_i, a_j) \notin R. \end{cases}$$

Пример 1.7. Пусть задано множество $M = \{1, 2, 3, 4, 5, 6\}$. Отношение R определяется как $a_i \le a_j$. Элемент матрицы $c_{23} = 1$, так как $a_2 \le a_3 (2 \le 3)$:

a_j / a_i	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	0	0	1	1	1	1
4	0	0	0	1	1	1
5	0	0	0	0	1	1
6	0	0	0	0	0	1

Для любого множества M бинарное отношение E, заданное матрицей, в которой по главной диагонали стоят 1, а в остальных местах 0, называется отношением равенства на множестве M.

Обратные отношения. Отношение R^{-1} называется ***обратным*** к отношению R, если $a_j R^{-1} a_i$ имеет место тогда и только тогда, когда $a_i R\, a_j$.

Из определения следует, что если по отношению к обратному отношению вновь взять обратное, то получим исходное отношение.

1.5.2. Типы отношений

1. Отношение R называется ***рефлексивным,*** если для любого элемента $a \in M$ имеет место $aR\,a$. Главная диагональ его матрицы содержит только единицы (например, отношение ≥).

Отношение R называется ***антирефлексивным***, если ни для одного $a \in M$ не выполняется $aR\,a$. Тогда главная диагональ содержит все нули (например, отношение «быть сыном»).

2. Отношение R называется ***симметричным,*** если для любой пары $(a,b) \in M^2$ из $aR\,b$ следует $bR\,a$, то есть для любой пары отношение R выполняется либо в обе стороны, либо не выполняется вообще (например, отношение «быть супругом»). Матрица симметричного отношения симметрична относительно главной диагонали. Отношение называется ***антисимметричным***, если из $a_i R\,a_j$ и $a_j R\,a_i$ следует $a_i = a_j$ (например, отношение «начальник-подчиненный»).

3. Отношение R называется ***транзитивным***, если для любых a, b, c из $aR\,b$ и $bR\,c$ следует $aR\,c$ (например, отношение «жить в одном городе», «начальник-подчиненный», ≥).

Транзитивное замыкание. Для любого отношения R его ***транзитивным замыканием*** называется отношение $\hat{R}$, которое определяется следующим образом: $a\hat{R}\,b$, если в M существует цепочка n элементов $a = a_1, a_2, ..., a_n = b,$ в которой между соседними элементами выполняются отношения $a_1 R\,a_2, a_2 R\,a_3, ..., a_{n-1} R\,a_n$. Если R транзитивно, то $\hat{R} = R$.

Отношение эквивалентности. Одним из важных типов отношений является отношение эквивалентности на множестве. Рассмотрим отношение R на множестве людей, определенное следующим образом: xRy имеет место тогда и только тогда, когда x гражданин той же страны, что и y. Полагаем, что каждый человек является гражданином только одной страны. Отношение R в данном случае удовлетворяет трем свойствам:

а) рефлексивности (x – гражданин той же страны, что и x);

б) симметричности (если x – гражданин той же страны, что и y, то y – гражданин той же страны, что и x);

в) транзитивности (если x – гражданин той же страны, что и y, а y – гражданин той же страны, что и z, то x – гражданин той же страны, что и z).

Отношение R, которое обладает свойствами рефлексивности, симметричности и транзитивности называется ***отношением эквивалентности***.

1.5.3. Разбиение элементов множества на классы

В приведенном выше примере отношение R (отношение эквивалентности) разбивает все множество живущих людей на подмножества людей в соответствии с их гражданством.

Пусть в общем случае на множестве M, содержащем элементы $x \in M$, задано отношение эквивалентности R. Разбиением множества M на классы эквивалентности называют представление этого множества в виде объединения некоторых подмножеств S_i таким образом, что элементы каждого подмножества находятся в отношении эквивалентности, причем $M = \cup S_i$ и $S_i \cap S_j = \varnothing$. Т.о. класс эквивалентности для x – это подмножество элементов множества M, эквивалентных x, т.е. $[x]_R = \{y \mid y \in M \text{ и } xRy\}$.

Разбиение множества на классы эквивалентности производится в следующем порядке:

1) выберем произвольный элемент $a_1 \in M$ и образуем класс C_1, состоящий из a_1 и всех других элементов множества M, эквивалентных a_1;

2) выберем элемент $a_2 \notin C_1$ и образуем класс C_2, содержащий a_2 и другие элементы, эквивалентные a_2.

Продолжая эту процедуру, получим систему классов $C_1, C_2, ..., C_n$ такую, что любой элемент исходного множества входит в один из этих классов. Полученная система называется ***системой классов эквивалентности*** по отношению R. Мощность этой системы называется ***индексом разбиения***.

Система обладает следующими свойствами:

1) любые два элемента из одного класса эквивалентны;
2) любые два элемента из разных классов не эквивалентны;
3) классы попарно не пересекаются;
4) объединение всех классов эквивалентности равно исходному множеству M.

1.5.4. Упорядочивание элементов на множествах

Отношение называется отношением ***строгого порядка,*** если оно антирефлексивно, антисимметрично и транзитивно.

Отношение называется отношением ***нестрогого порядка,*** если оно рефлексивно, антисимметрично и транзитивно. Оба типа отношений называются ***отношениями порядка.***

Пример 1.9. Отношения $>$, $<$ для чисел являются отношениями строгого порядка. Отношения $\geq$, $\leq$ являются отношениями нестрогого порядка.

Элементы a и b ***сравнимы***, если выполняется aRb или bRa.

Множество M, на котором задано отношение порядка, называется ***полностью упорядоченным,*** если любые два элемента сравнимы, и ***частично упорядоченным*** в противном случае.

1.6. Вопросы и задания для самоконтроля

1. Приведите несколько примеров множества, его подмножества и универсального множества.
2. Если заданы три множества $A=\{1,2,3\}$, $C=\{1,a,3\}$ и $B=\{1,2,3,4,5\}$, то какие из них соотносятся как множество и его подмножество?
3. Дать определение основных операций над множествами.
4. Определить значение множества F, заданного в виде:
   а) $F=((B\cup A)\setminus M)\cap C$;
   б) $F=(A\div B)\cup(C\setminus M)$;
   в) $F=(D\cup C)\setminus(M\cup B)$.

   При следующих значениях множеств:
   $A=\{1,2,7\}, B=\{10,7,3\}, D=\{a,b\}, M=\{1,5,a,e\}, C=\{7,10,20\}.$
5. Дать определение понятия вектор.

6. Найти декартово произведение следующих множеств:
   а) *{c,5}* и *{4, 9};*
   б) *{a,b}, {1,2}* и *{f}.*
7. Дать определение понятия соответствие.
8. Дать определение понятия функция.
9. Какие существуют способы задания функций?
10. Какое отношение называется бинарным?
11. Пусть задано множество *T={1,2,3}.* Отношение *R* определяется как $a_i \neq a_j$. Построить матрицу заданного бинарного отношения *R.*

2. ЭЛЕМЕНТЫ ОБЩЕЙ АЛГЕБРЫ

2.1. Понятие алгебры

Функция типа $\varphi: M^n \to M$ называется n- ***арной операцией*** на множестве M. Здесь n называют ***арностью*** операции φ.

Множество M с набором заданных на нем операций $\Omega = \{\varphi_1, \varphi_2, ..., \varphi_m\}$ называется ***алгеброй***. Таким образом, алгебра – это система $A = (M; \varphi_1, \varphi_2, ..., \varphi_m)$.

Множество M называется ***основным*** или ***несущим множеством*** (***носителем*** алгебры A). Совокупность операций Ω называется ***сигнатурой***, а вектор арностей операций называется ***типом*** алгебры.

Свойства бинарных алгебраических операций. **Существуют следующие свойства бинарных алгебраических операций:**

1) операция φ называется ***ассоциативной***, если для любых элементов a, b, c имеет место $(a \varphi b) \varphi c = a \varphi (b \varphi c)$;

2) операция φ называется ***коммутативной***, если для любых a и b выполняется $a \varphi b = b \varphi a$;

3) операция φ называется ***дистрибутивной слева*** относительно операции ψ, если для любых a, b, c $a \varphi (b \psi c) = (a \varphi b) \psi (a \varphi c)$, и называется ***дистрибутивной справа*** относительно ψ, если $(a \psi b) \varphi c = (a \varphi c) \psi (b \varphi c)$.

2.2. Гомоморфизм и изоморфизм

Пусть даны две алгебры одинакового типа $A=(K;\varphi_1,\varphi_2,...,\varphi_m)$ и $B=(M;\psi_1,\psi_2,...,\psi_m)$. Предположим, что между множествами K и M имеет место некоторое соответствие Γ.

Гомоморфизмом алгебры A на алгебру B называется отображение $\Gamma: K \to M$, удовлетворяющие условию:

$$\Gamma\left[\varphi_i\left(k_{j1},k_{j2},...,k_{jl}\right)\right]=\psi_i\left[\Gamma\left(k_{j1}\right),...,\Gamma\left(k_{jl}\right)\right]. \qquad (1)$$

Смысл этого уравнения заключается в том, что независимо от того, выполнена ли сначала операция в алгебре A, а затем выполнено отображение Γ, либо сначала выполнено отображение Γ, а затем выполнена соответствующая операция в алгебре B, результат будет один и тот же.

Алгебры A и B называют ***изоморфными***, если имеет место взаимно-однозначный гомоморфизм A на B и B на A. В этом случае должно существовать обратное отображение $\Gamma^{-1}: M \to K$. Тогда $k_i=\Gamma^{-1}\left(m_j\right)$.

Если подставить последнее выражение в выражение (1) и применить к обеим частям равенства отображение Γ^{-1}, то получим:

$$\varphi_i\left[\Gamma^{-1}\left(m_{j1}\right),\Gamma^{-1}\left(m_{j2}\right)...,\Gamma^{-1}\left(m_{jl}\right)\right]=\Gamma^{-1}\left[\psi_i\left(m_{j1},...,m_{jl}\right)\right].$$

Отображение Γ^{-1} – это гомоморфизм B на A.

Мощности несущих множеств изоморфных алгебр равны.

Если $K=M$, то такой изоморфизм называется ***автоморфизмом*** (изоморфизм на себя). Если $K \subset M$, то такой изоморфизм называется ***изоморфизмом в себя.***

Отношение изоморфизма является отношением эквивалентности на множестве алгебр. Если алгебры A и B изоморфны, то элементы и операции B можно так переименовать, что B совпадет с A. Любое эквивалентное соотношение в алгебре A сохраняется в любой изоморфной ей алгебре B. Это позволяет, получив соотношение в алгебре A, автоматически распространить его на все алгебры, изоморфные

алгебре A. В частности, изоморфизм сохраняет ассоциативность, коммутативность и дистрибутивность алгебраических операций.

2.3. Полугруппы и группы

Алгебра $P=(M;\circ)$ с одной ассоциативной бинарной операцией называется ***полугруппой***. Принятая форма записи $a\circ b$ называется ***мультипликативной***. Если операция коммутативная, то есть $a\circ b=b\circ a$, то полугруппа называется ***коммутативной*** или ***абелевой***. Если полугруппа содержит такой элемент e, что для любого элемента $a\in M$ выполняются равенства $a\circ e=e\circ a=a$, то e называется ***единицей.*** Полугруппа с единицей называется ***моноидом***.

Если любой элемент полугруппы $P=(M;\circ)$ можно представить как произведение некоторого числа элементов множества $M_0\subset M$, то множество M_0 называется ***порождающим***, а его элементы называются ***образующими***. Если полугруппа имеет только одну образующую, то все элементы являются степенями этой образующей. Такая полугруппа называется ***циклической***.

Группой называется полугруппа с единицей, в которой для каждого элемента a существует элемент a^{-1}, называемый ***обратным*** к a и удовлетворяющий условию: $a\circ a^{-1}=a^{-1}\circ a=e$.

Число элементов в конечной группе называется ***порядком группы***. Так же, как и полугруппа, группа может быть абелевой (если соблюдается коммутативность), и если все элементы группы являются некоторыми степенями одного и того же элемента, она называется ***циклической***.

2.4. Алгебраические системы. Решетки

Множества, на которых кроме операций задаются еще и отношения, называются ***алгебраическими системами***. Понятие изоморфизма для алгебраических систем вводится аналогично, только к условию (1) сохранения операций добавляется условие сохранения отношений при изоморфизме.

Рассмотрим пример алгебраической системы, которая находит широкое применение в теоретической алгебре. Такой алгебраической системой является решетка.

Пусть задано некоторое частично упорядоченное множество M, в котором отношение порядка обозначается $\geq$. Верхней гранью для элементов a и b из M назовем любой элемент $c \in M$ такой, что $c \geq a, c \geq b$, а их нижней гранью – любой элемент $d \in M$, такой, что $a \geq d, b \geq d$.

Упорядоченное множество удобно отображать диаграммой, в которой элементам соответствуют точки, а стрелки связывают элементы таким образом, что если $b \geq a$, то стрелка направлена из a в b.

Решеткой называется частично упорядоченное множество, в котором для любых двух элементов a и b существует их пересечение $a \cap b$ – такая наибольшая нижняя грань, что любая другая нижняя грань элементов a и b меньше $a \cap b$, и существует такая верхняя грань $a \cup b$ – такая наименьшая верхняя грань, что любая другая верхняя грань для a и b больше $a \cup b$.

Таким образом, ***решетка*** — это алгебраическая система $\{M; \geq; \cap, \cup\}$

Пример 2.1. Любое полностью упорядоченное множество целых чисел можно представить как решетку, определив для любых $a, b \in N$ операцию $a \cup b = \max(a, b)$ и операцию $a \cap b = \min(a, b)$.

Пример 2.2. Определим на N отношение частичного порядка следующим образом: будем считать что $a \geq b$, если a / b – целое, тогда $a \cup b$ – наименьшее общее кратное чисел a и b, а $a \cap b$ –наибольший общий делитель.

Пример 2.3. Пусть имеется множество B_3 – множество двоичных векторов длиной $n = 3$. Будем считать, что вектор $a \geq b$, если в a единицы стоят на тех же местах, что и в b, но могут также стоять и в других местах (например, $011 \geq 010$).

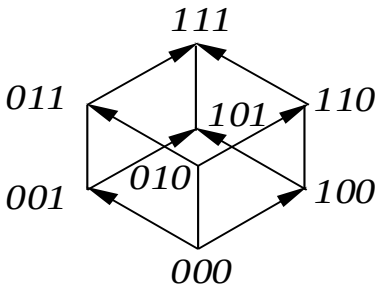


Рис.2.1

Упорядоченное таким образом множество является ***решеткой***. Его диаграмма приведена на рис.2.1. Здесь $a \cup b$ – это вектор, в котором единицы стоят на тех и только тех местах, в которых единица стоит в a или в b, а $a \cap b$ – это вектор, в котором единицы стоят на тех и только тех местах, в которых единицы стоят и в a, и в b.

3. ТЕОРИЯ ГРАФОВ

3.1. Основные понятия теории графов

3.1.1. Понятие графа. Области применения графов

Пусть X - любое множество. Обозначим через $V(X)$ множество всех *неупорядоченных пар* его различных элементов. Например, если $X=\{1,2,3\}$, то $V(X)=\{(1,2), (1,3), (2,3)\}$; если $X=\{1,2\}$, то $V(X)=\{(1,2)\}$. Если $X=\{1\}$, то $V(X)=\varnothing$, так как различных элементов в X нет.

Графом называется пара множеств $G=(X, A)$, где X - любое непустое множество, а $A\subseteq V(X)$. Элементы из X называются ***вершинами*** графа, а элементы из A - его ***ребрами***. Приведем пример графа:

$$X=\{1,2,3,4,5\},\ A=\{(1,2),(2,3),(3,4),(1,5),(1,4),(3,1)\}.$$

Опишем традиционную ***геометрическую интерпретацию*** графа. Пусть $G=(X, A)$ - некоторый граф и $X=\{x_1,x_2,...,x_p\}$, $A=\{a_1,a_2,...,a_q\}$. Фиксируем на плоскости произвольным образом p точек и произвольным образом дадим им имена вершин данного графа; в итоге на плоскости возникнут точки, обозначенные как $x_1,x_2,...,x_p$. Затем, для каждой пары точек x_i,x_j таких, что $(x_i,x_j)\in A$, проведем отрезок прямой, соединяющий точки x_i,x_j. В результате таких действий возникнет некоторый рисунок, который и являются геометрической интерпретацией графа.

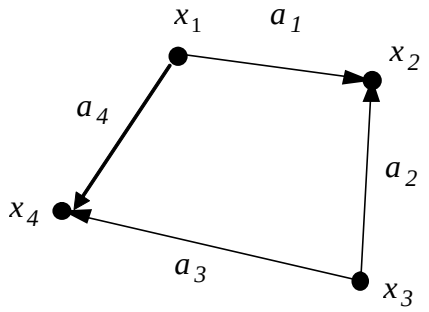


Рис. 3.1

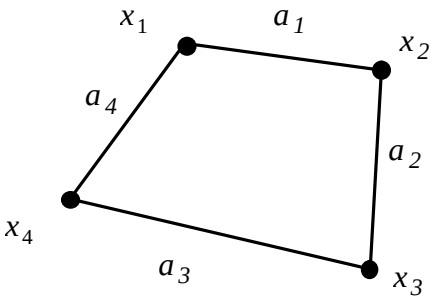


Рис.3.2

Такую форму удобно использовать в качестве наглядной формы представления системы взаимосвязанных объектов. Граф $G=(X, A)$ называют ***ориентированным***, если он состоит из конечного непустого множества вершин X и множества A упорядоченных пар вершин, называемых дугами. Считается, что дуга $a=(x_i, x_j)$ направлена из вершины x в вершину x_j. На изображении ориентированного графа направление дуг показывается стрелками (рис.6.1). Если ребра не имеют ориентации, то граф называется ***неориентированным*** (рис.3.2). Граф, который содержит и дуги, и неориентированные ребра, называется ***смешанным***.

Если в ориентированном графе $G=(X, A)$ можно пренебречь направлением дуг, то соответствующий ему неориентированный граф называется ассоциированным с орграфом G (неориентированным двойником G), обозначается $\overline{G}=(X, \overline{A})$.

В неориентированном графе, который соответствует ориентированному графу, если пренебрегают направлением дуг, то всякая дуга a_i рассматривается как неориентированное ребро и обозначается $\overline{a}_i$.

Число вершин в графе – есть ***мощность*** множества X.

Графы применяются в следующих случаях:

1. Для описания дискретных систем (x_i – возможные состояния системы, a_j – возможные переходы из одного состояния в другое).

2. Для представления многомашинных вычислительных сетей обработки информации и систем связи (x_i – узлы обработки информации, элементы сети, a_j – каналы передачи информации или передаваемая информация).

3. Для представления алгоритма, определяющего последовательность обработки информации (x_i – блоки-операторы алгоритма; a_j – связи по управлению или по информации).

4. Для представления структуры организационной системы, например иерархической структуры предприятия или военной организации (x_i – подразделения, отделы, люди; a_j – отношения

подчиненности, связи; передачи команд и информации).

5. Для представления последовательности работ при проектировании в виде сетевого графика выполнения работ (x_i – работы, этапы проектирования и изготовления; a_j - переходы, определяющие их последовательность).

В виде графов могут быть представлены схемы перемещения на местности, схемы соединения в электрических схемах, система связи и т.д.

Смежные вершины, окрестность**. Две вершины X_a и X_b называются** *смежными***, если они соединены дугой.**

Дуга a, соединенная с вершиной x, называется ***инцидентной*** вершине x, а вершина x – ***коинцидентной*** дуге a.

Множество вершин, смежных с вершиной x_i, называется ее ***окрестностью*** и обозначается $\Gamma(x_i)$.

Если рассматривается ориентированный граф, то окрестностью x_i называется множество вершин, соединенных с x_i дугой, которая начинается в x_i.

Например, задан граф (рис.3.3):

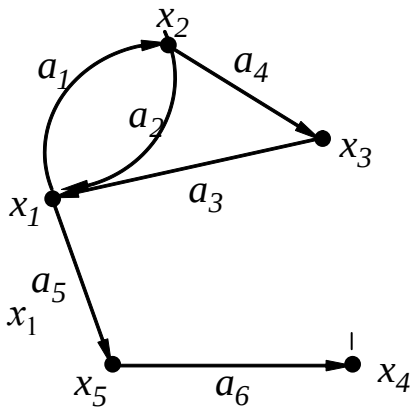


Рис.3.3

Для этого графа

$$\Gamma(x_1) = \{x_2, x_5\};$$
$$\Gamma(x_2) = \{x_1, x_3\};$$
$$\Gamma(x_3) = \{x_1\};$$

$$\Gamma(x_4) = \varnothing;$$
$$\Gamma(x_5) = \{x_4\}.$$

Соответствие Γ является отображением $X \to X$; это закон, по которому каждому $x_i \in X$ ставится в соответствие некоторое подмножество $\Gamma(x_i) \subset X$.

Т.к. множество вершин X и множество окрестностей полностью определяют граф, то граф может задаваться парой $G = (X, \Gamma)$.

Если граф неориентированный (или смешанный), то соответствие Γ задает эквивалентный ориентированный граф, который получается из исходного заменой каждого неориентированного ребра двумя противоположно направленными дугами, соединяющими те же вершины.

Введем также соответствие $\Gamma^{-1}(x_i)$, которое определяет множество вершин, из которых есть дуги в рассматриваемую вершину x_i. Если $\Gamma(x_i)$ – множество таких $x_j \in X$, для которых в G существует дуга (x_i, x_j), то $\Gamma^{-1}(x_i)$ есть множество $x_k \in X$, для которых в G существует дуга (x_k, x_i). Очевидно, что для неориентированного графа $\Gamma^{-1}(x_i) = \Gamma(x_i)$.

Рассмотрим случай, когда отображение Γ вводится не для одной вершины, а для множества вершин.

Пусть $X_q = \{x_1, x_2, ..., x_q\}$. Тогда под $\Gamma(X_q)$ понимают объединение: $\Gamma(X_q) = \Gamma(x_1) \cup \Gamma(x_2) \cup ... \cup \Gamma(x_q)$.

Вводится также ***многократное*** (двойное, тройное) отображение $\Gamma(X_i)$.

$\Gamma(\Gamma(x_i))$ записывается как $\Gamma^2(x_i)$; $\Gamma(\Gamma(\Gamma(x_i)))$ записывается как $\Gamma^3(x_i)$.

Для рассматриваемого графа (рис.6.3):

$\Gamma^2(x_1) = \Gamma(\Gamma(x_1)) = \Gamma(\{x_2, x_5\}) = \{x_1, x_3, x_4\}$;

$\Gamma^3(x_1) = \Gamma(\Gamma^2(x_1)) = \Gamma(\{x_1, x_3, x_4\}) = \{x_1, x_2, x_5\}$.

Аналогично определяется $\Gamma^k(x)$ для любого k. По аналогии вводится $\Gamma^{-2}(x_i), ..., \Gamma^{-k}(x_i)$.

Степени вершин графов. Число ребер, инцидентных данной вершине x_i, называется ***степенью*** этой вершины и обозначается $d(x_i)$ или d_i:

$$d(x_i) = |\Gamma(x_i)|.$$

Вершина, у которой $d_i = 0$, называется ***изолированной***.

Вершина, у которой $d_i = 1$, называется ***висячей***.

Справедливо утверждение: сумма степеней всех вершин графа равна $2m$, где m – число ребер графа, т.е.

$$\sum_{i=1}^{n} d(x_i) = 2m.$$

Для ориентированного графа вместо степени вершин вводят понятие ***полустепеней***. Число дуг, которые имеют вершину x_i своей начальной вершиной, называется ***полустепенью исхода*** вершины x_i и обозначается $d^-(x_i)$. Число дуг, которые имеют x_i своей конечной вершиной, называется ***полустепенью захода*** вершины x_i и обозначается $d^+(x_i)$. Для полустепеней справедливы выражения:

$$d^-(x_i) = |\Gamma(x_i)|, \qquad d^+(x_i) = |\Gamma^{-1}(x_i)|,$$

$$\sum_{i=1}^{n} d^-(x_i) = \sum_{i=1}^{n} d^+(x_i) = m.$$

Пути и маршруты. Пусть $G = (X, A)$ – неориентированный граф, в котором $x_1, x_2, x_3, ..., x_{k-1}, x_k$ – последовательность смежных вершин. Тогда последовательность ребер $a_1, a_2, ..., a_{k-1}$, где a_j соединяет вершины x_j с x_{j+1} и называется ***маршрутом***, соединяющим x_1 с x_k.

Если первая и последняя вершины маршрута совпадают, такой маршрут называется ***замкнутым***.

Маршрут, в котором ребра не повторяются, называется ***цепью***.

Цепь называется ***простой***, если все ее вершины различны.

Замкнутая цепь, в которой первая и последняя вершины совпадают, называется ***циклом***.

Простой цикл – это цикл, в котором все вершины различны.

Для ориентированного графа вводятся аналогичные понятия.

Путём (или ***ориентированным маршрутом***) называется последовательность дуг, в которой конечная вершина всякой дуги (кроме последней) является начальной вершиной следующей.

На рис.3.3: $<a_1, a_4, a_3, a_5, a_6>$ – это путь.

Ориентированной цепью (или *орцепью*) называется такой путь, в котором каждая дуга используется не более одного раза.

Простой орцепью (элементарный путь) называется такой путь, в котором каждая вершина используется не более одного раза.

Путь $a_1, a_2, ..., a_q$ называется ***замкнутым,*** если в нем начальная вершина дуги a_1 совпадает с конечной вершиной дуги a_q.

Простой орцикл называется ***контуром***. Контур, проходящий через *все* вершины графа, называется ***гамильтоновым*** контуром.

Дуга, начальная и конечная вершины которой совпадают, называется ***петлей.***

Количество ребер, входящих в маршрут, называется ***длиной*** маршрута.

Длиной пути называется число дуг, входящих в него, причем каждая дуга считается столько раз, сколько она входит в этот путь.

Взвешенный граф. Иногда дугам графа G ставятся в соответствие некоторые числа, называемые ***весом*** дуги. Такой граф G называется ***графом с взвешенными дугами***. Физический смысл веса – длина, стоимость, время
и т. д.

Если веса приписываются вершинам x_i графа, то такой граф называется ***графом с взвешенными вершинами.***

Если в графе веса приписаны и дугам, и вершинам, то он называется просто ***взвешенным.***

3.1.2. Подграфы

Пусть дан граф $G = (X, A)$. Удаляя из исходного графа некоторые ребра и вершины, можно получить подграф, то есть часть графа. ***Подграф*** – это граф, содержащий часть вершин исходного графа $x_i \in X_s$, $X_s \subseteq X$ и часть ребер $a_j \in A_s$, $A_s \subset A$ причем начальные и конечные вершины ребер подграфа принадлежат множеству X_s.

Выделяют два характерных вида подграфов: остовный и порожденный.

Остовным подграфом G_p графа G называется граф, который имеет все те же вершины, что и исходный граф, а его дуги – часть дуг исходного графа. Таким образом, $G_p = (X_1, A_1)$ – есть остовный подграф графа $G = (X, A)$, если $X_1 = X$ и $A_1 \subseteq A$.

Порождённым подграфом G_S называется граф (X_S, Γ_S), в котором $X_S \subset X$ и для каждой вершины $x_i \in X$ выполняется

$\Gamma_S(x_i) = \Gamma(x_i) \bigcap X_S$,

т.е. порожденный подграф содержит часть вершин X_S и все те дуги исходного графа, у которых начальные и конечные вершины принадлежат X_S. Для графа (рис.3.4) приведены примеры остовного (рис.3.5) и порожденного (рис.3.6) подграфов.

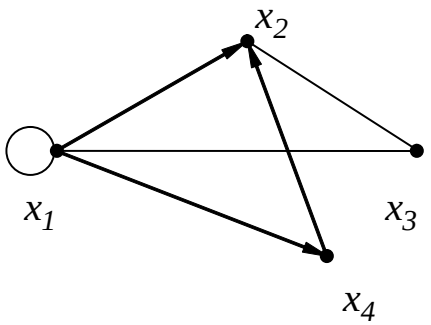


Рис.3.4

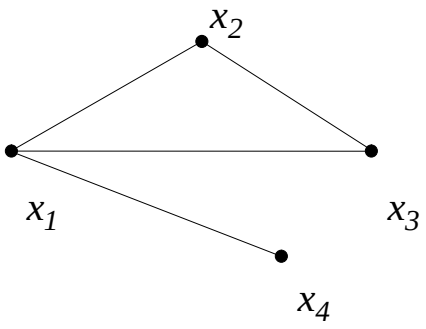


Рис.3.5

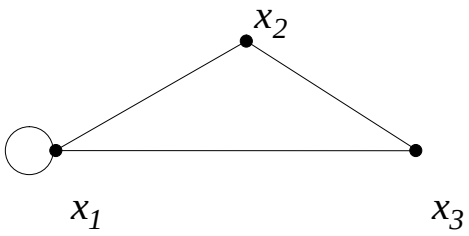


Рис.3.6

3.1.3. Типы графов

Рассмотрим следующие типы графов:

1. Граф, не имеющий ребер, называется пустым.
2. Граф $G = (X, A)$ называется полным (рис.3.7), если для любой пары вершин x_i и x_j из X существует ребро (x_i, x_j). Полный граф, содержащий n вершин, обозначается K_n.

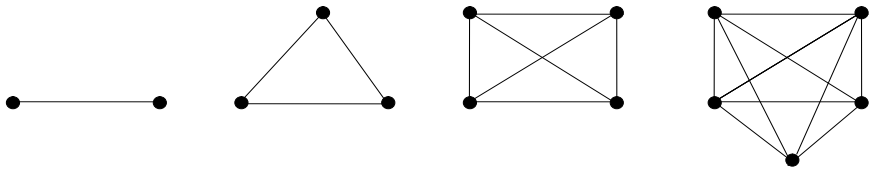


Рис.3.7

3. Граф $G = (X, A)$ называется ***симметрическим***, если во множестве A для любой дуги (x_i, x_j) существует также противоположно ориентированная дуга (x_j, x_i), рис. 3.8.

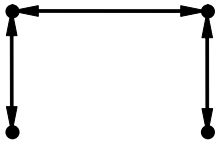


Рис.3.8

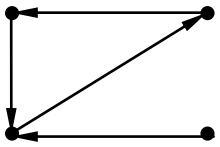


Рис.3.9

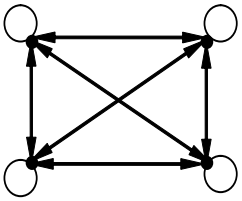


Рис.3.10

4. ***Антисимметрическим*** графом называется такой граф, для которого справедливо условие: если $(x_i, x_j) \in A$, то во

множестве A нет противоположно ориентированной дуги, т.е. $(x_j, x_i) \notin A$ (рис.3.9). Очевидно, в таком графе нет петель.

5. ***Полный симметрический*** граф обладает одновременно свойствами полного и симметрического графа (рис.3.10).

6. Полный антисимметрический граф называется ***турниром***.

Примером симметрического графа может быть графическое представление отношений родственников, друзей. Отношения подчинения образуют антисимметрический граф.

7. Неориентированный граф $G = (X, A)$ называется ***двудольным***, если множество вершин X можно разбить на два таких подмножества X^a и X^b, что каждое ребро имеет один конец в X^a, а другой - в X^b (рис.3.11).

Ориентированный граф G называется двудольным, если его неориентированный двойник $\overline{G}$ – двудольный граф.

Поскольку X разбивается на две части X^a и X^b, то $X^a \bigcap X^b = \varnothing$ и $X^a \bigcup X^b = X$.

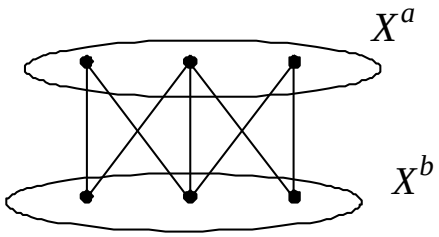


Рис.3.11

Можно показать, что если граф не содержит циклов нечетной длины, то такой граф является двудольным. Для подтверждения данного положения рассмотрим пример графа, приведенный на рис. 3.12. Выполним последовательно следующие операции:

1) выберем одну из вершин x_i и пометим ее индексом “+”;

2) все вершины из $\Gamma(x_i)$ помечаем противоположным знаком;

3) операцию пометки вершин продолжаем до тех пор, пока не пометим все вершины.

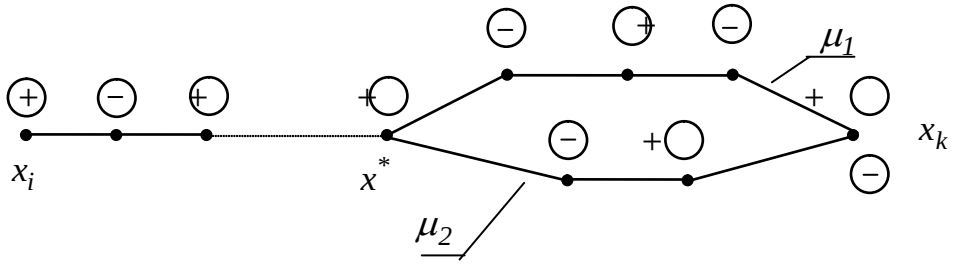


Рис.3.12

Возможны следующие варианты исхода:

А. Какие-то вершины оказались непомеченными. Если между помеченными и непомеченными вершинами не существует ребра, тогда граф распадается на несколько частей, и каждый из них рассматривается отдельно.

Б. Знаки при обходе по различным маршрутам согласованы. Тогда получили двудольный граф. Все вершины, помеченные символом "+", отнесем к множеству X^a; все вершины, помеченные символом "–", отнесем к множеству X^b.

В. Знаки на некоторых вершинах не согласовываются (рис.3.12). Пусть x^* – предпоследняя общая вершина маршрутов μ_1 и μ_2 имеет пометку "+". Вершина x_k, уже помеченная на маршруте μ_1 знаком "+", со стороны другой вершины (на маршруте μ_2) должна быть помечена противоположным знаком "–".

Разные знаки x_k на маршрутах μ_1 и μ_2 могут быть только в том случае, если на одном из них (μ_1) – четное число ребер, а на другом (μ_2) – нечетное. В этом случае цикл $(x^* \to x_k \to x^*)$ имеет нечетную длину, что противоречит предположению. Значит, случай (В) невозможен.

Если нужно подчеркнуть, что граф является двудольным, то применяют обозначение $\left(X^a \cup X^b, A\right)$.

8. Двудольный граф называется ***полным***, если для любых двух вершин $x_i \in X^a$, $x_j \in X^b$ существует ребро $\left(x_i, x_j\right)$.

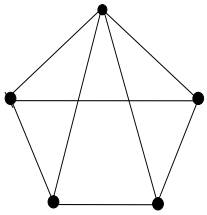


Рис.3.13

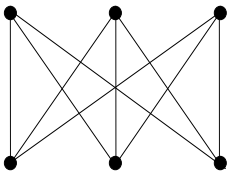


Рис.3.14

Полный неориентированный двудольный граф $G=\left(X^a \cup X^b, A\right)$ обозначается $K_{r,s}$, если $\left|X^a\right|=r;\ \left|X^b\right|=s$.

9. Граф $G=(X,A)$ называется ***планарным***, если он может быть нарисован на плоскости таким образом, что произвольные две дуги графа ***не пересекаются*** друг с другом.

10. В теории графов играют важную роль ***графы Куратовского - непланарные*** гра фы (рис.3.13 и 3.14).

Изоморфизм графов. Графы, которые в определенных условиях неразличимы, могут рассматриваться как один и тот же объект. Например,

если переобозначить вершины (рис.3.15), то эти графы одинаковы, и их считают изоморфными.

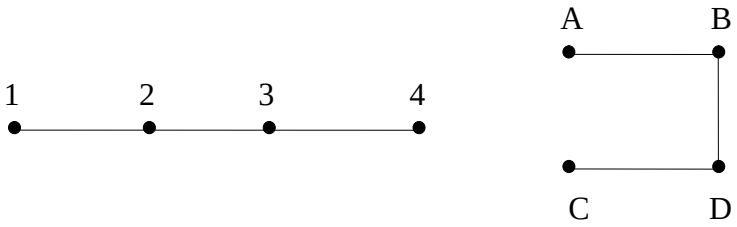


Рис.3.15

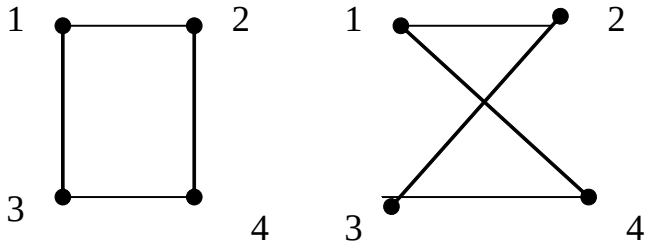


Рис.3.16

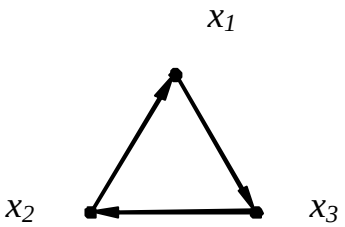


Рис.3.17

Графы $G = (X, A)$ и $G_1 = (X_1, A_1)$ называются ***изоморфными***, если между множествами их вершин можно установить взаимно однозначное соответствие, такое, что вершины смежны в одном из графов в том и только в том случае, когда соответствующие им вершины смежны в другом графе (рис.3.16).

Связность в графе. Граф называется ***связным***, если две любые его вершины можно соединить цепью.

Рассмотрим произвольный неориентированный граф $G = (X, A)$. Возьмем вершину $x_1 \in X$. Пусть X_1 – множество всех вершин, которые
можно соединить цепью с x_1. Тогда $G_1(X_1, A_1)$ – ***порожденный подграф***. Это связный граф, и в G нет ребер, соединяющих вершины $x_i \in X_1$ с вершинами из $X \setminus X_1$.

Возьмем далее $x_2 \in X \setminus X_1$. Аналогично получаем $G_2 = (X_2, A_2)$. Если $X_1 \cup X_2 \neq X$, то продолжаем эту операцию до тех пор, пока не останется свободных вершин.

В результате будут построены связные графы $G_1, G_2, ..., G_p$, причем $X = X_1 \cup X_2 \cup ... \cup X_p$. Эти графы не имеют общих вершин и ребер. Графы $G_1, G_2, ..., G_p$ называют ***компонентами связности*** графа G.

Величина p (число компонент связности графа) называется ***числовой характеристикой*** графа. Для связного графа $p = 1$.

Если граф не является связным и распадается на несколько компонент, то последующий анализ сводится к самостоятельному рассмотрению этих компонент.

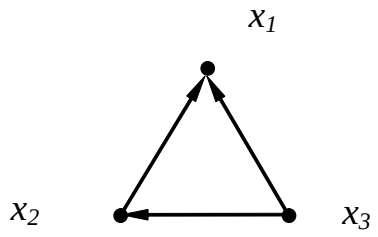


Рис.3.18

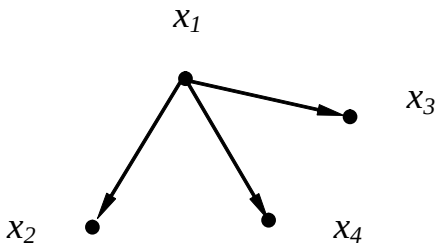


Рис.3.19

Ориентированный граф называется ***сильно связным*** или ***сильным***, если для любых двух различных вершин x_i и x_j существует по крайней мере один путь, соединяющий x_i с x_j. Из этого следует, что любые две вершины такого графа ***взаимно достижимы*** (рис.3.17).

Орграф называется ***односторонне связным*** или ***односторонним***, если для любых двух различных вершин x_i и x_j существует по крайней мере один путь из x_i в x_j или из x_j в x_i (рис.3.18).

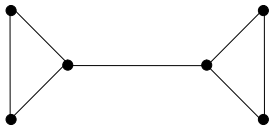


Рис.3.20

Ориентированный граф называется ***слабо связным*** или ***слабым***, если для любых двух различных вершин x_i и x_j существует по крайней мере один маршрут в соответствующем неориентированном графе, соединяющий их (рис.3.19).

Пусть задан граф $G=(X, A)$.

Ребро графа, через которое проходит хотя бы один цикл, называется ***цикловым ребром***.

Ребро, которое не входит ни в один цикл, называется ***перешейком*** (рис.3.20)

Лемма 3.1: При удалении из связного графа циклового ребра он остается связным. При удалении из связного графа перешейка граф распадается на две компоненты.

Если $|X|=n,\ |A|=m$, то величина $\lambda = m-n+p$ называется ***цикломатическим числом графа***.

3.2. Матричное представление графов

Для описания графов удобно использовать матрицы различного типа.

1. ***Матрица смежности.*** Пусть дан граф G, его матрица смежности $A=\left[a_{ij}\right]$ – это матрица, элементы которой определяются следующим образом:

$a_{ij}=1$, если в G существует дуга $\left(x_i, x_j\right)$;

$a_{ij}=0$, если в G нет дуги $\left(x_i, x_j\right)$.

Построим матрицу смежности в форме таблицы для следующего графа (рис.3.21):

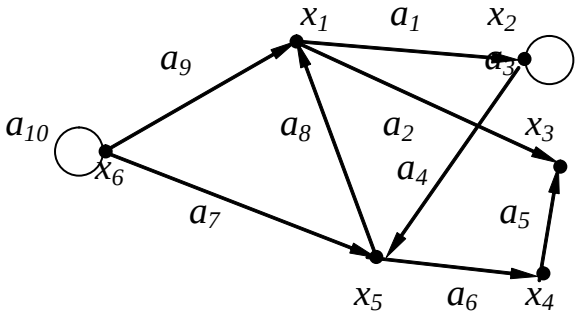


Рис.3.21

$A=$

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	0	0	0
x_2	0	1	0	0	1	0
x_3	0	0	0	0	0	0
x_4	0	0	1	0	0	0
x_5	1	0	0	1	0	0
x_6	1	0	0	0	1	1

Матрица смежности полностью определяет структуру графа. Сумма всех элементов строки x_i – это ***полустепень исхода*** вершины x_i. Сумма единиц в столбце x_j – это ***полустепень захода*** этой вершины. Множество столбцов, имеющих единицу в строке x_i, – это окрестность $\Gamma\left(x_i\right)$. Множество строк, имеющих единицу в столбце x_j, – это $\Gamma^{-1}\left(x_j\right)$.

Рассмотрим, какую информацию несет возведение матрицы A в степень. Элемент $a_{ik}^{(2)}$ матрицы $A^2 = A \cdot A$ определяется выражением $a_{ik}^{(2)} = \sum_{j=1}^{n} a_{ij} \cdot a_{jk}$.

Каждое слагаемое равно единице только тогда, когда $a_{ij} = 1$ и $a_{jk} = 1$, а это означает, что существует путь из x_i в x_k вершину длиной *2*. Таким образом, $a_{ik}^{(2)}$ – это число путей длины *2*, ведущих из x_i в x_k.

По аналогии $a_{ik}^{(p)}$ – элемент матрицы A^p, он определяет число путей длины p, ведущих из x_i в x_k.

2. ***Матрица инцинденций***. Это матрица $B=[b_{ij}]$ элементы которой определяются следующим образом:

$b_{ij} = 1$, если x_i является начальной вершиной дуги a_j;

$b_{ij} = -1$, если x_i является конечной вершиной дуги a_j;

$b_{ij} = 0$, если x_i не является концевой вершиной для a_j или если a_j является петлей.

Матрице инцинденций соответствует таблица, строками которой являются вершины графа G, а столбцами – его дуги. Для графа, приведенного на рис.3.21, матрица инцинденций имеет вид

$B=$

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
x_1	1	1	0	0	0	0	0	-1	-1	0
x_2	-1	0	0	1	0	0	0	0	0	0
x_3	0	-1	0	0	-1	0	0	0	0	0
x_4	0	0	0	0	1	-1	0	0	0	0
x_5	0	0	0	-1	0	1	-1	1	0	0
x_6	0	0	0	0	0	0	1	0	1	0

Так как каждая дуга (за исключением петли) инцидентна только двум различным вершинам, то в каждом столбце либо все элементы равны нулю (это соответствует петле), либо столбец содержит один элемент “+1” и один элемент “-1”.

Если граф неориентированный, то в его матрице инцинденций все элементы “-1” заменяются на “+1”.

3. ***Матрица достижимости***. Вершина x_j считается достижимой из x_i, если в графе существует путь из x_i в x_j. Матрица достижимости R – это матрица $R=[r_{ij}]$, элементы которой определяются следующим образом:

$$r_{ij}=\begin{cases}1, \text{ если } x_j \text{ достижима из } x_i;\\ 0, \text{ если } x_j \text{ не достижима из } x_i.\end{cases}$$

Для каждой строки x_i совокупность вершин x_j, соответствующих столбцам, в которых $r_{ij}=1$, образует множество $R(x_i)$ – множество элементов, которые достижимы из x_i.

Для определения $R(x_i)$ сначала формируем $\Gamma(x_i)$ – множество вершин, для которых в графе существуют дуги (x_i, x_j); это - множество вершин, достижимых из x_i с помощью пути, длина которого равна единице.

Далее определяется $\Gamma(\Gamma(x_i))=\Gamma^2(x_i)$ – множество вершин, достижимых из x_i с помощью путей длиной 2. Продолжая эту операцию, находим $\Gamma^p(x_i)$ – множество вершин, достижимых из x_i с помощью путей длиной p.

Так как любая вершина графа G, которая достижима из x_i, может быть достигнута с использованием путей различной длины, то множество всех достижимых вершин $R(x_i)$ может быть определено как объединение всех вариантов:

$$R(x_i)=\{x_i\}\cup\Gamma(x_i)\cup\Gamma^2(x_i)\cup\ldots\cup\Gamma^p(x_i).$$

Операция объединения выполняется последовательно (слева направо) до тех пор, пока текущее множество не перестанет увеличиваться.

Общее правило построения матрицы достижимости R:

1. Для каждой из вершин графа $x_i \in X$ находим рассмотренным способом $R(x_i)$.

2. Определяем элементы матрицы r_{ij}; получаем:

$r_{ij}=1$, если $x_j \in R(x_i)$, и $r_{ij}=0$ в противном случае.

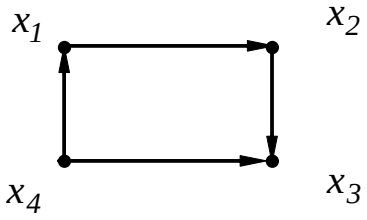


Рис.3.22

Например, пусть задан граф (рис.3.22), для этого графа

$R(x_1)=\{x_1\}\cup\{x_2\}\cup\{x_3\}=\{x_1,x_2,x_3\};$

$R(x_2)=\{x_2\}\cup\{x_3\}=\{x_2,x_3\};$

$R(x_3)=\{x_3\};$

$R(x_4)=\{x_4\}\cup\{x_1,x_3\}\cup\{x_2\}\cup\{x_3\}=\{x_1,x_2,x_3,x_4\}.$

$$R=\begin{array}{|cccc|}\hline 1 & 1 & 1 & 0\\ 0 & 1 & 1 & 0\\ 0 & 0 & 1 & 0\\ 1 & 1 & 1 & 1\\ \hline\end{array}$$

3.3. Эйлеровы графы

Выдающийся математик Эйлер, решая задачу, которая известна как задача о Кенигсбергских мостах (рис.3.23), пытался ответить на вопрос, можно ли обойти все участки $(1,2,3,4)$ расположенного на реке города, начав с некоторой точки, и и вернуться в исходную точку, пройдя по всем мостам (a,b,c,d,e,f,g) по одному разу. Схеме (рис.3.23) можно поставить в соответствие граф, показанный на рис.3.24 .

В общем случае ***эйлеровым графом*** называется граф, который содержит эйлеровый цикл, то есть замкнутый маршрут, включающий в себя все ребра, каждое из которых проходится один раз. Такой граф можно нарисовать, не отрывая перо от бумаги, причем процесс рисования начинается и заканчивается в одной точке. Следует отметить, что граф, изображенный на рис.3.24, не является эйлеровым.

Конечный неориентированный граф G является эйлеровым тогда и только тогда, когда он связен и степени всех его вершин четные (рис.3.25).

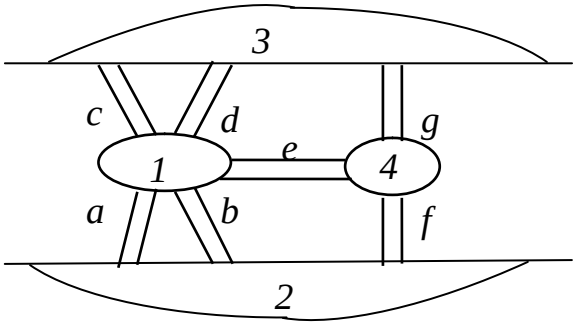


Рис.3.23

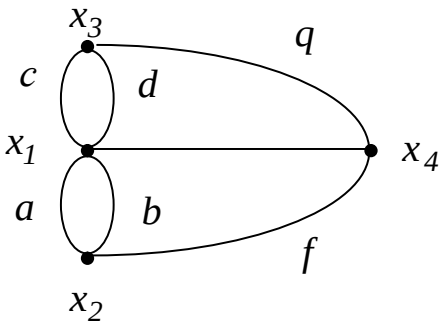


Рис.3.24

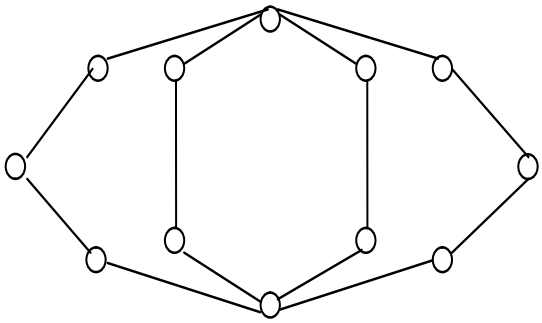


Рис. 3.25

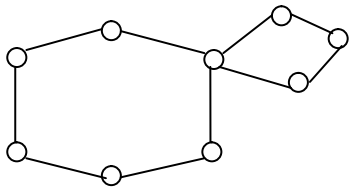


Рис.3.26

Действительно, совершая цикл мы по одному ребру входим, а по другому выходим. Таким образом, все прилегающие к вершине ребра можно разбить на пары; следовательно степени вершин в построенном цикле четные. Если построенный цикл не охватывает все ребра (рис. 3.26), то можно построить новый цикл, в котором (с учетом тех же соображений) степени вершин

тоже будут четными. Процедуру следует повторять, пока не будут охвачены все ребра.

Эйлерова цепь. Если цепь включает все ребра заданного неориентированного графа G, но имеет разные начало V' и конец V'', то она называется ***эйлеровой цепью***. Чтобы в графе существовала эйлерова цепь, необходимы его связность и четность степеней всех вершин, кроме двух V' и V'' (доказывается аналогично).

Эйлеровы ориентированные графы. Чтобы в конечном орграфе существовал эйлеровый цикл, необходимо и достаточно равенство полустепеней исхода и захода всех вершин этого графа, то есть для всех вершин $V_i \in G$ должно выполняться следующее равенство:

$$d^+(V_i) = d^-(V_i).$$

Любому неориентированному графу соответствует ориентированный граф, в котором каждое ребро заменяется двумя дугами во встречных направлениях. При этом обеспечивается выполнение указанного условия. Поэтому в конечном связном графе всегда можно построить цикл, проходящий через каждое ребро по одному разу в каждом из двух направлений. Такой цикл является одним из способов обхода всех ребер графа.

3.4. Деревья

Связный граф без циклов, петель и кратных ребер называется ***деревом.***

Несвязный граф без циклов, без петель и кратных ребер называется ***лесом.***

Теорема 3.3. Любые две вершины V' и V'' дерева связаны одной и только одной цепью.

Если предположить, что две вершины графа V' и V'' связаны двумя цепями, то в графе образуется цикл, что противоречит определению дерева. Верно и обратное: если две любые вершины связаны единственной цепью, то этот граф является деревом (рис.3.27).

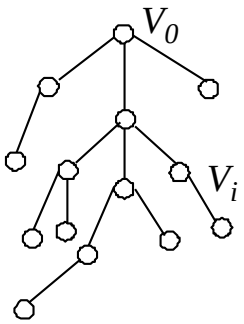


Рис.3.27

Вершины графа называются ***узлами***, а основу V_0 (рис.3.27) дерева называют ***корнем***. Если корень удалить, то дерево превращается в лес, а если к лесу добавить корень, то лес превращается в дерево.

Дерево обычно просматривается от корня к вершинам; при этом между узлами имеет место отношение «исходный–порожденный». Все узлы, кроме корня, имеют только один исходный узел и могут иметь один или несколько (или не иметь совсем) порожденных узлов.

Ветвь дерева – это порожденный подграф дерева G, множество вершин которого связано с корнем цепями, проходящими через одну вершину V_i.

Концевые вершины, имеющие степень *1*, называются ***листьями***.

В ориентированных деревьях ребра обычно ориентированны в одном направлении (либо от корня, либо к корню).

Если у нескольких порожденных узлов, имеющих один исходный, вводится порядок, то такое дерево называется ***упорядоченным***.

Одним из распространенных классов деревьев являются ***бинарные*** деревья, то есть такие деревья, в которых каждый узел имеет не более двух порожденных узлов.

Для любого дерева справедливо свойство взаимно однозначного соответствия между числом вершин и ребер:

$m = n - 1$,

где m – число ребер, n – число вершин.

Произвольный связанный неориентированный граф может быть преобразован в дерево удалением части его ребер.

Деревья достаточно широко используются для решения различных задач. Они удобны для представления структуры систем, классификации, оптимального выбора вариантов построения технических и программных комплексов. Как правило, алгоритмы решения инженерных задач строятся на основе выбора определенного порядка обхода деревьев.

В качестве примера рассмотрим две задачи, иллюстрирующие использование бинарных деревьев в системах обработки данных.

Пример 1. ***Задача поиска данных в массиве.*** Пусть в некотором массиве хранится множество данных (или блоков данных) $M = \{X_1, X_2, \ldots X_k\}$. Данные в массиве нумеруются таким образом, чтобы можно было извлечь любую запись X_i по ее номеру (адресу).

Наиболее простой способ поиска данных по заданному номеру (ключу) состоит в последовательном просмотре списка данных и сравнении текущего номера записи с заданным номером записи, поиск которой производится. В тот момент, когда фиксируется совпадение указанных номеров, операция поиска считается законченной, и найденная запись извлекается для обработки или корректируется.

Для повышения эффективности поиска используется ряд методов, которые позволяют уменьшить время нахождения нужной записи. Одним из них является метод деления пополам. Его сущность заключается в том, что в исходном списке определяется средний номер (узловая точка A_0), который делит исходное множество данных на два подмножества $M_1=\{X_1, \ldots X_{A0-1}\}$ и $M_2=\{X_{A0}, \ldots X_k\}$. Далее каждый из новых списков вновь делится пополам в узловых точках B_1 и B_2, затем в точках C_1, C_2, C_3, C_4 и так далее. Деление продолжается до тех пор, пока в полученных новых списках остается не более одной точки.

Процедура поиска состоит в последовательном просмотре узловых точек, начиная с A_0. Если номер узловой точки меньше, чем номер искомой записи, то последующий поиск осуществляется в первом списке (далее проверяется точка B_1); если больше – во втором списке (проверяется точка B_2).

Для формирования алгоритма поиска, реализующего данный метод, построим упорядоченное двоичное дерево,

вершинами которого являются узловые точки. Корень дерева – точка А0. Порожденные узлы формируются по правилу: узловая точка, соответствующая меньшему номеру, рисуется слева, а точка, соответствующая большему номеру – справа.

Список данных

x_1	
x_2	---D_1
x_3	-------C_1
x_4	---D_2
x_5	----------B_1
x_6	---D_3
x_7	-------C_2
x_8	---D_4
x_9	
x_{10}	----------A_0
x_{11}	---D_5
x_{12}	
x_{13}	-------C_3
x_{14}	---D_6
x_{15}	---------B_2
x_{16}	
x_{17}	---D_7
x_{18}	------C_4
x_{19}	---D_8
x_{20}	

Рис.3.28

На рис.3.29 приведен пример построения такого дерева для списка данных, приведенного на рис.3.28. Процедура поиска заключается в последовательном обходе узлов дерева, начиная с корня, в соответствии с приведенным выше правилом просмотра узловых точек. Например, для поиска записи X_{12} последовательность выбора узлов следующая: X_{10}, X_{15}, X_{13}, X_{11}, X_{12}.

Из графа на рис.3.29 следует, что количество проверок не превышает пяти для любой искомой записи. Таким образом,

использование данного метода позволяет существенно ускорять процедуру поиска данных.

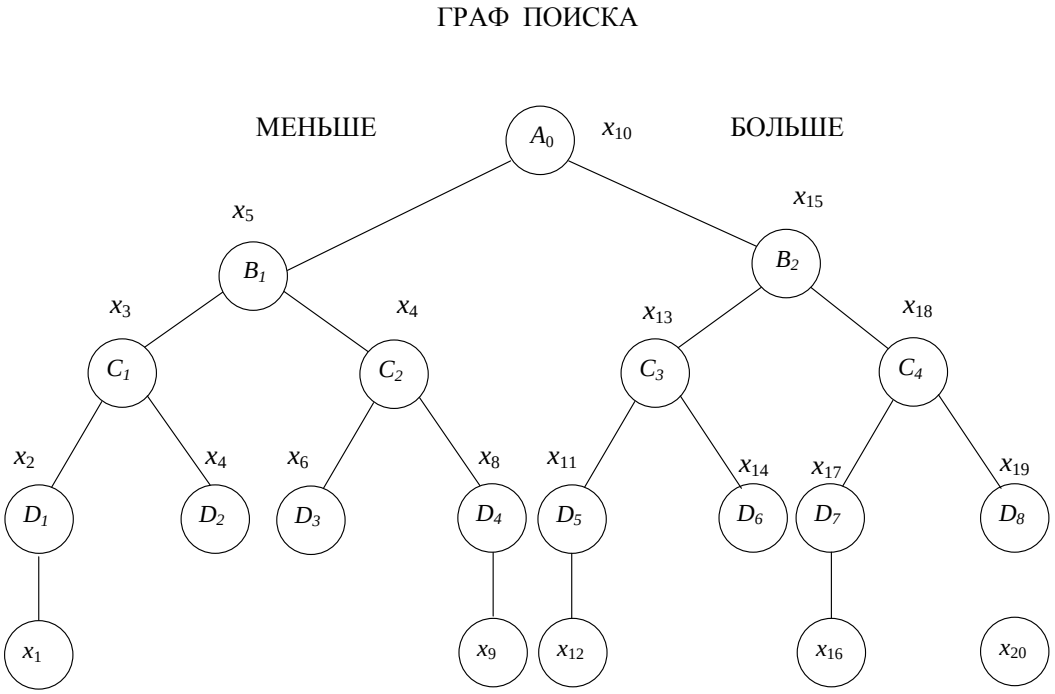


Рис.3.29

Пример 2. ***Алгоритм определения последовательности закраски изображения.*** Одной из наиболее сложных задач при построении трехмерной картины реального мира средствами машинной графики является задача удаления невидимых элементов изображения, в частности тех, которые перекрываются другими телами, расположенными ближе к наблюдателю. Такие более отдаленные объекты не должны быть видны в формируемом изображении. Одним из эффективных способов решения данной задачи является применение “метода художника”. Рисуя картину на полотне, художник вначале рисует фон, затем последовательно наносит более близкие объекты, покрывая красками ранее нарисованные объекты дальнего плана.

Аналогично можно формировать изображение и в памяти компьютера. При формировании отображаемого кадра в начале в память следует занести самый удаленный объект, затем последовательно вносить объекты, которые ближе к

наблюдателю. При этом в той области дискретной матрицы точек экрана, которая соответствует изображению вводимого более близкого объекта, записываются коды цветов точек поверхности этого объекта.

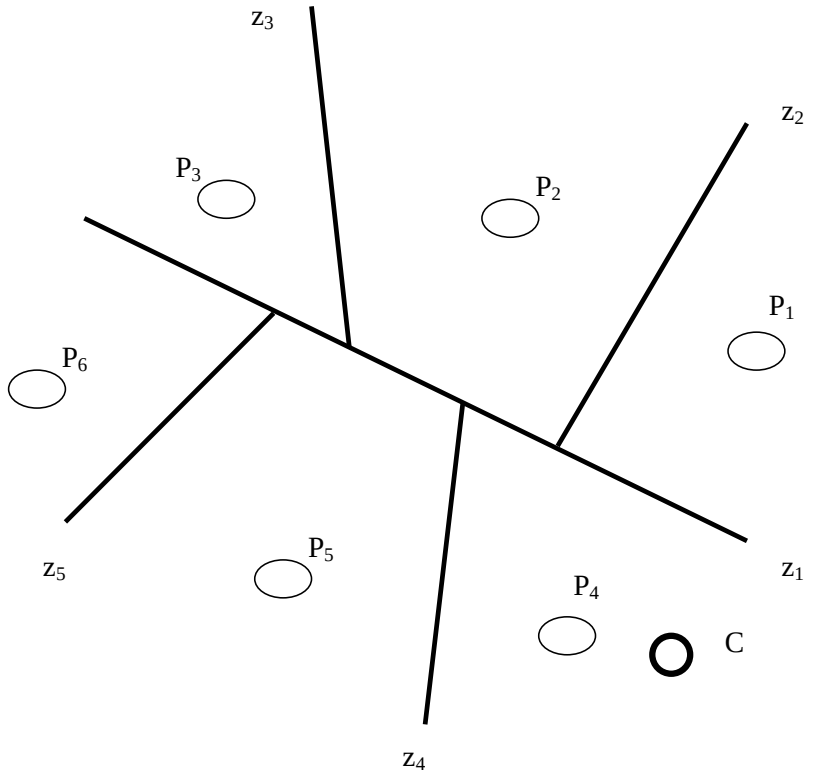


Рис.3.30

Очевидно, что при использовании данного способа все объекты сцены предварительно должны быть упорядочены по степени удаленности от наблюдателя. Эта задача решается на основе построения бинарного дерева после разбиения всего пространства сцены на отдельные зоны – кластеры. Разбиение пространства производится в следующем порядке (рис.3.30).

В объектном пространстве проводим плоскость z_1 таким образом, чтобы она разбивала множество всех объектов (например, $P_1 - P_6$) на два непересекающихся подмножества (кластера). В одном из этих кластеров находится наблюдатель *C*. Принципиально важно то, что ни один из объектов подпространства, не содержащего наблюдателя, ($P_1 - P_3$) не

перекрывает объектов ближнего подпространства, в котором находится наблюдатель.

Далее каждый кластер вновь делим на два подмножества очередной плоскостью (z_2, затем z_3 и так далее). Эта операция повторяется до тех пор, пока в каждом кластере не останется только один объект сцены.

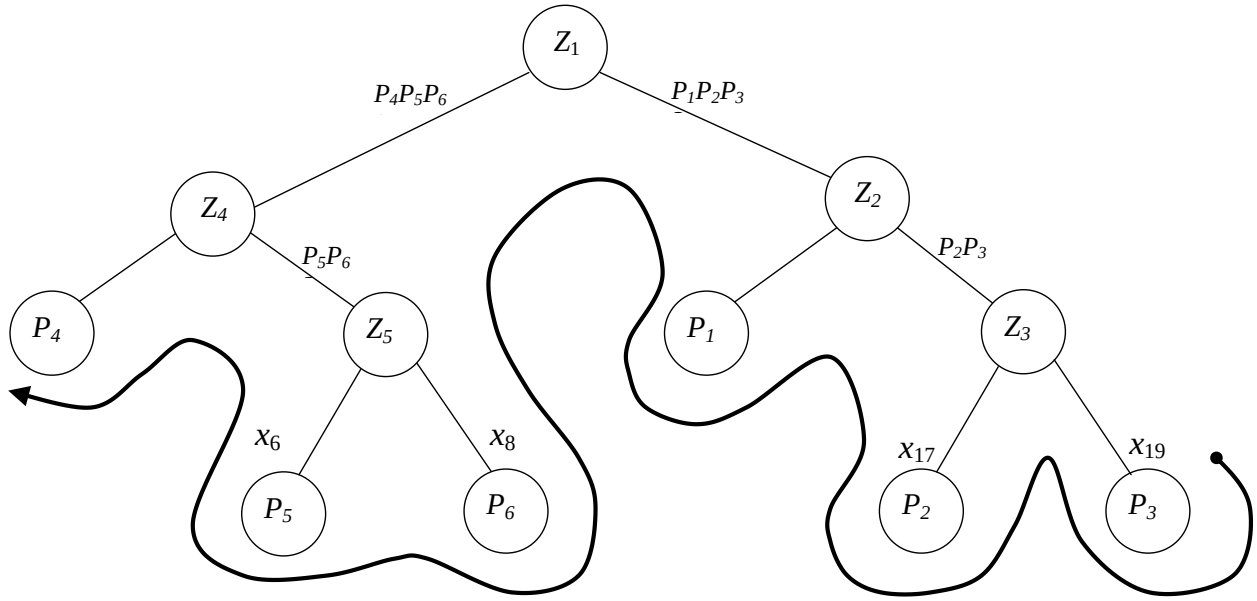


Рис.3.31

После этого строится дерево. Узлами дерева являются секущие плоскости, а листьями – объекты сцены. Начинается построение дерева (рис.3.31) с корня z_1. Два порожденных узла соответствуют следующим секущим плоскостям (z_2 и z_4). Ребра, соединяющие исходный и порожденные узлы, соответствуют подмножествам объектов в кластерах, на которые разбивает пространство плоскость исходного узла.

При построении графа обязательно выполнение правила: влево рисуется ребро, соответствующее объектам того кластера, в котором находится наблюдатель (то есть расположенные ближе к наблюдателю). Например, в рассматриваемом случае ребру (z_1, z_2) соответствует подмножество объектов $M_1=\{P_1, P_2, P_3\}$, а ребру (z_1, z_4) соответствует подмножество объектов $M_2=\{P_4, P_5, P_6\}$.

Построенное дерево является основой для определения порядка вывода объектов (от самого дальнего к самому

близкому). На рис. 6.31 жирной линией показан порядок обхода дерева для рассматриваемого примера. Таким образом, для того чтобы ни один из выводимых объектов не перекрывал другой объект, расположенный ближе к наблюдателю, объекты должны записываться в файл отображаемого кадра в следующем порядке:

$P_3 \rightarrow P_2 \rightarrow P_1 \rightarrow P_6 \rightarrow P_5 \rightarrow P_4$.

При перемещении наблюдателя относительно объектов требуется формировать новый граф. Однако разбиение пространства на кластеры для одной и той же сцены производится один раз.

Остовные деревья. Если $G=(X, A)$ – неориентированный связанный граф с n вершинами (рис.3.32), то ***остовным деревом*** (или остовом) графа G называется любой остовный подграф G, являющийся деревом (рис.3.33).

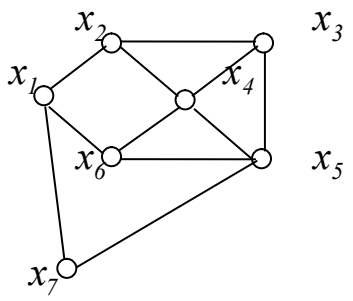


Рис.3.32

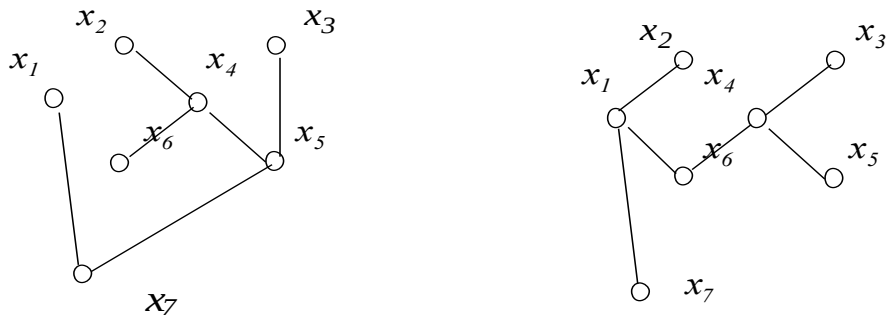


Рис.3.33

Существует ряд способов генерирования всех остовов для заданного графа. Например, если получен один остовный подграф исходного графа G, то следующий остов можно получить при добавлении одного и исключении другого ребра.

Такую процедуру формирования всех остовных деревьев называют ***элементарными преобразованиями графов***.

3.5. Задачи на графах

3.5.1. Нахождение кратчайшего остова

Пусть задан взвешенный связный граф $G=(X,A)$, (рис.3.34). Требуется найти кратчайшее остовное дерево (КОД), в котором сумма всех весов ребер наименьшая (рис.3.35).

Такая задача возникает, например, при построении сети трубопроводов между населенными пунктами, соединении клемм электрической цепи проводом наименьшей длины и т.д.

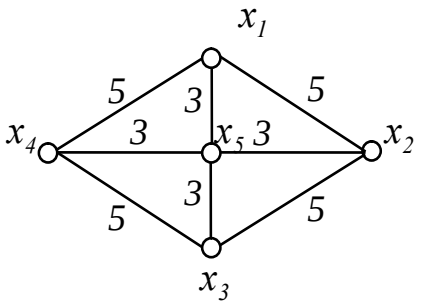


Рис.3.34

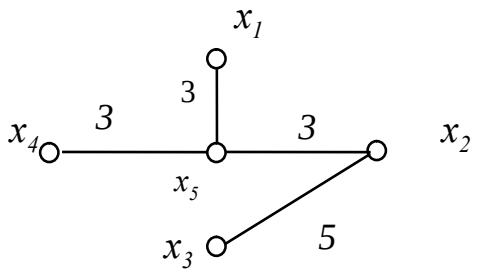


Рис.3.35

В основе формальной процедуры решения задачи поиска КОД лежит типовая операция – поиск ребра с наименьшим весом, соединяющего два поддерева T_i и T_j по правилу

$$\Delta_{ij} = \min_{x_i \in T_i,\ x_j \in T_j} \left\{ \min \left[c(x_i, x_j) \right] \right\}.$$

Процесс формирования КОД осуществляется по шагам. На каждом шаге к ранее построенной части дерева добавляется очередное кратчайшее ребро.

Алгоритм Прима. Предположим, что T_S – часть искомого КОД – уже сформирована. В начальный момент в качестве T_S принимается первая взятая вершина.

Далее каждой вершине $x_j \notin T_S$ присваиваются пометки $x_j = \left(a_j, b_j\right)$, где a_j – ближайшая к x_j вершина $x_i \in T_S$ (т.е.

вершина, для которой $c(x_i, x_j) = \min$), а b_j – это c_{ij} – вес ребра, соединяющего (a_j, x_j).

В основе алгоритма лежит пошаговая процедура. На каждом шаге определяется вершина x_j^*, которая имеет наименьшую пометку b_j и эта вершина добавляется к T_S.

После добавления ребра (a_j, x_j^*) осуществляется корректировка пометок вершин, оставшихся свободными, и процедура повторяется.

Указанные операции выполняются в следующем порядке:

Шаг 1. *Формирование* T_S. При этом произвольно выбирается первая вершина x_S и формируется дерево, в котором $T_S = \{x_S\}$ - множество его вершин, и $A_S = \varnothing$ - множество его дуг .

Шаг 2. *Начальная разметка.*

1. Для каждой $x_j \notin T_S$ определяется ближайшая $x_i \in T_S$ такая, что $c(x_i, x_j) = \min\limits_{x_i \in T_S} [c(x_i, x_j)] = b_j$.

2. Каждому x_j присваиваются пометки:

а) если x_i и x_j соединены ребром, то вершине x_j присваиваются пометки (a_j, b_j), где a_j – ближайшая вершина x_i, а b_j – расстояние до нее;

б) если из x_j нет ребра в x_i, то вершине x_j присваиваются пометки $(0, \infty)$.

Шаг 3. *Выбор ближайшей к* T_S вершины x_j^*. При этом выбирается такая x_j^*, в которой $b_j^* = \min(b_j)$.

Шаг 4. *Расширение* T_S. В T_S включается новая вершина:

$T_S = T_S \cup \{x_j^*\}$ $A_S = A_S \cup \{(a_j, x_j^*)\}$.

Шаг 5. *Проверка условия завершения построения КОД*. Если $|T_S| = n$, то граф сформирован. В противном случае процесс поиска продолжается.

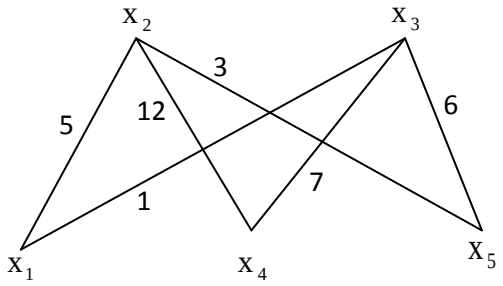
Шаг 6. *Обновление пометок.* Для всех $x_j \in \Gamma\left(x_i^*\right)$, где $x_j \notin T_s$, а $\Gamma(x_i)$ – окрестность последней включенной в граф вершины, производится обновление пометок, если вес ребра $c\left(x_i^*, x_j\right)$ меньше, чем b_j; при этом принимается

$a_j = x_i^*$,

$b_j = c\left(x_i^*, x_j\right)$.

После этого вновь выполняется шаг 3.

Пример. Используя алгоритм Прима, построить кратчайший остов для заданного графа.



Решение.

Шаг 1. Присвоение начальных значений. $T_s = \{x_1\}$, $A_s = \varnothing$, n = 5.

Шаг 2. Присвоение пометок вершинам.

$x_2\ [x_1,5]$, $x_3\ [x_1,1]$, $x_4\ [0,\infty]$, $x_5\ [0,\infty]$.

Шаг 3. Находим вершину с минимальным значением пометки - $x_3\ [x_1,1]$.

Вносим изменения в исходные множества.

$$T_s = T_s \cup min = \{x_1\} \cup \{x_3\} = \{x_1, x_3\},\ A_s = \{(x_1, x_3)\}.$$

Шаг 4. Изменяем значения пометок с учетом изменения структуры строящегося дерева $\Gamma(x_3) = \{x_4, x_5\}$, следовательно $x_4\ [x_3,7]$, $x_5\ [x_3,6]$.

Шаг 5. Проверка на окончание работы алгоритма $|T_s| = 2 < 5$, т.к. условие окончания не выполнилось, происходит переход на шаг 3 алгоритма.

Дальнейшие шаги имеют тот же смысл, что и описанные, поэтому комментариями они снабжаться не будут.

Шаг 3. Минимум - $x_5\ [x_3,6]$,

$$T_s = T_s \cup min = \{x_1, x_3, x_5\}, \quad A_s = \{(x_1, x_3), (x_3, x_5)\}.$$

Шаг 4. $\Gamma(x_5) = \{x_2\}$, следовательно - x_2 $[x_5, 3]$.

Шаг 5. Проверка на окончание. $|T_s| = 3 < 5$.

Шаг 3. Минимум - x_2 $[x_5, 3]$,

$$T_s = T_s \cup min = \{x_1, x_3, x_5, x_2\}, \; A_s = \{(x_1, x_3), (x_3, x_5), (x_5, x_2)\}.$$

Шаг 4. $\Gamma(x_2) = \{x_4\}$, следовательно - x_4 $[x_3, 7]$.

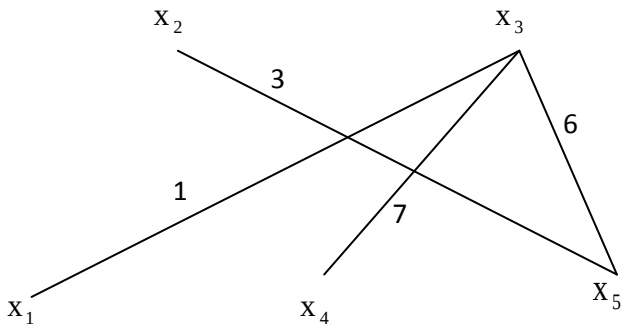
Шаг 5. Проверка на окончание. $|T_s| = 4 < 5$.

Шаг 3. Минимум - x_4 $[x_3, 7]$,

$$T_s = T_s \cup min = \{x_1, x_3, x_5, x_2, x_4\}, A_s = \{(x_1, x_3), (x_3, x_5), (x_5, x_2), (x_3, x_4)\}.$$

Шаг 4. $\Gamma(x_2) = \varnothing$, следовательно, пометки не изменяются.

Шаг 5. $|T_s| = 5 = 5$. Выполнилось условие завершения алгоритма, значит в множестве A_s содержатся все дуги, образующие остов. Минимальный остов имеет вид:



Алгоритм Краскала. Алгоритм построения кратчайшего остовного дерева методом Краскала основан на первоначальном упорядочивании всех ребер исходного графа в порядке возрастания их весов. В процессе построения кратчайшего остова из упорядоченного списка последовательно выбираются ребра, и очередное ребро включается в состав формируемого остова, если добавление данного ребра к ранее построенной части графа не приводит к появлению цикла (в противном случае формируемый граф не будет являться деревом). В отличие от алгоритма Прима, при использовании которого строится единое дерево, применение алгоритма Краскала предусматривает возможность появления в процессе работы алгоритма нескольких деревьев, которые впоследствии сращиваются (при добавлении

следующих ребер), образуя в конечном счете искомый остов с минимальным суммарным весом ребер.

Для того, чтобы исключить образование циклов при добавлении новых ребер, каждой вершине графа x_i приписываются пометки (r_i, p_i), где r_i - корень поддерева, содержащего вершину x_i, а p_i - исходный узел, для которого вершина x_i является порожденным узлом. Тогда, если в процессе проверки очередного ребра, которое соединяет две некоторые вершины графа, окажется, что обе вершины имеют один и тот же корень, то данное ребро нельзя включать в остов, так как это приводит к образованию цикла.

Построение кратчайшего остовного дерева производится в следующем порядке.

Шаг 1. *Начальные установки.* Формируем множество вершин T_s и множество ребер A_s строящегося остова. На данном этапе $T_s=\varnothing$; $A_s=\varnothing$. Всем вершинам приписываются начальные значения пометок: $x_i \rightarrow (x_i, 0)$.

Шаг 2. *Упорядочивание ребер.* Формируем упорядоченный список ребер (в порядке неубывания их весов).

Шаг 3. *Выбор очередного ребра.* Выбираем первое в списке из не просмотренных ребро $a_к=(x_i,x_j)$.

Шаг 4. *Проверка возможности появления цикла при подключении ребра.* Проверка осуществляется сравнением пометок r_i и r_j вершин, которые соединяет рассматриваемое ребро. Если значения корней в пометках этих вершин не совпадают, то выполняется переход на шаг 6. Если же $r_i= r_j$, то включение в строящийся граф рассматриваемого ребра приводит к циклу, что недопустимо, в этом случае выполняется переход на шаг 5.

Шаг 5. *Исключение рассматриваемого ребра*. Ребро, приводящее к возникновению цикла, отбрасывается, и производится переход к шагу 3 (переход к рассмотрению очередного ребра).

Шаг 6. *Включение выбранного ребра в КОД и корректировка пометок вершин остова*. Ребро $a_к$, включение которого не приводит к образованию цикла, добавляется к сформированному дереву $A_s=A_s \cup\{a_к\}$. В

множество T_s включаются концевые вершины ребра и другие я вершины сращиваемых поддеревьев.

Если добавляемым ребром объединяются два сформированных ранее поддерева, то у них разные корни, и после объединения у одного из них пометки сохраняются, а у второго – пометки необходимо корректировать. В качестве корня у всех вершин нового дерева принимается корень первого поддерева, а для корректировки значений p_i применяется специальный алгоритм изменения порядка предшествования узлов дерева.

Шаг 7. *Проверка условия завершения построения КОД*. Число ребер в дереве $m=n-1$, где n – число вершин в исходном графе. Поэтому, если $A_s=n-1$, то построение кратчайшего остова можно считать законченным, в противном случае выполняется переход на шаг 3.

Алгоритм изменения порядка предшествования узлов основан на последовательном просмотре вершин второго поддерева (при движении в направлении от корня первого поддерева) и фиксации в качестве исходного узла той вершины, которая рассматривалась на предыдущем шаге.

Пусть включаемое в остов ребро соединяет вершину первого поддерева $x_i \rightarrow (p_i,r)$ с вершиной второго поддерева $x_j \rightarrow (p_j,s)$. Вершина x_j полагается текущей. Корректировка значений p_j этой и последующих вершин производится в следующем порядке.

Шаг 1. *Определение следующей вершины корректируемого поддерева*. В качестве следующей вершины z полагается та, которая ранее была исходной (ближе к старому корню второго поддерева) $z=p_j$.

Шаг 2. *Корректировка пометки исходного узла для текущей вершины корректируемого поддерева.* Для текущего узла исходной становится вершина $x_i : p_j=x_i$.

Шаг 3. *Проверка условия завершения корректировки пометок*. Если рассматриваемая вершина является последней во втором поддереве, т.е. его корнем ($x_j=s$), процесс корректировки пометок завершается. В противном случае выполняется следующий шаг.

Шаг 4. *Фиксация рассмотренной вершины, как предшествующей*. Текущий узел считается пройденным, и производится переименование предыдущего узла: $x_i=x_j$.

Шаг 5. *Переход к корректировке пометок следующей вершины*. Следующая вершина, определенная на шаге 1, становится текущей: $x_j=z$. После этого выполняется переход на шаг 1.

3.5.2. Задача определения кратчайшего пути из одной вершины графа в другие

Пусть дан граф $G=(X,A)$, дугам которого приписаны веса $c=\{c_{ij}\}$. Задача о кратчайшем пути состоит в нахождении пути наименьшей стоимости между начальной вершиной S и конечной вершиной t.

Алгоритм Дейкстры. Он является эффективным алгоритмом для случая, когда все веса положительны. Сущность алгоритма состоит в выполнении пошаговой корректировки пометок, которые присваиваются каждой вершине и определяют верхнюю границу длины пути от исходной вершины S к данной. На каждом шаге значение пометок уменьшается. Одна из них на шаге i становится постоянной, и это означает, что получено значение длины кратчайшего пути от S до данной вершины.

Решение задачи осуществляется в следующем порядке:

Шаг 1. *Присваивание начальных значений.*

1. Длина пути из вершины S в текущую вершину $l(S)=0$. Для всех $x_i \neq S$ $l(x_i)=\infty$. Эти пометки считаются временными.

2. Устанавливается $p=S$, где p – текущая вершина кратчайшего пути.

Шаг 2. *Поиск окрестности p и обновление пометок.*

1. Определяется список вершин $x_i \in \Gamma(p)$ с временными пометками.

2. Корректируются пометки этих вершин:

$$l(x_i) = \min_{x_i \in \Gamma(p)} [l(x_i), l(p) + c(p, x_i)].$$

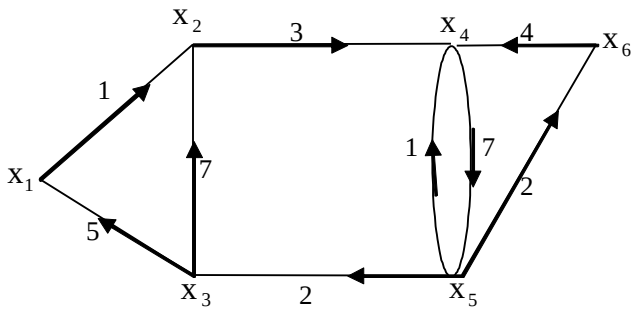
Шаг 3. *Определение текущей вершины кратчайшего пути* x_i^*. Среди всех вершин с временными пометками находим x_i^* такую, что

$$l(x_i^*) = \min[l(x_i)].$$

Шаг 4. *Переопределение* p. Найденную вершину считаем текущей вершиной кратчайшего пути: $p = x_i^*$. Пометка при x_i^* становится постоянной.

Шаг 5. *Проверка достижения конечной вершины* t. Если вершина t достигнута, процесс завершается, в противном случае выполняется переход на шаг 2.

Пример. Для заданного графа, используя алгоритм Дейкстры, найти кратчайшие пути из x_1 во все остальные вершины.



Решение.

Шаг 1. Присваивание начальных значений пометок вершинам $l(x_1) = 0,\ p = x_1,\ x_i \neq x_1,\ l(x_i) = \infty$. p - текущая вершина кратчайшего пути.

Результаты присвоения пометок будут занесены в специальную таблицу, где указан шаг итерации и значение пометки на этом шаге. Символом "+" в таблице будут отмечены постоянные пометки.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-1	0	∞	∞	∞	∞	∞
Признак постоянной	+					

пометки						

Шаг 2. Поиск окрестности p и обновление пометок. $\Gamma(p)=\Gamma(x_1)=\{x_2\}$, $l(x_2)=min(\infty;\ 0+1)=1$.

Шаг 3 и 4. Определение текущей вершины кратчайшего пути и переопределение p: $p=x_2$.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-2	0	1	∞	∞	∞	∞
Признак постоянной пометки	+	+				

Шаг 5. Так как в графе еще присутствуют вершины, имеющие временные пометки, то алгоритм свою работу не закончил и осуществляется переход на шаг 2.

Шаг 2. $\Gamma(x_2)=\{x_4\}$, $l(x_4)=min(\infty;\ 1+3)=4$.

Шаг 3 и 4. $p=x_4$.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-3	0	1	∞	4	∞	∞
Признак постоянной пометки	+	+		+		

Шаг 5. Условие завершения не выполнилось, алгоритм свою работу не закончил, переход на шаг 2.

Шаг 2. $\Gamma(x_4)=\{x_5\}$, $l(x_5)=min(\infty;\ 4+7)=11$.

Шаг 3 и 4. $p=x_5$.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-4	0	1	∞	4	11	∞
Признак постоянной пометки	+	+		+	+	

Шаг 2. $\Gamma(x_5)=\{x_3, x_6\}$, $l(x_3)=min(\infty;\ 11+2)=13$, $l(x_6)=min(\infty;11+2)=13$.

Шаг 3 и 4. $p = x_3$. Если несколько вершин имеют одинаковые минимальные значения, то можно выбрать любую из них.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-5	0	1	13	4	11	13
Признак постоянной пометки	+	+	+	+	+	

Шаг 5. Условие завершения не выполнилось, алгоритм свою работу не закончил, переход на шаг2.

Шаг 2. $\Gamma(x_3) = \{x_2, x_1\}$. Так как пометки всех вершин, попавших в рассматриваемую окрестность, постоянны, то они не изменяют свои значения.

Шаг 3 и 4. $p = x_6$.

Вершина	x_1	x_2	x_3	x_4	x_5	x_6
Шаг итерации-6	0	1	13	4	11	13
Признак постоянной пометки	+	+	+	+	+	+

Шаг 5. Так как в графе все вершины имеют постоянные пометки, то алгоритм свою работу закончил, а пометки вершин отражают значения кратчайших путей из x_1, во все остальные вершины.

Алгоритм поиска кратчайшего пути для случая общей матрицы весов. Данный алгоритм используется для поиска кратчайшего пути из вершины S_0 в другие вершины, когда веса ребер могут быть отрицательными.

Отметим, если в графе G существует цикл с отрицательным суммарным весом, то кратчайшего пути не существует.

Рассматриваемый метод также является итерационным, при этом используются пометки $e^k(x_i)$ – длина дуги до вершины x_i на k-й итерации, однако никакие из них в процессе решения не рассматриваются как окончательные.

Порядок решения задачи:

Шаг 1. *Присваивание начальных значений.* Введем область S – множество активных вершин. На первом шаге принимаем $S=\Gamma(S_0)$, $k=1$, $l^1(S_0)=0$, $l^1(x_i)=c(S_0,x_i)$ для всех $x_i \in \Gamma(S_0)$; для всех прочих x_i $l^1(x_i)=\infty$.

Шаг 2. О*бновление пометок на очередной итерации* $k+1$.Для каждой вершины $x_i \in \Gamma(S)$ изменяется ее пометка следующим образом:

$$l^{k+1}(x_i)=\min_{x_j \in T_i}\left[l^k(x_i), \min\left\{l^k(x_j)+c(x_i,x_j)\right\}\right],$$

где $T_i=\Gamma^{-1}(x_i)\cap S$.

Для других вершин $x_i \notin \Gamma(S), l^{k+1}(x_i)=l^k(x_i)$.

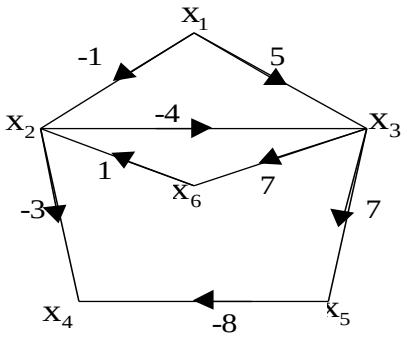
Шаг 3. *Проверка на окончание.* Если в результате предыдущего шага не изменила значение ни одна пометка, т.е. $l^{k+1}(x_j)=l^k(x_j)$ для всех вершин, то процесс окончен. Получены значения кратчайших путей во все вершины. Если хотя бы для одной это равенство не выполняется, то процесс поиска продолжается после проверки условия $k<n-1$, где n – число вершин в графе. Если же $k=n-1$, то мы прошли все вершины, но не получили окончательного ответа. Это значит, что в графе имеет место цикл с отрицательным весом и задача не имеет решения.

Шаг 4. *Обновление* S. В множество активных вершин включаются те новые вершины, пометки которых изменились в текущей итерации:

$$S=S\cup\left\{x_i \mid l^{k+1}(x_i)\neq l^k(x_i)\right\}.$$

После этого осуществляется переход на шаг 2.

Пример. Используя алгоритм для общей матрицы весов, найти кратчайшие пути из x_1 во все остальные вершины графа.



Решение.

Шаг 1. Присваивание начальных значений. $l^1(x_1)=0, K=1, S=\Gamma(x_1)=\{x_2, x_3\}$, $l^1(x_2)=-1$, $l^1(x_3)=5$. Для всех остальных вершин графа, таких что $x_i \notin S$, $l^1(x_i)=\infty$. Результаты расстановки пометок занесем в таблицу.

Шаг итерации	x_1	x_2	x_3	x_4	x_5	x_6
1	0	-1	5	∞	∞	∞

Шаг 2. Обновление пометок на очередной итерации. Определяется множество вершин, для которых будут определены новые значения пометок

$\Gamma(S)=\Gamma(\{x_2, x_3\})=\{x_4, x_3, x_6, x_5\}$.

Для каждой из них значение пометки вычисляется по формуле

$$l^{k+1}(x_i)=\min_{x_j \in T_i}\left[l^k(x_i), \min\left\{l^k(x_j)+c(x_i,x_j)\right\}\right],$$

где $T_i=\Gamma^{-1}(x_i)\cap S$.

Применим формулу.

$T_4=\Gamma^{-1}(x_4)\cap S=\{x_2\}$, $l^2(x_4)=min(\infty,-1+(-3))=-4$;

$T_3=\Gamma^{-1}(x_3)\cap S=\{x_1, x_2\}$, $l^2(x_3)=min(5,min((-1)+(-4),0+5)=-5$;

$T_6=\Gamma^{-1}(x_6)\cap S=\{x_3\}$, $l^2(x_6)=min(\infty,5+7)=12$;

$T_5=\Gamma^{-1}(x_5)\cap S=\{x_3\}$, $l^2(x_5)=min(\infty,min(5+7)=12$.

Остальные вершины пометки не изменят. Таблица значений пометок примет вид.

Шаг итерации	x_1	x_2	x_3	x_4	x_5	x_6
1	0	-1	5	∞	∞	∞
2	0	-1	-5	-4	12	12

Шаг 3. Проверка на окончание. По таблице видно, что существуют вершины, для которых имеет место условие $l^k(x_i) \neq l^{k+1}(x_i)$, т.е. пометки предшествующего и последующего шагов итерации отличаются друг от друга. При этом выполняется и другое условие $k=1<5$, это означает, что алгоритм работу не закончил и осуществляется переход на шаг 4.

Шаг 4. *Обновление* S. Множество S образуют те вершины, которые изменили свои пометки. $S = \{x_3, x_4, x_5, x_6\}$.

Шаг 5. Увеличение счетчика итераций $k=k+1=2$, после этого осуществляется переход на шаг 2.

Шаг 2. $\Gamma(S) = \{x_6, x_4, x_2, x_5\}$

$T_6 = \Gamma^{-1}(x_6) \cap S = \{x_3\},\ l^3(x_6) = min(12, -5+7) = 2;$

$T_4 = \Gamma^{-1}(x_4) \cap S = \{x_5\},\ l^3(x_4) = min(-4, 12+(-8)) = -4;$

$T_2 = \Gamma^{-1}(x_2) \cap S = \{x_6\},\ l^3(x_2) = min(-1, 12+1) = -1;$

$T_5 = \Gamma^{-1}(x_5) \cap S = \{x_3\},\ l^3(x_5) = min(12, -5+7) = 2.$

Шаг итерации	x_1	x_2	x_3	x_4	x_5	x_6
1	0	-1	5	∞	∞	∞
2	0	-1	-5	-4	12	12
3	0	-1	-5	-4	2	2

Шаг 3. Существуют x_i такие, что $l^k(x_i) \neq l^{k+1}(x_i)$ и $k = 2 < 5$.

Шаг 4. $S = \{x_5, x_6\}$.

Шаг 5. $k=k+1=3$, переход на шаг 2.

Шаг 2. $\Gamma(S) = \{x_4, x_2\}$

$T_4 = \Gamma^{-1}(x_4) \cap S = \{x_5\},\ l^4(x_4) = min(-4, 2-8) = -6;$

$T_2 = \Gamma^{-1}(x_2) \cap S = \{x_6\},\ l^4(x_2) = min(-1, 2+1) = -1.$

Шаг итерации	x_1	x_2	x_3	x_4	x_5	x_6
1	0	-1	5	∞	∞	∞
2	0	-1	-5	-4	12	12
3	0	-1	-5	-4	2	2
4	0	-1	-5	-6	2	2

Шаг 3. Существуют x_i такие, что $l^k(x_i) \neq l^{k+1}(x_i)$ и $k = 3 < 5$.

Шаг 4. $S = \{x_4\}$.

Шаг 5. $k=k+1=4$.

Шаг 2. $\Gamma(S) = \varnothing$, т.е. на этом шаге не может быть изменена ни одна из пометок вершин графа.

Шаг итерации	x_1	x_2	x_3	x_4	x_5	x_6
1	0	-1	5	∞	∞	∞
2	0	-1	-5	-4	12	12
3	0	-1	-5	-4	2	2
4	0	-1	-5	-6	2	2
5	0	-1	-5	-6	2	2

Шаг 3. $k < 5$, при этом для всех x_i выполняется $l^k(x_i) = l^{k+1}(x_i)$. Значит, алгоритм закончил свою работу, и последняя строка таблицы содержит пометки вершин, отражающие значения кратчайших путей из x_1, во все остальные вершины.

Алгоритм Флойда, для нахождения кратчайшие пути между всеми парами вершин. Если необходимо найти кратчайшие пути между всеми парами вершин, то можно n раз применить предыдущий алгоритм, считая очередную вершину графа начальной. При большой размерности графа это приводит к большому объему вычислений.

Разработанный для решения данной задачи алгоритм Флойда основан на использовании матрицы весов C, элементы которой на k-й итерации определяют длины кратчайших путей между каждой парой (x_i, x_j), в которых промежуточные вершины принадлежат подмножеству $\{x_1, x_2, ..., x_k\}$.

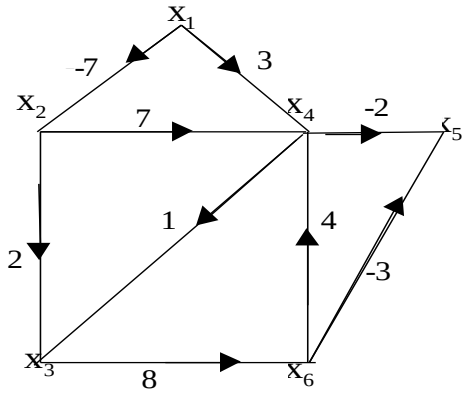
Шаг 1. Вводится матрица весов $c = \lfloor c_{ij} \rfloor$. Для смежных вершин c_{ij} принимаем равным весу дуги c_{ij}, а если вершины x_i и x_j не смежны, то $c_{ij} = \infty$, $c_{ii} = 0$. Далее производится последовательный перебор всех вершин x_k (сначала принимаем $k = 1$).

Шаг 2. Производим проверку пар (x_i, x_j), которые связываются через x_k: для всех $i \neq k$ и для всех $j \neq k$ вычисляется

$c_{ij} = \min\left[c_{ij}, (c_{ik} + c_{kj})\right]$.

Шаг 3. Проведем проверку на окончание. Если $k < n$, то k увеличивается на *1* и производится переход на шаг 2. Если $k = n$ и все диагональные элементы $c_{ij} \geq 0$, то получена матрица кратчайших путей C. Если $k = n$ и хотя бы один диагональный элемент $c_{ij} < 0$, то задача не имеет решения, т.к. существует контур отрицательной длины.

Пример. Используя алгоритм Флойда, найти кратчайшие пути между всеми парами вершин графа.



Решение.

Шаг 1. Строится исходная матрица весов. Полагается *k=1*.

k=1	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	-7	∞	3	∞	∞
x_2	∞	0	2	7	∞	∞
x_3	∞	∞	0	∞	∞	8
x_4	∞	∞	1	0	-2	∞
x_5	∞	∞	∞	∞	0	∞
x_6	∞	∞	∞	4	-3	0

Шаг 2. Для всех пар (x_i, x_j), которые связываются через x_k, таких что $i \neq k$ и $j \neq k$, и $c_{ik} \neq \infty$, $c_{kj} \neq \infty$, вычисляются c_{ij}. Здесь

и в дальнейшем будут приводиться значения только для тех коэффициентов, которые не равны бесконечности.

c_{ik} , $i \neq k$: $c_{21}, c_{31}, c_{41}, c_{51}, c_{61}$.

c_{kj} , $j \neq k$: $c_{12} = -7, c_{13}, c_{14} = 3, c_{15}, c_{16}$.

Так как на этом шаге все $c_{ik} = \infty$, то матрица не изменится и осуществляется переход на шаг 3.

Шаг 3. Проверка на окончание. Так как $k=1<6$, то $k=k+1=2$, переходим на шаг 2 алгоритма.

Шаг 2. c_{ik} , $i \neq k$: $c_{12} = -7, c_{32}, c_{42}, c_{52}, c_{62}$.

c_{kj} , $j \neq k$: $c_{21}, c_{23} = 2, c_{24} = 7, c_{25}, c_{26}$.

Определяются новые значения коэффициентов матрицы.

$c_{13} = min[\infty; -7 + 2] = -5$; $c_{14} = min[3; -7 + 7] = 0$.

k=2	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	-7	-5	0	∞	∞
x_2	∞	0	2	7	∞	∞
x_3	∞	∞	0	∞	∞	8
x_4	∞	∞	1	0	-2	∞
x_5	∞	∞	∞	∞	0	∞
x_6	∞	∞	∞	4	-3	0

Шаг 3. Проверка на окончание. Так как $k=2<6$, то $k=k+1=3$, переходим на шаг 2 алгоритма.

Шаг 2. c_{ik} , $i \neq k$: $c_{13} = -5, c_{23} = 2, c_{43} = 1, c_{53}, c_{63}$.

c_{kj} , $j \neq k$: $c_{31}, c_{32}, c_{34}, c_{35}, c_{36} = 8$.

$c_{16} = min[\infty; -5 + 8] = 3$; $c_{26} = min[\infty; 8 + 2] = 10$; $c_{46} = min[\infty; 1 + 8] = 9$.

k=3	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	-7	-5	0	∞	3
x_2	∞	0	2	7	∞	10
x_3	∞	∞	0	∞	∞	8
x_4	∞	∞	1	0	-2	9
x_5	∞	∞	∞	∞	0	∞
x_6	∞	∞	∞	4	-3	0

Шаг 3. Проверка на окончание. Так как $k=3<6$, то $k=k+1=4$, переходим на шаг 2 алгоритма.

Шаг 2. $c_{ik}\ ,\ i \neq k:\ c_{14}=0, c_{24}=7, c_{34}, c_{54}, c_{64}=4.$

$c_{kj}\ , j \neq k: c_{41}, c_{42}, c_{43}=1, c_{45}=-2, c_{46}=9.$

$c_{13}=min[-5;0+1]=-5;\ c_{15}=min[\infty;0+(-2)]=-2;\ c_{16}=min[3;0+9]=3;$

$c_{23}=min[2;7+1]=2;\ c_{25}=min[\infty;7+(-2)]=5;\ c_{26}=min[10;7+9]=10;$

$c_{63}=min[\infty;4+1]=5;\ c_{65}=min[-3;4+(-2)]=-3;\ c_{66}=min[0;4+9]=0.$

k=4	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	-7	-5	0	-2	3
x_2	∞	0	2	7	5	10
x_3	∞	∞	0	∞	∞	8
x_4	∞	∞	1	0	-2	9
x_5	∞	∞	∞	∞	0	∞
x_6	∞	∞	5	4	-3	0

Шаг 3. Проверка на окончание. Так как $k=4<6$, то $k=k+1=5$, переходим на шаг 2 алгоритма.

Шаг 2. $c_{ik}\ ,\ i \neq k:\ c_{15}=-2, c_{25}=5, c_{35}, c_{45}=-2, c_{65}=-3;$

$c_{kj}\ , j \neq k: c_{51}, c_{52}, c_{53}, c_{54}, c_{56}.$

Так как на этом шаге все $c_{kj}=\infty$, то матрица не изменится, т.е. матрица для $k=5$ совпадает с матрицей для $k=4$, осуществляется переход на шаг 3.

Шаг 3. Проверка на окончание. Так как $k=5<6$, то $k=k+1=6$, переходим на шаг 2 алгоритма.

Шаг 2. $c_{ik}\ ,\ i \neq k:\ c_{16}=3, c_{26}=10, c_{36}=8, c_{46}=9, c_{56};$

$c_{kj}\ , j \neq k:\ c_{61}, c_{62}, c_{63}=5, c_{64}=4, c_{65}=-3.$

$c_{13}=min[-5;3+5]=-5;\ c_{14}=min[0;3+\ ]=0;\ c_{15}=min[-2;3+(-3)]=-2;$

$c_{23}=min[2;10+5]=2;\ c_{24}=min[7;10+4]=7;\ c_{25}=min[5;10-3]=5;$

$c_{33}=min[0;8+5]=0;\ c_{34}=min[\infty;8+4]=12;\ c_{35}=min[\infty;8-3]=5;$

$c_{43}=min[1;9+5]=1;\ c_{44}=min[0;9+4]=0;\ c_{45}=min[-2;9-3]=-2.$

K=6	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	-7	-5	0	-2	3
x_2	∞	0	2	7	5	10

x_3	∞	∞	0	12	5	8
x_4	∞	∞	1	0	-2	9
x_5	∞	∞	∞	∞	0	∞
x_6	∞	∞	5	4	-3	0

Полученная матрица является решением поставленной задачи и отображает значения кратчайших путей между всеми парами вершин заданного графа.

3.6. Транспортные сети

С помощью графов можно описывать функционирование систем, в которых передается какой-либо продукт.

Ориентированный граф $G=(X, A)$ называется ***транспортной сетью*** (рис.3.36), если:

а) каждой дуге a_i приписывается положительное число $c(a_i)$, называемое ***пропускной способностью*** дуги;

б) в графе выделены две вершины S и t, которые называют соответственно ***истоком*** и ***стоком***, при этом в графе нет дуг, заходящих в S и исходящих из t.

Например, граф описывает систему железнодорожную или вычислительную сеть, по дугам которой передается продукт, может быть представлен как транспортная сеть.

В железнодорожной сети вершины – это станции, а по дугам передвигаются поезда.

В вычислительной сети вершинами являются узлы обработки информации, передаваемым продуктом является информация.

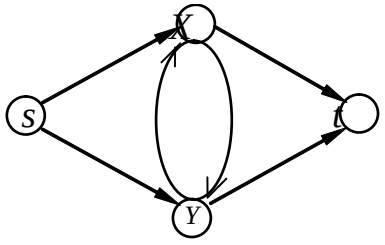


Рис.3.36

Введем следующие понятия:

Пропускная способность дуги $c(a)$ – это максимальное количество продукта, которое можно переместить по этой дуге в единицу времени.

Поток $\varphi(a)$– это фактическое количество продукта, передаваемое по дуге a в единицу времени.

Величина потока Ф – это общая величина продукта, выходящего из вершины S и прибывающего в вершину t.

Возьмем некоторую вершину X. Обозначим множество входящих в нее дуг A_X^+, а множество исходящих дуг – A_X^-.

Основные условия существования потока следующие:

1) на любой дуге $a \in A$ поток

$$\varphi(a) \geq 0;$$

2) поток не может превышать пропускной способности дуги

$$\varphi(a) \leq c(a);$$

3) для всех вершин графа, за исключением $X = S$ и $X = t$,

$$\sum_{a \in A_X^+} \varphi(a) = \sum_{a \in A_X^-} \varphi(a). \qquad (3.1)$$

Последнее условие называется ***условием сохранения потока*** (количество продукта, приходящего в какую-либо вершину, всегда равно количеству продукта, выходящего из него);

4) общее количество продукта, передаваемого в единицу времени из S в t:

$$\Phi = \sum_{a \in A_S^-} \varphi(a) = \sum_{a \in A_t^+} \varphi(a).$$

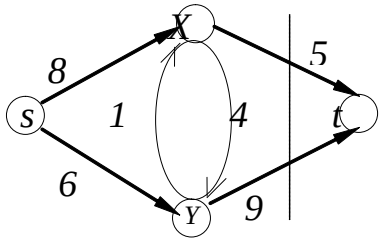


Рис.3.37

Пример возможного потока в сети показан на рис 3.37, при этом

$$\Phi = 5 + 9 = 14.$$

В транспортной сети может быть несколько потоков одновременно. Традиционной задачей, которая решается на таких графах, является организация такого перемещения по сети, чтобы количество передаваемого из S в t груза было максимальным.

Задача нахождения максимального потока при условии не превышения пропускной способности дуг. Введем понятие ***разреза***. Если все множество вершин сети разбить на две части N и $\overline{N}$ таким образом, что $S \in N$, а $t \in \overline{N}$, то множество дуг $\left(N, \overline{N}\right)$, ведущих из одной части в другую, называется ***разрезом***:

$$\left(N, \overline{N}\right) = \left\{(x,y) \,\middle|\, x \in N, y \in \overline{N}\right\}.$$

На рис. 3.37 пунктирной линией показан разрез, в котором $N = \{S, X\}$.

Суммарная пропускная способность дуг, входящих в разрез, называется ***пропускной способностью разреза*** и обозначается $C(N, \overline{N})$.

Лемма 3.2. Величина потока в сети не может превышать пропускной способности любого разреза, $\Phi \le C(N, \overline{N})$.

Доказательство:

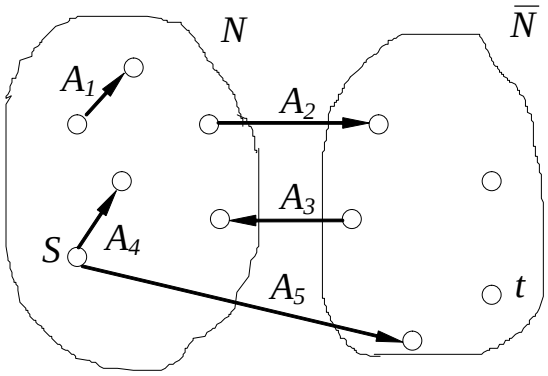


Рис.3.39

Рассмотрим множество вершин N (рис.3.39). С учетом условия уравнения баланса (3.1) можно записать следующее выражение:

$$\sum_X \varepsilon \varphi(a) = 0 \ , \qquad (3.2)$$

где $\varepsilon = +1$ для поступающего продукта $a \in A_X^+$ и $\varepsilon = -1$ для дуг, по которым продукт покидает вершину $a \in A_X^-$.

Дуги, инцидентные вершинам N, можно разделить на пять групп:

A_1 – дуги между двумя внутренними вершинами N: $x \in N, y \in N$, $x \ne S$, здесь $\varepsilon = 0$;

A_2 – это дуги, по которым продукт уходит из N: $x \in N, y \in \overline{N}$, $x \ne S$, $\varepsilon = -1$;

A_3 – входящие дуги, для которых $x \in \overline{N}, y \in N$, для них $\varepsilon = +1$;

A_4 – подмножество дуг, которые исходят из истока S, а заканчиваются в N: $x = S, y \in N$; по этим дугам продукт поступает в N из истока, $\varepsilon = +1$;

A_5 – подмножество, которое исходит из N и заканчивается в $\overline{N}$: $x \in N,\ y \in \overline{N},\ x = S,\ \varepsilon = 1 - 1 = 0$.

Суммируя выражения для всех потоков (3.2) всех дуг, получаем:

$$\sum_{a \in A_S} \varphi(a) - \sum_{a \in (N, \overline{N})} \varphi(a) + \sum_{a \in (\overline{N}, N)} \varphi(a) = 0.$$

Первое слагаемое определяет величину потока в сети, следовательно:

$$\Phi = \sum_{a \in (N, \overline{N})} \varphi(a) - \sum_{a \in (\overline{N}, N)} \varphi(a). \qquad (3.3)$$

Отсюда следует, что

$$\Phi \le \sum_{a \in (N, \overline{N})} \varphi(a) \le \sum_{a \in (\overline{N}, N)} c(a). \qquad (3.4)$$

Так как $\sum_{a \in (N, \overline{N})} c(a) = C(N, \overline{N})$, то лемма доказана.

Теорема Форда-Фалкерсона. Наибольшая величина потока в транспортной сети равна наименьшей пропускной способности разреза. Эта теорема лежит в основе метода построения максимального потока в сети.

Сравнивая выражения (6.3) и (6.4), можно сделать вывод, что поток Φ максимальный, если выполняются два условия:

$$\begin{cases} \text{для всех } a \in (\overline{N}, N)\ \ \varphi(a) = 0, \\ \text{для всех } a \in (N, \overline{N})\ \ \varphi(a) = c(a). \end{cases} \qquad (3.5)$$

В основе алгоритма определения максимального потока лежит выбор произвольного разреза и проверка условия (3.5). Если оно не выполняется, то рассматривается возможность увеличения Φ. Для этого вводится понятие прибавочной цепи.

Прибавочная цепь – это маршрут, который связывает S и t, и ни для одной дуги этого маршрута условие (3.5) не выполняется.

Пусть между двумя смежными вершинами этого маршрута x_{i-1} и x_i имеется дуга. Если это дуга (x_{i-1}, x_i), то ее называют ***прямая*** дуга, а если это дуга обратного направления (x_i, x_{i-1}), то такую дугу называют ***обратной***.

Если для каждой прямой дуги маршрута, связывающего S и t, $\varphi(a) < c(a)$, а для каждой обратной дуги $\varphi(a) > 0$, то для потока

найдена прибавочная цепь, и можно получить новый поток больше предыдущего, если увеличить $\varphi(a)$ для всех прямых дуг и уменьшить $\varphi(a)$ для всех обратных дуг на некоторую фиксированную величину. Пример прибавочной цепи приведен на рис.3.40.

Над дугами указана через запятую пропускная способность и поток по дуге. На рис.6.41 показан этот же маршрут после увеличения величины потока.

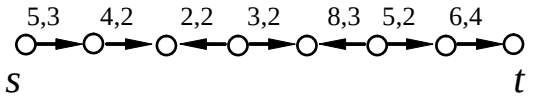


Рис.3.40

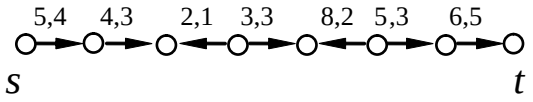


Рис.3.41

Алгоритм построения максимального потока основан на последовательном просмотре вершин сети и присвоении им пометок в следующем порядке:

Шаг 1. Помечается источник S.

Шаг 2. Выбирается одна из помеченных вершин и если соседние вершины еще не помечены, то их помечают в случае:

а) для исходящей дуги – если $\varphi(a)<c(a)$;

б) для входящей дуги – если $\varphi(a)>0$.

В просмотренной вершине пометку снимаем.

Процесс расстановки пометок продолжается до тех пор, пока:

а) не будет достигнута вершина t;

б) дуг, для которых выполняется условие (3.5), больше нет.

Если достигнута вершина t, то получена прибавочная цепь и выполняется шаг 3.

Шаг 3. Увеличивается поток на этой цепи и осуществляется переход к шагу 1.

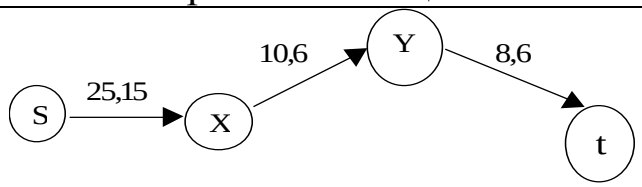
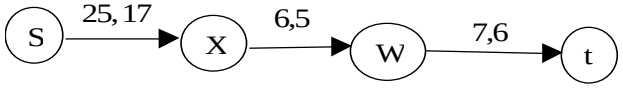
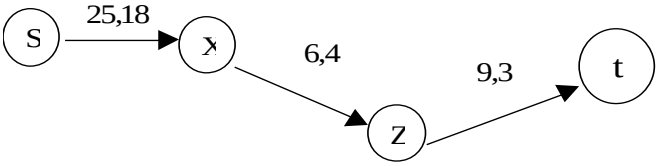
Если все помеченные вершины просмотрены, а сток t не достигнут, то получен разрез.

Шаг 4. Помеченные вершины включаются во множество N.

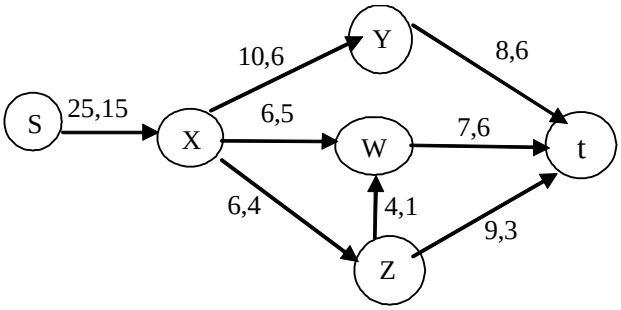
Дуги, идущие из помеченной вершины в непомеченную $a \in \left(N, \overline{N}\right)$, образуют разрез. Для каждой из них выполняется условие (3.5), иначе другой конец дуги был бы помечен.

Рассчитываем $C\left(N, \overline{N}\right) = \sum_{a \in \left(N, \overline{N}\right)} c(a) = \Phi$. Это ***максимальный поток.***

Таблица 3.1

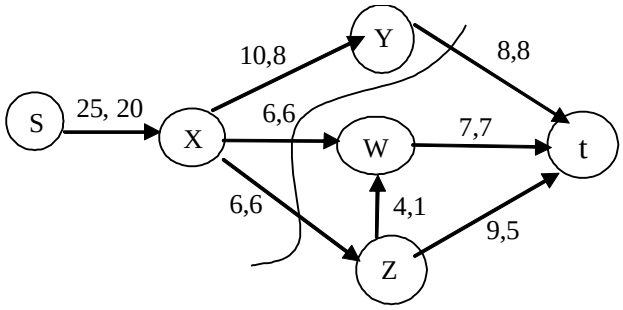
№	S	X	Y	W	Z	t	Прибавочная цепь
1	* * * *	S S S S	X X	X X	X X	Y	
2	* * * *	S S S		X X		W	
№	S	X	Y	W	Z	t	Прибавочная цепь
3	* * * *	S S S			X X	Z	
4	* * *	S S	X				Прибавочная цепь не строится, т.к. сток не получил пометку и это значит, что уже построенный поток в сети является максимальным. Требуется построить разрез и определить величину потока.

Пример. **Для заданной транспортной сети найдите максимальный поток.**



Решение. Процесс расстановки пометок вершин и построения прибавочных цепей будет отображаться в следующей табл.3.1.

Ниже приведен граф сети с новыми значениями потока на дугах. Особо помечены дуги, которые войдут в разрез, для определения величины максимального потока в сети следует просуммировать их пропускные способности.



$$\Phi = c(X,W)+c(X,Z)+c(Y,t)=6 + 6 + 8 = 20.$$

3.7. Раскраска графов

Неориентированный граф G называется r ***-хроматическим***, если его вершины могут быть раскрашены в r цветов (красок) таким образом, что никакие две смежные вершины не окрашиваются в один цвет. Наименьшее число $r=\gamma(G)$, при котором граф является r-хроматическим, называется ***хроматическим числом графа*** G.

Определение хроматического числа называют задачей раскраски графа. Раскраска разбивает все множество вершин на r подмножеств.

Рассмотрим неориентированный граф $G=(X,\Gamma)$. Множество вершин, в котором никакая пара вершин не соединена ребром, называется ***независимым множеством вершин*** (НМВ). Следовательно НМВ удовлетворяет условию

$$S \cap \Gamma(S)=\varnothing. \qquad (3.6)$$

Данное условие является формальным условием окраски всех вершин НМВ в один и тот же цвет.

Если множество S_m удовлетворяет условию (3.6) и не является подмножеством другого независимого множества

вершин, то оно называется максимальным независимым множеством вершин (МНМВ).

Для графа, изображенного на рис.3.37, приведенные ниже множества $q_0 - q_8$ являются независимыми множествами вершин, а $q_1 - q_8$ —МНМВ.

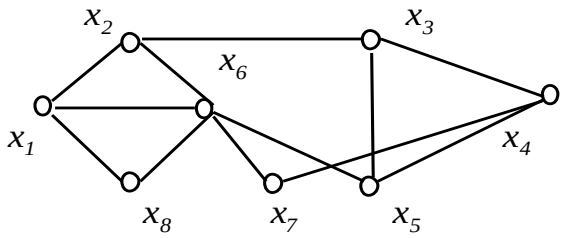


Рис.3.37

$$q_0 = \{x_2, x_7, x_8\};$$
$$q_1 = \{x_2, x_5, x_7, x_8\};$$
$$q_2 = \{x_1, x_3, x_7\};$$
$$q_3 = \{x_4, x_6\};$$
$$q_4 = \{x_3, x_6\};$$
$$q_5 = \{x_2, x_4, x_8\};$$
$$q_6 = \{x_1, x_5, x_7\};$$
$$q_7 = \{x_1, x_4\};$$
$$q_8 = \{x_3, x_7, x_8\}.$$

Если Q – семейство всех МНМВ, то число $\alpha(G) = \max\limits_{q_i \in Q} |q_i|$ называют ***числом независимости*** графа G. В нашем примере $\alpha(G) = 4$, а множество q_i, на котором этот максимум достигается, называется ***наибольшим независимым множеством*** (q_i).

Для хроматического числа $\gamma(G)$ вводятся оценки: нижняя – минимально возможное число цветов $\gamma(G) \geq \dfrac{n}{\alpha(G)}$, где n – общее число вершин, и верхняя – максимально возможное значение этого параметра

$$\gamma(G) \leq 1 + \max_{x_i \in X}[d(x_i) + 1],$$ где $d(x_i)$ – степень вершины.

Существует ряд приближенных и точных методов закраски графов.

Простой приблизительный алгоритм раскраски, основанный на упорядочении множества вершин.

1. Все вершины упорядочиваются по невозрастанию степеней. Для приведенного на рис.3.37 графа порядок вершин следующий:

$$x_6, x_1, x_2, x_3, x_4, x_5, x_7, x_8 .$$

2. Первая вершина (x_6) окрашивается в цвет 1 (например, в красный).

3. Список всех вершин просматривается сверху вниз и в цвет 1 окрашивается всякая вершина, несмежная с вершинами, окрашенными в тот же цвет (x_3).

4. Возвращаемся к первой в списке неокрашенной вершине (x_1) и окрашиваем ее в цвет 2.

5. Осуществляется окраска по тому же правилу последующих неокрашенных вершин.

6. Далее берутся следующие цвета, и процедура продолжается, пока не будет раскрашен весь граф (в приведенном примере $r = 3$).

Для получения решения, более близкого к оптимальному, описанный алгоритм может быть модифицирован: после окраски каждой вершины производится переупорядочивание неокрашенных вершин. Оставшиеся неокрашенными вершины записываются в порядке убывания их степеней, то есть степеней вершин в графе, который остается после удаления окрашенных вершин вместе с ребрами, инцендентными удаленным вершинам.

Рассмотрим способ строго математического решения задачи оптимальной раскраски.

Математическая формулировка задачи раскраски. Введем матрицу раскраски $S = \lfloor s_{ij} \rfloor$,

$$\text{где } s_{ij} = \begin{cases} 1 \text{ , если } i\text{-я вершина окрашена в } j\text{-й цвет,} \\ 0 \text{ в противном случае.} \end{cases}$$

Полагаем, что матрица смежности $A = [a_{ik}]$ задана; элемент этой матрицы $a_{ik} = 1$ если i-я и k-я вершины связаны ребром.

Тогда условия правильной раскраски могут быть математически определены следующим образом:

1. $\sum_{j=1}^{q} s_{ij} = 1$, где $q -$ верхняя оценка хроматического числа.

Это условие означает, что любая вершина может быть окрашена только в один цвет.

2. $L(1-s_{ij})-\sum_{k=1}^{n} a_{ik}s_{kj} \geq 0$, где $L-$ целое, большее n. Это условие означает, что если $s_{ij}=1$, то нет другой смежной с x_i вершины x_k, окрашенной в тот же цвет.

Тогда пусть каждому цвету поставлен в соответствие штраф p_j, и задача оптимальной раскраски графа (раскраска с наименьшим числом цветов) формулируется как задача минимизации функции $Z=\sum_{i=1}^{n}\sum_{j=1}^{q} p_j s_{ij} \to \min$. При этом дополнительно вводится условие $p_{j+1} \geq hp_j$, где h – верхняя оценка для числа вершин, которые могут быть окрашены в один цвет, то есть $h \geq \alpha(G)$.

Это условие обеспечивает то, что цвет $j+1$ не будет использован в раскраске вершин, пока еще можно использовать цвета от 1 до j.

- Известные математические методы оптимизации (в частности, метод линейного программирования) позволяют определить значения s_{ij} и таким образом решить задачу оптимальной раскраски.

Практическое использование методов раскраски.

1. Простая задача размещения (загрузки). Необходимо разместить n предметов по ящикам при наличии ограничения, связанного с недопустимостью совместного хранения некоторых предметов.

Каждому предмету ставим в соответствие вершину графа, каждому ящику ставится в соответствие j-й цвет. Если каких-либо два предмета нельзя размещать в одном ящике, то соответствующие им вершины соединяем ребром.

Задача нахождения наименьшего числа ящиков сводится к задаче раскраски полученного графа. Если емкость ящика ограничена, то необходимо ввести дополнительное условие $\sum_{i=1}^{n} s_{ij} \leq Q_j$, где Q_j – количество предметов, которые можно положить в один ящик.

2. Задача распределения ресурсов. Пусть для выполнения n работ имеется m ресурсов. Предположим, что один ресурс может одновременно использоваться только для одной работы. Для выполнения всего комплекса работ составляется расписание, в котором работы распределяются во времени по периодам.

Для составления расписания формируется граф, вершины которого соответствуют планируемым работам. Если для двух работ x_i и x_k требуется один ресурс, то вершины графа, соответствующие этим работам, соединяем ребром.

Раскраска такого графа (j-й цвет соответствует j-му периоду выполнения работ) позволяет спланировать выполнение заданного комплекса работ за минимальное время. Смежные вершины (работы, которые требуют один и тот же ресурс) при этом не могут окрашиваться в один цвет, следовательно, они выполняются в разные периоды времени.

3.8. Вопросы и задания для самоконтроля

1. Какой объект называется графом? На основании каких компонент определяется любой граф?
2. Что означает понятие смежные вершины?
3. Что называется окрестностью вершины?
4. Что понимается под степенью вершины графа? Как это понятие определяется для ориентированного и неориентированного графа?
5. Что означают термины: остовный подграф и порожденный подграф?
6. Перечислите основные типы графов. Дайте их краткую характеристику.
7. Как определяется матрица смежности графа?
8. Между какими компонентами графа устанавливается связь в матрице инциденции? От чего зависит ее размерность?
9. Как определяется матрица достижимости графа?
10. Какой граф называется деревом?
11. Какой подграф называется остовным деревом?

12. Назначение алгоритма Прима и последовательность действий при его реализации?
13. Построить граф, имеющий только положительные веса дуг и найти кратчайшие пути из некоторой выбранной вершины во все остальные. Для решения задачи использовать алгоритм Дейкстры.
14. Воспроизвести последовательность действий в алгоритме для общей матрицы весов.
15. Воспроизвести последовательность действий алгоритма Флойда.
16. Что такое транспортная сеть?
17. Определить основные условия существования потока.
18. Определить понятие разреза.
19. Что называется прибавочной цепью?
20. Воспроизвести последовательность шагов в алгоритме нахождения максимального потока в сети.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Кузнецов О.П., Адельсон-Вельский Г.Н. Дискретная математика для инженера. М.: Энергоатомиздат, 1988.
2. Горбатов В.А. Основы дискретной математики: Учеб. пособие для студентов вузов. М.: Высш. шк., 1986.
3. Яблонский С.В. Введение в дискретную математику: Учеб. пособие для вузов. М.: Наука, 1986.
4. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир, 1978.
5. Бауэр В. Введение в теорию конечных автоматов. М.: Радио и связь, 1987.
6. Темников Ф.Е. Теоретические основы информационной техники. М.: Энергия, 1971.
7. Тейз А., Грибомон П. Логический подход к искусственному интеллекту: от классической ложки к логическому программированию. М.: Мир, 1990.
8. Глаголев В.В. Основы теории систем. Методы дискретной математики. /ТПИ. Тула, 1987.
9. Клини С.К. Математическая логика. М.: Мир,1973.

10. Мендельсон Э. Введение в математическую логику. М.: Наука, 1984.
11. Грэй П. Логика, алгебра и базы данных. М.: Машиностроение, 1989.
12. Рейуорд Смит Дж. Теория формальных языков. Вводный курс. М.:Мир,1990.
13. Оре О. Теория графов. М.: Наука, 1980.
14. Гаврилов Г.П., Сапоженко А.А. Сборник задач по дискретной математике. М.: Наука, 1977.
15. Лавров И.А., Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов. М.: Наука, 1984.
16 Потоцкий С.И., Гринченков Д.В., Гордиенко А.В. Дискретная математика. Новочеркасск: ЮРГТУ(НПИ), 2001.
17. Новиков Ф.А. Дискретная математика для программистов, СПб.:Питер, 2001.
18. Цымбал В.П. Теория информации и кодирования, Киев.:Вища школа, 1982..
19. Сикорский Р. Булевы алгебры, М.:Мир, 1969.

ОГЛАВЛЕНИЕ

1. Теория множеств 3
1.1. Основные понятия теории множеств 3
1.1.1. Множества, способы задания множеств 3
1.1.2. Основные операции над множествами и их свойства 4
1.2. Прямое произведение множеств 6
1.3. Соответствия 7
1.3.1. Понятие соответствия. Типы соответствий 7
1.3.2. Связь мощности множеств, входящих в соответствие 8
1.4. Функции 10
1.4.1. Функции и отображения 10
1.4.2. Способы задания функции 12
1.5. Отношения на множествах 12
1.5.1. Понятие отношения 12
1.5.2. Типы отношений 14
1.5.3. Разбиение элементов множества на классы 16
1.5.4. Упорядочивание элементов на множествах 17
1.6. Вопросы и задания для самоконтроля 17
2. Элементы общей алгебры 18
2.1. Понятие алгебры 18
2.2. Гомоморфизм и изоморфизм 18
2.3. Полугруппы и группы 20
2.4. Алгебраические системы. Решетки 20
3. Теория графов 23
3.1. Основные понятия теории графов 23
3.1.1. Понятие графа. Области применения графов 23
3.1.2. Подграфы 29
3.1.3. Типы графов 30
3.2. Матричное представление графов. 36
3.3. Эйлеровы графы 40
3.4. Деревья 42
3.5. Задачи на графах 50
3.5.1. Нахождение кратчайшего остова 50
3.5.2. Задача определения кратчайшего пути из одной вершины графа в другие 56
3.6. Транспортные сети 68
3.7. Раскраска графов 74
3.8. Вопросы и задания для самоконтроля 78
Библиографический список 79

Учебное издание

Гринченков Дмитрий Валерьевич
Селезнев Сергей Анатольевич
Кущий Дарья Николаевна

ДИСКРЕТНАЯ МАТЕМАТИКА

Учебное пособие

Редактор *Н.А.Юшко*

Подписано в печать 10.08.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 4,88. Уч.-изд.л. 5,0. Тираж 100 экз. Заказ ____.

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ИСТОРИЯ

Методические указания к практическим занятиям и самостоятельной работе студентов

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 93/94(076.5)
ББК 74/266/3

Рецензент - Доктор исторических наук, профессор А.П. Скорик

Скорик А.П., Бондарев В.А., Бочан С.А., Кравченко В.Я., Панкова-Козочкина Т.В., Шматова Е.С.

История: методические указания к практическим занятиям и самостоятельной работе студентов / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 34 с.

В данном издании представлены методические указания к практическим занятиям и самостоятельной работе по курсу «История». Методические указания предназначены для студентов очной и заочной форм обучения следующих направлений подготовки:

*01.03.04 «Прикладная математика», направленность «Математическое моделирование в экономике и технике»; 02.03.03 «Математическое обеспечение и администрирование информационных систем», квалификация «бакалавр»; направленность «Администрирование информационных систем»; 08.03.01 «Строительство», направленности: «Промышленное и гражданское строительство», «Городское строительство и хозяйство», «Производство строительных материалов, изделий и конструкций», «Экспертиза и управление недвижимостью»; 08.05.01 Специализация «Строительство уникальных зданий и сооружений», «Высотные, большепролетные здания и сооружения»; 08.03.01 «Строительство» по направленностям: «Водоснабжение и водоотведение», «Теплогазоснабжение и вентиляция»; 09.03.01 «Информатика и вычислительная техника»; направленность «Вычислительные машины, комплексы, системы и сети»; «Программные и аппаратные средства встраиваемых вычислительных систем»; 09.03.02 «Информационные системы и технологии»; 09.03.03 «Прикладная информатика»; 09.03.04 «Программная инженерия», направленность «Разработка программно-информационных систем»; 10.03.01 «Информационная безопасность», направленность «Комплексная защита объектов информатизации»; 10.05.02 «Информационная безопасность телекоммуникационных **систем**», направленность «Защита информации в системах связи и управления»11.03.02 «Инфокоммуникационные технологии и системы связи» направленность Инфокоммуникационные технологии в сервисах и услугах связи 11.03.04 «Электроника и наноэлектроника» направленность «Промышленная электроника»; «Нанотехнология в электронике»;12.03.01 «Приборостроение» направленность «Информационно-измерительная техника и технологии»;13.03.01 «Теплоэнергетика и теплотехника», направленность «Тепловые электрические станции», «Энергообеспечение предприятий», «Теплоэнергетические установки электростанций и предприятий»; 13.03.02 «Электроэнергетика и электротехника», направленность «Электромеханика», «Электрический транспорт», «Электрооборудование предприятий, нефтяной и газовой промышленности», «Электрооборудование и электрохозяйство предприятий, организаций и учреждений», «Электроснабжение», «Электропривод и автоматика», «Релейная защита и автоматизация электроэнергетических систем», «Электроэнергетические системы», «Электрические станции», «Энергосбережение и учет в системах ресурсообеспечения»; 13.03.03 «Энергетическое машиностроение», направленность «Автоматизированные гидравлические и пневматические системы и агрегаты»; 15.03.02 «Технологические машины и оборудование», направленности «Машины и аппараты пищевых производств», «Технологическое оборудование химических и нефтехимических производств», «Машины и оборудование нефтяных и газовых промыслов»; «Машины и оборудование нефтяных и газовых промыслов на суше»; 15.03.04 «Автоматизация технологических процессов и производств»; 15.03.05 «Конструкторско-технологическое обеспечение машиностроительных производств», направленность «Технология машиностроения»; 15.03.06 «Мехатроника и роботехника», направленность «Мехатроника»; 18.03.01 «Химическая Технология», направленности «Химическая технология тугоплавких неметаллических и силикатных материалов», «Химическая технология неорганических и органических веществ», «Технология электрохимических производств», «Химическая технология неорганических веществ»; 19.03.02 «Продукты питания из растительного сырья», направленность «Технология бродильных производств и виноделие»; 20.03.01 «Техносферная безопасность» направленности «Инженерная защита окружающей среды»; 21.03.01 «Нефтегазовое дело», направленность «Бурение нефтяных и газовых скважин»; 21.05.01 «Прикладная геодезия», (специалитет), специализация «Прикладная геодезия»; 21.05.02 «Прикладная геология» (специалитет), специализации «Геология нефти и газа», «Геологическая съемка, поиски и разведка месторождений твердых полезных ископаемых», «Поиски и разведка подземных вод и инженерно-геологические изыскания»; 21.05.04 «Горное дело «(специалитет), специализации «Маркшейдерское дело», «Подземная разработка пластовых месторождений», «Открытые горные работы», «Технологическая безопасность и горноспасательное дело», «Горные машины и оборудование»; 22.03.02 «Металлургия», направленности «Металлургия сварочного производства», «Порошковая металлургия»; 23.03.01 «Технология транспортных процессов», направленности «Организация и безопасность движения», «Транспортно-экспедиторская деятельность»; 23.03.02 «Наземные транспортно-технологические комплексы», направленности «Подъемно-транспортные, строительные, дорожные машины и оборудование»; 23.03.03 «Эксплуатация транспортно-технологических машин и комплексов», направленности «Автомобили и автомобильное хозяйство», «Автомобильный сервис», «Сервис транспортных и транспортно-технологических машин и оборудования (Строительные, дорожные и коммунальные машины)»; 23.05.01 «Наземные транспортно-технологические средства», специализации «Автомобили и тракторы», «Подъемно-транспортные, строительные, дорожные средства и оборудование»; 27.03.01 «Стандартизация и метрология», направленность «Стандартизация и сертификация»;27.03.02 «Управление качеством», направленность «Управление качеством производственных процессов»; 27.03.04 «Управление в технических системах», направленность «Управление и информатика в технических системах»; 29.03.04 «Технология художественной обработки материалов»;38.03.01 «Экономика», направленности «Экономика предприятий и организаций», «Мировая экономика», «Математические методы в экономике»; 38.03.02 «Менеджмент», направленность «Производственный менеджмент»; 38.03.03 «Управление персоналом» направленность «Экономика труда»; 38.03.04 «Государственное и муниципальное управление» направленность «Муниципальное управление»; 38.03.05 «Бизнес-информатика», направленность «Математические и инструментальные методы в экономике»; 39.03.03 «Организация работы с молодежью»; 54.03.01 «Дизайн».*

УДК 93/94(076.5)

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов Очная форма обучения	Литература
1	Первые киевские князья: Олег, Игорь, Ольга, Святослав.	2	7 [1-3, 8-10, 12, 15, 20]
2	Русь между востоком и западом: борьба за независимость (XII - начало XIII вв.).	2	7 [1-3, 8-10, 12, 15, 20]
3	Эпоха Ивана IV Грозного.	2	7 [1-3, 8-10, 12, 15, 20]
4	Россия при первых Романовых.	2	7 [1-3, 8-10, 12, 15, 20]
5	Эпоха «дворцовых переворотов» (1725 – 1762 гг.).	2	7 [1-3, 8-10, 12, 15, 20]
6	Культура, наука и просвещение в России XVIII столетия.	2	7 [1-3, 8-10, 12, 15, 20]
7	Общественная мысль и политические движения в России XIX века.	2	7 [1-3, 8-10, 12, 15, 20]
8	СССР накануне и в годы Великой Отечественной войны.	2	7 [1-3, 8-10, 12, 15, 20]
9	Становление рыночной экономики современной России (1992-1998 гг.).	2	7 [1-3, 8-10, 12, 15, 20]

Практическое занятие №1.
ПЕРВЫЕ КИЕВСКИЕ КНЯЗЬЯ: Олег, Игорь, Ольга, Святослав.

1. Князья Олег, Игорь – основатели Древнерусского государства.

Захват Олегом Киева в 882 г. Вокняжение Олега в Киеве. Объединение Киева и Новгорода. Русско-хазарская война. Расширение границ Киевской Руси.

Внешняя политика князя Олега. Походы Олега на Царьград в 907 и в 911 гг. Заключение с Византией договоров. Легенды о гибели князя Олега: данные летописей и иных источников.

Правление князя Игоря. Столкновение с печенегами. Походы князя Игоря на Византию в 941 г. и в 944 г. Заключение русско-греческого договора. Походы князя с дружиной на древлян. Гибель князя Игоря.

2. Регентство и реформы княгини Ольги. Военно-политическая деятельность князя Святослава.

Княгиня Ольга – первая христианка-правительница и первый реформатор на Киевском престоле. Налоговая реформа княгини Ольги. Административные преобразования. Крещение княгини Ольги. Распространение христианства на Руси.

Святослав – князь-воин, «утвердитель военно-политического авторитета Киевской Руси». Война с Хазарским каганатом. Походы князя Святослава на Дунайскую Болгарию.

Заключение договоров с Византией Расширение границ Киевской Руси и укрепление международного авторитета.

3. Эпоха Ярослава Мудрого и Владимира Мономаха: власть, общество, государственность.

Основные направления государственной деятельности Ярослава Мудрого. Социально-экономический строй Киевской Руси. Формирование крупной земельной собственности. Складывание сословного строя. Основные категории свободного и зависимого населения. «Русская правда» и «Правда Ярославичей». Время правления сыновей Ярослава Мудрого и начало распада единого Древнерусского государства. Народные восстания и княжеские междоусобицы. Порядок престолонаследия и Любечский съезд 1097 г. Владимир Мономах и его роль в приостановке процесса политической раздробленности Руси. Рост международного авторитета Руси и отношения с соседями.

Практическое занятие №2.
РУСЬ МЕЖДУ ВОСТОКОМ И ЗАПАДОМ: БОРЬБА ЗА НЕЗАВИСИМОСТЬ (XII - НАЧАЛО XIII ВВ.)

1. Монголо-татарское нашествие на русские земли.

Русь и монголо-татары. Образование империи Чингисхана и монгольское войско. Битва на Калке. Монголо-татарское нашествие на Русь: походы Батыя 1237-1240 гг. Социально-политические изменения в русских землях в период ордынского господства. Политическое и социально-экономическое влияние Золотой Орды на Русь. Социокультурные, политические, экономические и другие последствия ордынского влияния на древнерусское общество. Историки о степени ордынского влияния на характер последующего развития русского общества: не основное и не определяющее (С.Соловьев, В.Ключевский, С.Платонов), глубокое, определяющее (Н.Карамзин, Н.Костомаров), позитивное – Русь стала частью Золотой Орды (евразийцы - П.Савицкий, Н.Трубецкой). Проблема монголо-татарского влияния в современной отечественной и зарубежной историографии.

2. Угроза со стороны Запада.

Расширение влияния Швеции в финно-угорских землях. Невская битва. Политика «Drang nach Osten» («натиска на Восток») европейских крестоносцев. Поддержка и благословение агрессии папской курией. Угроза окатоличивания Руси, захвата исконных русских земель и потери национальной самобытности. Борьба Северо-Западной Руси с агрессией крестоносцев. Ледовое побоище. Русско-Литовское государство и судьба южных и юго-западных русских земель. Взаимоотношения между Литвой и Северо-Восточной Русью.

3. Проблема выбора пути развития между Востоком и Западом.

Варианты развития XIII века: Южная и Юго-Западная Русь (Д.Галицкий - «злее зла честь татарская...»); Северо-Восточная Русь (А.Невский – «за союз с Ордой»). Последствия выбора А. Невского в пользу союза с Ордой. По Вашему мнению, кто был прав в этом историческом споре – Д. Галицкий или А. Невский? С кем должна была идти Русь – с Западом или с Востоком?

Практическое занятие № 3.

ЭПОХА ИВАНА IV ГРОЗНОГО.

1. Исторический портрет Ивана IV Грозного

Детство и особенности формирования личности Ивана IV. Влияние черт личности Ивана IV Грозного на внутри- и внешнеполитическую жизнь страны.

Венчание Ивана IV Грозного на царство. Укрепление власти царя. Периодизация царствования. Два лика Грозного царя. Отечественные историки о личности Ивана IV: разнообразие оценок и суждений. Истоки российского абсолютизма Роль Ивана IV в становлении абсолютизма в российском государстве.

2. Внутренняя и внешняя политика Ивана IV Грозного

Внутренняя политика и реформы Ивана IV до 1565 г., в период существования «Избранной рады»: государственный строй, Судебник 1550 г., военная реформа, «Стоглавый собор». Особенности состава «Избранной рады».

Опричнина, ее социально-политическая суть и последствия. Причины перехода Ивана Грозного к политике опричнины. Опричный террор. Предпосылки складывания самодержавных черт государственной власти и особенности российского абсолютизма.

Внешняя политика Ивана IV Грозного: присоединение и освоение новых земель, выход России к рубежам цивилизаций Востока, российско-крымские отношения, связи Московского государства с народами Кавказа.

Начало Ливонской войны. Разгром Ливонского Ордена, столкновение с Литвой, Польшей и Швецией. Взятие Полоцка. Завершение Ливонской войны. Ям-Запольский мир с Польшей и перемирие со Швецией.

Практическое занятие № 4.
РОССИЯ ПРИ ПЕРВЫХ РОМАНОВЫХ

1. Развитие российского государства при первых Романовых.

Земские соборы как форма согласия социально-политических сил и монарха. Особенности сословно-представительной монархии в Западной Европе и России. Усиление значения приказной системы, падение роли Боярской думы и Земских соборов. Рост централизации власти и формирование абсолютизма на базе феодальной системы. Царь Алексей

Михайлович. Соборное уложение 1649 г.: юридическое закрепление крепостного права и сословных функций государства. Федор Алексеевич: введение подворного обложения прямыми налогами и отмена местничества.

2.Социально-экономическое развитие после Смуты.

Выход из экономического кризиса. Сельское хозяйство, ремесло, появление мануфактур, торговля и начало формирования всероссийского рынка. Территория и население страны. Социальная структура российского общества, укрепление сословного строя. Укрепление феодально-крепостнических отношений.

«Бунташный» XVII век: городские восстания середины XVII в., «Соляной бунт 1648г., «Медный бунт» 1662 г., поход Василия Уса, восстание под предводительством Степана Разина 1670-1671 гг. «Соловецкое сидение».

3. Внешняя политика России при первых Романовых: задачи и направления внешней политики.

Смоленская война 1632 - 1634 гг., «Азовское сидение» донских казаков 1637 - 1642 гг. Процесс вхождения Украины в состав России. Украинское общество на путях выбора союзников. Разработка Иннокентием Гизелем условий вхождения Украины в состав России. Богдан Хмельницкий. Переяславская рада 1654 г. Русско-польская война 1654 - 1667 гг. Андрусовское перемирие.

Русско-шведская война 1656 - 1658 гг. Кардисский мир. Война России с Турцией в 1677-1681гг. Чигиринское походы. Бахчисарайский мир. Строительство Изюмской черты.

Практическое занятие № 5.
Эпоха «дворцовых переворотов» (1725 – 1762 гг.)

1. Власть и общество в период «эпохи дворцовых переворотов».

Причины и общая характеристика «эпохи дворцовых переворотов». Укрепление позиций дворянства. Борьба старой аристократии и новой «петровской» элиты. Значение гвардии в политической жизни Российской империи: гвардейские полки как активная сила при совершении дворцовых переворотов. Династический кризис. «Дело царевича Алексея» как свидетельство поражения приверженцев традиционализма в России. Указ Петра I

о престолонаследии: содержание и оценки. Смерть Петра и нерешенность вопроса о наследнике престола: «отдайте все...»

Россия при Екатерине I (1725 - 1727 гг.). Судьба А. Меншикова. Правление Петра II (1727 – 1730 гг.): всесилие и падение семейства Долгоруких. Заговор «верховников». Анна Иоанновна и «бироновщина»: засилье иностранцев в России. Современные оценки развития России в период правления Анны Иоанновны. Иван VI Антонович и судьба «Брауншвейгского семейства». Елизавета Петровна (1741 – 1761 гг.) и попытка возвращения к «заветам Петра». Петр III и его правление. Манифест 1762 г. «О даровании вольности и свободы всему российскому дворянству». Нарастание недовольства в обществе открытым пренебрежением к обычаям и традициям России. Июньский заговор 1762 года и триумфальное воцарение Екатерины II.

2. Внешняя политика России в первой половине XVIIIвека.

Задачи и основные направления внешней политики: балтийское, польское, черноморское. Русско-польские отношения. Русско-турецкая война 1735 – 1739 гг. Война со Швецией 1741 – 1743 гг. Начало присоединения Казахстана к России. Борьба за колонии и коалиционный раскол Европы. Семилетняя война и значение победы в ней на рост международного авторитета России.

3. Внешняя политика и рост международного влияния России во второй половине XVIII века.

Цели и задачи российского внешнеполитического курса. Основные направления внешней политики. Южная политика Екатерины II и русско-турецкие войны 1768 – 1774, 1787 – 1791 годов, присоединение Крыма и Новороссии. Изменение геополитического положения в Восточной Европе и война со Швецией 1788 – 1790 гг. Образование США и Россия. Речь Посполитая и Российская империя, разделы Польши и территориальные приобретения России. Царизм в борьбе против французской революции, войны с революционной Францией. Россия и Европа: изменения международного положения Российской империи в конце XVIIIв. Рост военно-политического престижа России как великой европейской державы.

Практическое занятие № 6.
КУЛЬТУРА, НАУКА И ПРОСВЕЩЕНИЕ В РОССИИ XVIII СТОЛЕТИЯ

1. Первая половина XVIII века: ломка старых традиций и зарождение новой российской культуры.

Отличительные черты и национальные особенности русской культуры XVIII столетия: усиление светских тенденций, рационалистическое мировоззрение, больший демократизм и открытость в контактах с культурами других народов и стран, взаимодействие русских национальных традиций и достижений европейской культуры. Качественный скачок образования: возникновение и развитие светской школы, формирование системы общего образования. Рождение отечественной науки: создание Петербургской академии наук.

Реформы Петра в области культуры. Изменение традиционного быта и нравов русского народа. Просвещение: возникновение и развитие светской школы, становление системы общего специального образования. Возникновение отечественной науки. Экспедиции и открытия в области науки и техники. Архитектура и градостроительство. Литература. Живопись. Скульптура.

2. Культура России второй половины XVIII века.

Просвещение. Реформы И. Бецкого в системе образования. Открытие Московского университета. Деятельность М.В.Ломоносова. Русская наук, научно-технические открытия и изобретения. Литература. Архитектура. Скульптура. Живопись. Культурные стили. Театр. Выдающиеся представители русской культуры XVIII в.

3. Общественно-политическая мысль в России XVIII столетия.

Политические идеи, инициативы и проекты «петровского» времени. Ф. Салтыков, И. Посошков, П. Шафиров, Ф. Прокопович. С. Яворский как выразитель противников петровских преобразований.

Русское просветительство и развитие общественно-политической мысли. Консервативно-аристократическая идеология, взгляды М. Щербатова. Дворянско-либеральное направление: Н. Панин, А. Воронцов, Е. Дашкова. Представители умеренного просветительства – Н. Новиков, Д. Фонвизин, Д. Голицын. Критика крепостничества, А. Радищев.

Последствия культурного переворота XVIII века. Начало социокультурного раскола общества как результат незавершенности модернизационных процессов в стране.

Отдаление «низов» и «верхов» общества, появление двух типов культур.

Практическое занятие № 7.
ОБЩЕСТВЕННАЯ МЫСЛЬ И ПОЛИТИЧЕСКИЕ ДВИЖЕНИЯ В РОССИИ XIX ВЕКА

1. Идейная борьба и общественные движения первой половины XIX века.

Влияние идей Просвещения и Французской революции на развитие общественно-политической мысли и движений в России в первой половине XIX в. Декабризм: причины возникновения и характер движения, первые политические организации, конституционные проекты. Восстание в Петербурге и на Украине. Причины поражения и значение декабризма.

Охранительное направление, его идеологи и пропагандисты: Н. Карамзин, С. Уваров и «теория официальной народности», Ф. Булгарин, Н. Греч, М. Погодин и др. Проблема соотношения в «охранительстве» реакционного и национально-патриотического начал.

Либеральное направление: идейное наследие П. Чаадаева, его концепция культурно-исторического развития России вне европейской цивилизации; становление идеологии русского либерализма, славянофильство (К. и И. Аксаковы, И. и П. Киреевские, А. Хомяков, Ю. Самарин) и западничество (Т. Грановский, К. Кавелин, С. Соловьев, В. Боткин, П. Анненков и др.).

Радикальное направление: кружок братьев Критских; «Сунгуровское общество»; кружок Станкевича; В. Белинский и его письмо к Н. Гоголю; «Петрашевцы»; А. Герцен и Н. Огарев.

2. Общественно-политическая жизнь России во второй половине XIX в.

Причины подъема общественно-политических движений и течений. Общественное движение как фактор политической жизни России. Оформление трех направлений в общественном движении.

Консервативное направление. Социальная база и идеологи консерватизма (К. Победоносцев, Д. Толстой, М. Катков). Неославянофилы, почвенники, религиозно-философское течение.

Понимание консерваторами своеобразия исторического пути развития России.

Либеральное направление. Социальная основа российского либерализма, его течения. Идеологи русского либерализма второй половины века: К. Кавелин, Б. Чичерин, Д. Шаховский. Борьба между либералами и консерваторами за влияние на правительственную политику. Земское движение.

Радикальное направление. Социальная база, течения и этапы развития радикализма второй половины XIX в. Русский аграрный социализм. «Шестидесятники», «Земля и воля» и тайные организации, С.Г. Нечаев и «нечаевщина». Основные направления народничества: бунтарское (М. Бакунин), пропагандистское (П. Лавров), заговорщическое (П. Ткачев). «Хождение в народ». «Черный передел», «Народная воля». Марксизм в России: марксистские кружки и рабочие организации, группа «Освобождение труда», Г.В. Плеханов. «Союз борьбы за освобождение рабочего класса». В.И. Ульянов (Ленин). «Легальный марксизм».

Практическое занятие № 8.
СССР НАКАНУНЕ И В ГОДЫ ВЕЛИКОЙ ОТЕЧЕСТВЕННОЙ ВОЙНЫ.

1. Мир и СССР накануне и в первые годы Второй мировой войны.

Коллизии Версальской системы. Возникновение двух очагов войны: в Европе (Германия) и на Дальнем Востоке (Япония). Укрепление международного положения Советского Союза: договоры о ненападении с Польшей, Финляндией, Латвией, Эстонией и Францией, нормализация советско-американских отношений. Вступление в СССР в Лигу Наций. Участие в Женевской международной конференции. Проблема коллективной безопасности в мире в середине 1930-х гг. Переговоры о Восточном пакте, их результаты. Переговоры о Тихоокеанском пакте. Договоры о взаимной помощи с Францией, Чехословакией, Монголией, о ненападении с Китаем. Политика «умиротворения агрессора»: причины ее возникновения, суть и итоги. Мюнхенский сговор.

Внешняя политика Советского Союза во второй половине 30-х гг. Отражение японской агрессии, присоединение Прибалтики, финская война. Военно-политические блоки и их позиции. Начало

фашистской агрессии. Оккупация Германией ряда европейских стран. Планы Гитлера по поводу захвата Великобритании. Фашистский режим в Европе. Национально-освободительное, антифашистское движение Сопротивления.

2. Начальный период войны (июнь 1941 – ноябрь 1943 гг.).

Замыслы нацистского руководства: план «Барбаросса». Планы советского командования. Причины неудач Красной Армии в начальный период войны. Оборонительные бои за Киев и Смоленск; блокада Ленинграда; причины неудач Красной Армии в начальный период; оборонительный этап Московской битвы, контрнаступление Красной Армии под Москвой, военно-политические итоги и международное значение победы под Москвой; военные действия весной-летом 1942 г.; Сталинский приказ № 227 «Ни шагу назад!». Перестройка экономики на военный лад; военно-промышленный потенциал страны в 1941 г.; обеспечение продовольствием. Формирование антигитлеровской коалиции.

3. Период коренного перелома (ноябрь 1942 – декабрь 1943 гг.).

Разгром фашистских войск под Сталинградом, международное значение победы в Сталинградской битве; освобождение Северного Кавказа и Дона; совершенствование организации Красной Армии. Курская битва; военные действия Красной Армии во второй половине 1943 г. Советский тыл, особенности развития военной индустрии в данный период.

4. Завершающий период войны (январь 1944 – сентябрь 1945 гг.)

Наступательные операции советских войск в 1944 г.; освобождение стран восточной Европы. Заключительные операции Красной Армии; битва за Берлин. Победа в Великой Отечественной войне – триумф и трагедия советского народа. Вклад союзников СССР в общую победу над фашизмом. Цена победы. Потери СССР в войне. Менталитет и духовный облик фронтового поколения. Какие тенденции наблюдались в общественном сознании населения СССР на завершающем этапе и после Великой Отечественной войны? Участие СССР в войне против Японии. Итоги Великой Отечественной и Второй мировой войны, всемирно-историческое значение победы советского народа над фашизмом.

Практическое занятие № 9
СТАНОВЛЕНИЕ РЫНОЧНОЙ ЭКОНОМИКИ СОВРЕМЕННОЙ РОССИИ (1991-2017 гг.).

1. Социально-экономическое развитие России в 1992-1998 гг.

Этапы экономического развития современной России и их характеристика. «Шоковая терапия». Разрушение единого государственно-хозяйственного комплекса. Приватизация и акционирование в промышленности и сельском хозяйстве. Формирование рыночных отношений, финансовой и кредитно-банковской систем. Падение производства и престижа производительного труда. Усиление сырьевой ориентации экономического развития. Финансовые «пирамиды» и дефолт августа 1998 г. Социальные последствия экономических реформ. Рост цен и безработица. Задержка выплат заработной платы и социальных пособий. Социальное расслоение российского общества. Преобладание на потребительском рынке импортных товаров, снижение образовательного уровня, превышение смертности над рождаемостью и сокращение населения страны.

2. Российская экономика после кризиса 1998 г.

Поиски путей выхода из экономического кризиса. Смена правительств (весна-лето 1999 г.). Социально-экономический курс В.В. Путина. Укрепление финансовой и банковской систем. Изменение внутренней и международной экономической ситуации. Повышение цен на нефть. Налоговая и инвестиционная реформы. Рост отечественного производства. Отказ от внешних финансовых заимствований. Поворот к социально ориентированной экономике. Национальные проекты. Пенсионная реформа. «Монетизация льгот». Бюджетная политика. Противоречия современного экономического развития России. Мировой экономический кризис 2008-2009 гг. и его влияние на российскую экономику. Подписание договора между Российской Федерацией и Республикой Крым о принятии Республики Крым в состав России

3. Экономическое развитие Российского государства и общества с 2009-2017гг.

Подъем иностранных инвестиций в Россию. Увеличение числа субъектов малого и среднего предпринимательства. Создание инновационного центра «Сколково». Разработка федеральных целевых программ. Принятие Стратегии социально-

экономического развития Дальнего Востока и Байкальского региона. Развитие Байкало-Амурской магистрали (БАМА). Проведение Зимних Олимпийских игр 2014г.

2. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

№	Наименование тем занятий	Кол-во часов Очная форма обучения	Литература
1	Тема № 18 Восточные славяне в VI - IX вв.	6 часов	7 [1-3, 8-10, 12, 15, 20]
2	Тема № 19 Русские земли и княжества в период феодальной раздробленности	6 часов	7 [1-3, 8-10, 12, 15, 20]
3	Тема № 20 Российское государство и донское казачество в XVI - XVII вв.	6 часов	7 [1-3, 8-10, 12, 15, 20]
4	Тема № 21 Внешняя политика России в XVIII XIX вв.	6 часов	7 [1-3, 8-10, 12, 15, 20]
5	Тема № 22 Становление парламентаризма в России	6 часов	7 [1-3, 8-10, 12, 15, 20]
6	Тема №23 Научно-техническая революция и ее влияние на ход общественного развития	6 часов	7 [1-3, 8-10, 12, 15, 20]
7	Тема №24 Духовная жизнь и культура советского периода	6 часов	7 [1-3, 8-10, 12, 15, 20]
8	Тема №25 СССР на международной арене (1960-е –1980-е гг.)	5 часов	7 [1-3, 8-10, 12, 15, 20]
9	Тема №26 История современной России (1991-2010 гг.)	5,2 часа	7 [1-3, 8-10, 12, 15, 20]

Задания для самостоятельного изучения и конспектирования

Тема № 1.
ВОСТОЧНЫЕ СЛАВЯНЕ В VI - IX ВВ.

1. Проблемы этногенеза восточных славян.

Античный мир и древнейшие народы и государства на территории России и сопредельных регионов: иранцы юга и греческие колонии в Северном Причерноморье, «черняховцы» и готы, гуннское нашествие и наследники гуннов, Хазарский каганат. Влияние Византии на историческое развитие Восточной Европы. Происхождение славян: поиски прародины.

Миграционная и автохтонная теории. Этнокультурные и социально-политические процессы становления восточнославянского общества. Территория расселения восточнославянских племенных объединений.

2. Общественный строй восточных славян.

Эволюция общины как основы общественной жизни. Социальная структура восточнославянского общества. Причины появления княжеской власти и ее функции. Материальная культура. Верования, нравы и обычаи восточных славян. Отношения восточных славян с соседями: славянская ассимиляция или этнокультурный синтез. Изменения в недрах восточнославянского общества на рубеже VIII - IX вв., формирование условий, необходимых для возникновения государства: появление соседской общины, социальная и имущественная дифференциация, преодоление племенной замкнутости и формирование территориально-политических объединений («союзов племен»), а затем, на их основе, еще более крупных образований – «суперсоюзов».

Тема № 2.
РУССКИЕ ЗЕМЛИ И КНЯЖЕСТВА В ПЕРИОД ФЕОДАЛЬНОЙ РАЗДРОБЛЕННОСТИ

1. Эволюция древнерусской государственности в XI - XП вв.

Начало удельного периода. Причины раздробленности: упадок политической власти Киева, рост вотчинного землевладения, возникновение удельной системы, экономическая

самостоятельность отдельных княжеств и земель вследствие господства натурального хозяйства и др. Превращение Руси в своеобразную федерацию княжеств во главе с Великим князем Киевским. Любечский съезд князей 1097 г. Дальнейшее снижение влияния Киева и утрата им политического лидерства на Руси.

2. Формирование политических центров и становление различных социокультурных моделей развития древнерусского общества и государства.

Новгород Великий, Владимиро-Суздальское княжество и Галицко-Волынское княжество. Социально-политическая структура русских земель периода политической раздробленности. Особенности развития хозяйства, политических институтов, культуры русских земель удельного периода. Современные дискуссии о значении и роли политической раздробленности в развитии древнерусского общества и государства. Последствия раздробленности.

Тема № 3.
РОССИЙСКОЕ ГОСУДАРСТВО И ДОНСКОЕ КАЗАЧЕСТВО В XVI - XVII ВВ.

1. Социально-политические процессы донского казачества.

Социально-политические и этнокультурные процессы в Подонье и Приазовье. Версии возникновения донского казачества. Особенности уклада жизни и быта донских казаков. Становление «Главного Донского войска». Политическая власть и управление. Грамота Ивана Грозного донским казакам и Договор 1570 г. Взаимоотношения донского казачества с Московским и другими соседними государствами. Ермак и присоединение Сибири.

2. Развитие социально-политических отношений.

Социальное расслоение донских казаков. Система управления и уклад жизни. Хозяйственное и экономическое развитие Дона. Участие донского казачества в событиях Смутного времени в России. Роль донских казаков в защите юго-восточных границ Российского государства. Азовское осадное сидение. Участие донского казачества в народном движении под предводительством Степана Разина. Азовские походы Петра I и донские казаки.

Тема 4.
ВНЕШНЯЯ ПОЛИТИКА РОССИИ XVIII-XIX вв.

1. Внешняя политика России в первой четверти XIX в. Отечественная война 1812 г.

Основные направления внешней политики. Борьба против французской экспансии, участие в войнах в составе антинаполеоновских коалициях. Войны с Турцией и Ираном. Территориальные приобретения: добровольное вхождение Грузии в Россию, завоевание Северного Азербайджана, большей части Закавказья, Дагестана и западного побережья Каспийского моря; присоединение Финляндии.

Отечественная война 1812 года: причины и характер войны, соотношение сил, планы сторон. Начальный период войны. Бородинское сражение. От Москвы до Малоярославца. Изгнание Наполеона из России. Значение и последствия войны. Заграничные походы русской армии в 1813 – 1814 гг. Послевоенная европейская политика России: Венский конгресс, Священный союз. Россия и революции в Европе; усиление реакционной линии в российской внешней политике.

2. Вторая четверть – конец XIX века: обострение международных противоречий

Изменение международного положения и основные направления внешней политики России. Кавказское направление: русско-персидская война 1826 – 1828 гг. Политика России в «Восточном вопросе»: кризис Османской империи; Россия и Греческое восстание 1821 г.; русско-турецкая война 1828 – 1829 гг. Россия и народы Северного Кавказа. Кавказская война.

Идеи панславянизма во внешней политики Российской империи. Крымская война 1853 – 1856 годов: причины войны и соотношение сторон; ход военных действий; поражение России в войне; Парижский мир. Оценки и последствия Крымской войны. Продажа Аляски и Алеутских островов США. Русско-японский договор 18 75г. о передаче южного Сахалина России. Укрепление военно-политического союза с Францией. Договоры с Турцией и Персией. Гаагская мирная конфедерация 1899г.

Тема № 5.
СТАНОВЛЕНИЕ ПАРЛАМЕНТАРИЗМА В РОССИИ.

1. Общественное движение в конце XIX – начале XX вв. Появление первых политических партий.

Усложнение общественной обстановки в конце XIX – начале XX вв., обострение и переплетение социальных конфликтов. Общественное недовольство режимом Николая II. Возникновение политических партий, особенности их формирования в России. Манифест Николая II «Об усовершенствовании государственного порядка». Становление российской многопартийности: социалистические партии, возникновение партий кадетов и октябристов, черносотенных организаций. Деятельность I и II Государственной думы, преобразования в политической системе. «Третьеиюньская» политическая система и итоги революции.

2. Деятельность III и IV Государственной Думы.

Консервативное направление в Думе («черносотенцы» и монархисты – А. Дубровин, Н. Марков), консервативно-реформаторское направление («Союз 17 октября» - А.И. Гучков, М.В. Родзянко), либеральное направление (кадеты – С.А. Муромцев, П.Н. Милюков, В.И. Вернадский), лево-радикальное крыло (эсеры – М.Р. Гоц, Г.А. Гершуни, В.М. Чернов; меньшевики – Г.В. Плеханов, Ю.О. Мартов; большевики – В.И. Ленин, Л.Д. Троцкий). Формирование правительственной оппозиции - «Прогрессивный блок». Националисты и трудовики в Государственной Думе. Депутаты Государственной думы в февральской революции 1917 г.

Тема № 6
НАУЧНО-ТЕХНИЧЕСКАЯ РЕВОЛЮЦИЯ И ЕЕ ВЛИЯНИЕ НА ХОД ОБЩЕСТВЕННОГО РАЗВИТИЯ.

1. Научно-техническая революция как глобальное явление второй половины XX века.

Революция в технике как проявление развития научно-технического прогресса. Глобальный характер научно-технической революции во второй половине XX века. Научные открытия, определившие развитие техники: открытие практического применения ядерной энергии, создание компьютерной техники, исследование космического пространства, развитие химической

промышленности и массовое производство синтетических материалов. Интеграция науки и производства в рамках отраслевых научно-производственных комплексов.

2. Основные направления и особенности развития НТР в Советском Союзе.

Теоретическая разработка и применение ядерного вооружения и мировое сообщество. Процесс развития «холодной войны» и борьба за мир. Деятельность А.Д. Сахарова. Развитие атомной энергии.

Исследование космоса. Успехи советских ученых: запуски первых искусственных спутников Земли и первого пилотируемого космического корабля 12 апреля 1961 г. Ю.А. Гагарин. Трудности, сложности и трагизм в освоении космического пространства. Научное соперничество и сотрудничество СССР и США в освоении космоса.

Профессиональная и социальная роль ученых и инженеров в создании научно-технического потенциала страны. Влияние научно-технической революции на развитие современной экономики. Переход мировой экономики к новому технологическому этапу развития НТР – массовому использованию высокоточных технологий во всех отраслях экономики. Причины отставания СССР в массовом освоении новых технологий в 70-х - первой половине 80-х гг. XX века. Современное состояние и развитие науки и техники.

Тема №7
ДУХОВНАЯ ЖИЗНЬ И КУЛЬТУРА СОВЕТСКОГО ПЕРИОДА.

1. Культура в 1945 – 1953 гг.

Диктат власти над духовной и культурной жизнью общества. Постановление ЦК КПСС «О журналах «Звезда» и Ленинград» и послевоенная литература. Идеологические нападки на кинематографистов, драматургов, музыкантов. Особенности развития советской послевоенной науки. Народное образование: введение обязательной семилетней школы, распространение вечернего и заочного профессионального обучения. Успехи и достижения советских деятелей культуры этого периода.

2. «Оттепель» в духовно-культурной сфере.

Перемены в духовной и культурной жизни общества после XX съезда КПСС. Ослабление идеологического давления на литературу и искусство. Поколение «шестидесятников». Особенности развития культуры: противоречивость, партийный контроль над деятельностью художественной интеллигенции и низкий художественный вкус власти. Гонения на Б.Л. Пастернака, Э.И. Неизвестного и др. «Дело молодых историков». Деятели культуры и их достижения. Театр «Современник». Журналы «Юность», «Молодая гвардия», «Иностранная литература» и др. Развитие науки: достижения научно-технической революции, наука и военно-промышленный комплекс, наука и освоение космоса, развитие атомной энергетики, расширение сети НИИ, создание Сибирского филиала АН СССР, советские ученые и их достижения, лауреаты Нобелевской премии. Реформы системы образования: укрепление связи школы и производства, введение одиннадцатилетнего и обязательного восьмилетнего обучения в школе, отмена платы за обучение, школы рабочей молодежи (ШРМ) – техникум – трехгодичная средняя школа с обязательным производственным обучением.

3. Советская культура «эпохи развитого социализма».

Культурная жизнь в 1953 – 1984 гг. Внутренняя противоречивость и неоднозначность культурных процессов. Идеологически-цензурное давление на культуру. Официальная и неофициальная культура. Народное образование и наука: введение всеобщего среднего образования, расширение сети вузов, успехи советской науки, признание трудов ученых, наука и народное хозяйство, научно-производственные объединения. Литература и искусство: дуализм развития, журналы «Новый мир», «Юность» и др., выдающиеся деятели и их произведения, запрещенные произведения и «самиздат». Культура периода «перестройки». Гласность. Освобождение духовной и культурной сфер от партийно-государственного контроля. Неоднозначность культурных процессов в обществе. Публикация ранее запрещенных художественных произведений. Нормализация отношений между государством, обществом и церковью. Внедрение элементов западной массовой культуры. Развитие художественной культуры и искусства.

Тема № 8.
СССР НА МЕЖДУНАРОДНОЙ АРЕНЕ
(1960 – 1980-е гг.)

1. Отношения СССР со странами Запада и третьего мира.

Направления внешней политики и международной деятельности СССР. «Хрущевское наследие» в сфере международных отношений: последствия Карибского кризиса, обострение отношений между Востоком и Западом, раскол социалистического лагеря, рост влияния и колоссальных затрат СССР в странах третьего мира. Идеологизация международных отношений. Достижение военно-стратегического паритета между СССР и США. Международная реакция на ввод войск Варшавского договора в Чехословакию. Программа мира XXIV и XXV съездов КПСС. Политика «разрядки» и активизация Советского Союза на международной арене. ООН и советский проект договора о нераспространении ядерного оружия. Визит де Голля в СССР. Нормализация отношений с ФРГ. Расширение экономических связей с Западом. Контакты и договоренности между СССР и США. Разрядка международной напряженности. Совещание по безопасности и сотрудничеству в Европе. Хельсинский процесс и Декларация принципов межгосударственных отношений. Обострение международных отношений в конце 1970-х – первой половине 1980-х годов: причины, региональные конфликты, ввод советских войск в Афганистан, падение международного авторитета СССР, виток гонки вооружений, срыв договоренностей и соглашений, конфронтация между Западом и Востоком. Поддержка национально-освободительных движений и режимов.

2. СССР и социалистические страны.

Сотрудничество со странами социалистического содружества: военное, политическое и экономическое. Устранение угрозы распада социалистического лагеря. Отношения КПСС и зарубежных коммунистических и рабочих партий. Международные совещания коммунистических и рабочих партий в Москве 1965 и 1969 гг. «Доктрина Брежнева» об «ограниченном суверенитете социалистических стран». Ввод войск Варшавского договора в Чехословакию и подавление антисоциалистических выступлений. Помощь Вьетнаму в борьбе с американской агрессией. СЭВ и социалистическое содружество. Отношения с КНР, Албанией, Румынией.

Тема 9.
ИСТОРИЯ СОВРЕМЕННОЙ РОССИИ (1991-2017 гг.)

1. Обретение Российского государственного суверенитета. Конституция РФ 1993 г. Политический курс Б.Н. Ельцина.

Общественное сознание и ожидание демократических реформ. Расстановка политических сил Российской Федерации после распада СССР. Конституционные противоречия государственного устройства РФ в 1992-1993 гг. Запаздывание политических реформ и нарастание кризиса власти. События сентября-октября 1993 г. Принятие Конституции Российской Федерации, выборы в Федеральное Собрание 1993 г. Становление президентской республики (1992-1999 гг.). Б.Н. Ельцин – первый российский Президент (1991-1999 гг.).

2. Экономическая и политическая стабилизация при В.В. Путине.

Расстановка политических сил после экономического кризиса августа 1998 г. Выборная кампания конца 1999 – начала 2000 гг. Социально-экономический курс В.В. Путина. Укрепление финансовой и банковской систем. Повышение цен на нефть. Налоговая и инвестиционная реформы. Рост отечественного производства. Отказ от внешних финансовых заимствований. Национальные проекты. Пенсионная реформа. «Монетизация льгот».

Политика государственного строительства В.В. Путина. Изменение статуса Совета Федерации. Роль Государственного Совета в политической жизни страны. Введение федеральных округов и института представителей президента РФ в федеральных округах. Общественная палата – новый институт гражданского общества. Приведение в соответствие законодательства субъектов Федерации с Конституцией РФ. Судебная реформа. Современное состояние партийного строительства. Противоречия современной политической системы.

3. Современный этап реформирования российского общества.

Начало президентского правления Д.А. Медведева. «Политический тандем» Президента и Председателя Правительства. Бюджетная политика. Мировой экономический кризис 2008-2009 гг. и его влияние на российскую экономику. Изменения в системе федеральных округов. Реформы в образовании. Реализация национальных проектов. Принятие закона «Об изменении срока полномочий Президента РФ и

Государственной Думы». Обновление губернаторского корпуса. Возрождение казачества.

События 2008 г. в Южной Осетии. Введение новой военной доктрины России. Стратегии национальной безопасности. Укрепление собственной обороноспособности, в том числе и в рамках Договора о Коллективной Безопасности (ДКБ). Создание Коллективных сил оперативного реагирования. Реформирование российской армии.

Экзаменационные вопросы по курсу

1. Восточные славяне в VI - IX вв.: происхождение, хозяйственные занятия, социальные отношения, культура и быт, религиозные воззрения.
2. Образование Древнерусского государства. Научная критика норманнской теории. Деятельность первых русских князей в конце IX - начале XII вв.
3. Культура Древней Руси IX - XII вв. Значение принятия Русью православия в формировании культуры и ментальности русского народа.
4. Феодальная раздробленность на Руси: причины и последствия. Политические центры периода раздробленности (Новгородская земля, Владимиро-Суздальское и Галицко-Волынское княжества).
5. Русские земли в XIII веке: экспансия с Востока и Запада. Монголо-татарское влияние на судьбу страны.
6. Объединение северо-восточных земель и княжеств вокруг Москвы (конец XIII - первая половина XVI вв.). Деятельность первых московских князей. Государи Всея Руси - Иван III и Василий III.
7. Внутренняя и внешняя политика Ивана IV (1533-1584 гг.). Оценка правления Ивана IV в российской историографии.
8. «Смутное время» в России (конец XVI - начало XVII): причины, основные события, итоги.
9. Московское государство при первых Романовых (1613 - 1682 гг.). Экономические и социально-политические преобразования и формирование абсолютизма.
10. Культура, быт и духовная жизнь русского общества в XIV - XVII вв. Деятели российской культуры (по выбору студента).
11. Россия при Петре I (1682-1725 гг.): петровская концепция модернизации страны. Удалось ли Петру I европеизировать

российское общество.

12. Внешняя политика России в первой половине XVIII в.

13. Россия в эпоху «дворцовых переворотов» (1725 - 1762 гг.): борьба политических группировок за власть.

14. Россия во второй половине XVIII в. «Просвещенный абсолютизм» Екатерины II. Деятели екатерининской эпохи (по выбору студента).

15. Донские казаки в народных движениях XVII - XVIII вв. И.И. Болотников, С.Т. Разин, К.А. Булавин, Е.И. Пугачев. Особенности крестьянских войн в России.

16. Внешняя политика России во второй половине XVIII века и расширение границ империи до «естественных пределов».

17. Александра I и попытки либеральных реформ. Его отношение к отмене крепостного права и принятию Конституции.

18. Россия во второй четверти XIX в. Николай I и его «охранительная» политика. Основные течения общественной жизни: декабристы, западники и славянофилы.

19. Внешняя политика России первой половины XIX в.: от наполеоновских войн к Крымской войне (1953-1856 гг.).

20. Отмена крепостного права и Великие реформы 1860-х - 1870-х гг. Александр II. Контрреформы Александра III.

21. Общественная жизнь России второй половины XIX в.: основные течения и представители. Победа радикального направления.

22. Внешняя политика России во второй половине XIX в. Кавказская война. Присоединение Средней Азии.

23. Культура России XVIII - XIX вв.: тенденции, течения, представители.

24. Особенности социально-экономической модернизация России на рубеже XIX-XX вв. Реформы С.Ю. Витте и П.А. Столыпина.

25. Первая революция в России 1905 - 1907 гг.: причины, этапы, последствия. Возникновение парламентаризма в России и формирование массовых политических партий.

26. «Серебряный век» русской культуры (конец XIX - начало XX вв.): течения и представители.

27. Россия в системе международных отношений в начале XX в.: от русско-японской к Первой мировой войне.

28. Россия в 1917 г.: от Февраля к Октябрю. Выбор исторического пути развития.

29. Гражданская война в России (1917 - 1922 гг.). Политика «военного коммунизма». Казачество Дона в годы Гражданской войны.

30. Экономический и политический кризис в начале 20-х годов XX

века и переход к новой экономической политики. Смысл и содержание НЭПа.

31.Форсированное строительство социализма в СССР в конце 1920-х-1930-е годы: индустриализация, коллективизация, культурная революция. Формирование сталинизма: идеологические компании и политические репрессии.

32.Международные отношения и советская внешняя политика в 1920-е - 1930-е гг. Начало Второй мировой войны.

33.Советская культура 1920-х - 1930-х г.: основные тенденции и представители.

34.Великая Отечественная война (1941 - 1945 гг.): от первых трагических дней к коренному перелому (июнь 1941- лето 1943 гг.)

35.Завершающий этап Великой Отечественной войны: (лето 1943 – май 1945 гг.). Героизм народа. Величие и цена Победы.

36.Социально-экономическое и общественно-политическое развитие СССР в 1945-1953 гг.

37.Реформаторская деятельность Н.С. Хрущева. «Хрущевская оттепель и ее проявления». События июня 1962 г. в Новочеркасске.

38.СССР в середине 1960-х - середине 1980-х гг. Брежневская стабильность и реформы А.Н. Косыгина, причины их свертывания. Ю.В. Андропов. К.У. Черненко.

39.М.С. Горбачев: политика «ускорения» и «перестройки», ее крах. Распад СССР (1985 - 1991 гг.).

40.Международные отношения и внешняя политика СССР в условиях «холодной войны» (1945 - 1991 гг.).

41.Научно-техническая революция и ее влияние на ход общественного развития Советского Союза.

42.Советская культура послевоенного периода (1945 - 1991 гг.): основные направления и представители.

43.Рыночные реформы и социально-экономическое развитие России в 1991 - 1999 гг.: цели и результаты, последствия преобразований. Цена «ельцинского лихолетья».

44. Оформление российской государственности, поиски новых моделей федерализма в 1990-х гг. Чеченский кризис. Первый Президент России Б.Н. Ельцин.

45. Экономическое и общественно-политическое развитие России в начале XXI в. В.В. Путин. Д.А. Медведев. Ростовская область как субъект Российской Федерации.

46. Российская Федерация (конец XX - начало XXI в): внешнеполитические приоритеты.

Контактная внеаудиторная работа

СРС: групповые консультаций в течение семестра – _0,4_ ч

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Основная учебная литература

1. Бочан С.А. История: учеб. пособие для студ. техн. вузов / С.А Бочан, Е.С. Шматова; ЮРГПУ (НПИ) им. М.И. Платова.- Новочеркасск: Изд-во ЮРГПУ (НПИ), 2013.-259с.

2. Скорик. А.П. Исторические зори России: [Электронный ресурс] история Отечества с древнейших времен до конца XI века: учеб. пособие для техн. вузов/ А.П. Скорик, В.А. Бондарев; ЮРГПУ (НПИ) им. М.И. Платова.- Новочеркасск: Изд-во ЮРГПУ (НПИ), 2013. - 291с. Режим доступа: http://lib.npi-tu.ru/infresource/el_katalog

2. Дополнительная учебная литература

Учебные издания

3. Бесов А.Г. Отечественная история: учебное пособие. [Электронный ресурс] - Юнити-Дана, 2012. - 383 с. Режим доступа: http://www.knigafund.ru

4. Валиуллин К.Б., Зарипова Р.К. Отечественная история. [Электронный ресурс] - ИНТУИТ 2012. - 218 с. - Режим доступа: http://www.knigafund.ru

5. Кузнецов И.Н. Отечественная история: [Электронный ресурс] - Учебник. - Дашков и К. 2014. - 815 с. Режим доступа: http://www.knigafund.ru

6. История России: учебник/под ред. Г.Б. Поляка. [Электронный ресурс] - Юнити-Дана 2012. - 686 с. Режим доступа: http://www.knigafund.ru

7. Ольштынский Л.И. Курс истории для бакалавров. Общие закономерности и особенности развития России в мировом историческом процессе. Уроки истории: учебное пособие. [Электронный ресурс] - Логос 2012. - 406 с. Режим доступа: http://www.knigafund.ru

3. Периодические издания

8. Вестник Московского университета Серия 8: История – Доступ eLIBRARY.RU

9. Известия ВУЗ. Северо-Кавказский регион Серия: Общественные науки – Доступ eLIBRARY.RU

10. Вопросы истории [Текст]: журнал.- до 2015г.

11. Российская история [Электр. ресурс]: журнал – Доступ eLIBRARY.RU

4. Печатные и рукописные методические указания, рекомендации, инструкции к практическим занятиям:

12. Примерная рабочая программа обязательной базовой дисциплины «История»: учебно-методическое пособие для студентов всех специальностей / Юж.-Рос. гос. политехн. ун-т.(НПИ) им. М.И. Платова – Новочеркасск: Лик, 2014. – 64 с.

13. Хронотоп российской истории: люди, события, факты. Учеб.пособ. для студентов всех форм обучения // Юж.-Рос. гос. политехн. ун-т. .(НПИ) им. М.И. Платова – Новочеркасск: Лик, 2014. – 60с.

14.Бондарев В.А. Левакин А.С. Крымская война (1853-1856гг.)Методическое пособие по базовому учебному курсу «История» / Юж.-Рос. гос. политехн. ун-т.(НПИ) им. М.И. Платова – Новочеркасск: Лик, 2014. – 40 с.

15.Бондарев В.А., Бочан С.А., Левакин А.С. История правящей династии Романовых: лица, факты, мнения. Учебно-методическое пособие для студентов всех специальностей по курсу «История»/ Юж.-Рос. гос. техн. ун-т (НПИ) им. М.И. Платова - Новочеркасск: 2015.- 23с.

16.Бондарев В.А., Бочан С.А., Левакин А.С. История просветительского общественного движения России X – начала XXI вв.: Учебно-методическое пособие для студентов всех специальностей по курсу «История»/ Юж.-Рос. гос. техн. ун-т (НПИ) им. М.И. Платова – Новочеркасск: 2015. – 23 с.

17.Панкова-Козочкина Т.В., Скорик А.П., Шматова Е.С. Удельная Русь: локальные варианты исторического развития: учебно-методическое пособие для студентов всех специальностей по курсу «История» / Юж.-Рос. гос. техн. ун-т (НПИ) им. М.И. Платова – Новочеркасск: 2015. – 38 с.

18.Левакин А.С. Первая чеченская война (1994-1996гг.): причины и уроки/ Учебно-методическое пособие по базовому курсу «История» - Новочеркасск ЮРГТУ, 2014. –31 с.

5. Интернет – ресурсы

19. Справочно-правовая система «Консультант Плюс» - Режим доступа: http: //www.consuitant.ru, свободный. (304 гл.к., 222 гл.к.)

20. Официальный сайт Президента РФ // Президент.рф
21. Официальный сайт Совета Федерации // council.gov.ru
22. Официальный сайт Государственной Думы //duma/gov.ru
23. Официальный сайт Правительства Ростовской области //donland.ru
24. Научная электронная библиотека eLIBRARY.RU - российский информационный портал в области науки, технологии, медицины и образования.
25. «Мир истории» - Режим доступа: http: //www.historia.ru свободный. (304 гл.к., 222 гл.к.)

Учебно-методическое издание

Бондарев Виталий Александрович
Бочан Светлана Александровна
Кравченко Владимир Яковлевич
Панкова-Козочкина Татьяна Викторовна
Скорик Александр Павлович
Шматова Елена Сергеевна

История

Методические указания
к практическим занятиям
и самостоятельной работе студентов

Редактор *Н.А.Юшко*
Подписано в печать 12.10.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,39. Уч.-изд.л. 1,5. Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический
университет (НПИ) имени М. И. Платова

Д.В. Гринченков, Д.Н. Кущий

ЛОГИКА ВЫСКАЗЫВАНИЙ И БУЛЕВЫ АЛГЕБРЫ

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 517.8 (075.8)
ББК 22.17
Г35

Рецензенты:

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

кандидат технических наук, профессор, профессор кафедры
«Программное обеспечение вычислительной техники»
Иванченко Александр Николаевич

Гринченков Д.В., Кущий Д.Н.
Г35 Логика высказываний и булевы алгебры: учебное пособие / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова.-Новочеркасск: ЮРГПУ (НПИ), 2016. – 82 с.

В пособие описаны: основные положение теории множеств; базовые вопросы булевой алгебры, включающие описание логических функций, специальные разложения логических функций, способы минимизации булевых функций; общие сведения о формальных и аксиоматических системах; основы логики высказываний и логики предикатов.

УДК 517.8 (075.8)

1. ТЕОРИЯ МНОЖЕСТВ

1.1. Основные понятия теории множеств

1.1.1. Множества, способы задания множеств

Под ***множеством*** понимают объединение в одно целое объектов, хорошо различимых нашей интуицией или нашей мыслью. Другими словами, ***множество*** — это совокупность объектов любой природы, рассматриваемая как единое целое. Обычно множества обозначают прописными латинскими буквами $A, B, M, \ldots$.

Пример. $N = \{1, 2, 3, \ldots\}$ – множество натуральных чисел.

Объекты, образующие множество, называются ***элементами*** множества (обозначаются маленькими буквами). Если элемент a входит во множество A, то это обозначается так: $a \in A$. Запись $a \notin A$ означает, что элемент a не принадлежит множеству A. Множество, содержащее конечное число элементов, называется ***конечным*** (в противном случае – ***бесконечным***). Если множество A конечно, то число его элементов называется ***мощностью*** множества и обозначается $|A|$. Если множество не содержит ни одного элемента, то оно называется ***пустым*** множеством и обозначается символом $\varnothing$. Множество A является подмножеством множества B, если любой элемент A принадлежит также множеству B. Это свойство обозначается $A \subseteq B$ (читается: B включает или равно A).

Равенство множеств. Множества A и B равны тогда и только тогда, когда их элементы совпадают. В этом случае пишут $A = B$. Так как при равенстве множеств A и B во множестве A нет элементов, не принадлежащих B, а в B нет элементов не принадлежащих A, то признаком равенства множеств является одновременное выполнение двух условий: $A \subseteq B$ и $B \subseteq A$.

Если $A \subseteq B$ и $B \neq A$, то множество A называется ***собственным подмножеством*** множества B. Обозначается $A \subset B$ (строгое включение). Одним из частных случаев является ситуация, когда элементами некоторого множества являются

другие множества.

Например, пусть A– множество футболистов команды «Спартак», B– множество команд высшей лиги. Тогда $A \in B$.

Если в рамках некоторого класса задач рассматриваются различные множества, то полная совокупность всех элементов, из которых могут формироваться все множества и подмножества, образует универсальное множество – “Универсум” или полное пространство. Обозначается универсальное множество символом U (генеральная совокупность).

Множество может быть задано:

1. Перечислением всех его элементов. Например,

$$A = \{a, b, c, d\},\ B = \{0, 1, 3, 8, 9\}.$$

2. Порождающей процедурой. ***Порождающая процедура*** представляет собой правило получения элементов множества на основе уже имеющихся элементов либо из других объектов. Элементами множества считаются все объекты, которые получены с помощью этой процедуры.

3. Описанием характеристик и свойств, которыми обладают все элементы множества. Например, $A = \{x |\ x \in N, x \le 100\}$.

1.2.1. Основные операции над множествами и их свойства

1. ***Объединение множеств*** $(A \cup B)$ – это множество, состоящее из тех элементов, которые принадлежат хотя бы одному из исходных множеств: $A \cup B = \{x |\ x \in A \text{ или } x \in B\}$.

Пример. $A = \{a, b, c\},\ B = \{b, c, d, m\},\ A \cup B = \{a, b, c, d, m\}$.

Операции над множествами удобно представлять с помощью диаграммы Венна – замкнутой линии, ограничивающей элементы одного множества. Для операции объединения диаграмма представлена на рис.1.1.

2. ***Пересечение множеств*** $(A \cap B)$ – это множество тех и только тех элементов, которые принадлежат и множеству A, и множеству B(рис.1.2): $A \cap B = \{x |\ x \in A \text{ и } x \in B\}$.

Пример. $A = \{a, b, c\},\ B = \{b, c, d, m\},\ A \cap B = \{b, c,\}$.

3. ***Разность множеств*** $(A \setminus B)$ – это множество, состоящее

из тех и только тех элементов множества A, которые не содержатся во множестве B (рис.1.3): $A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}$. Если $A \setminus B = \varnothing$, *то* $A \subseteq B$ (рис.1.4).

4. ***Симметричная разность*** $(A \div B)$ – это множество элементов, принадлежащих множествам A или B за исключением их общих элементов (рис.1.5): $A \div B = \{x \mid (x \in A \text{ и } x \notin B) \text{ или } (x \in B \text{ и } x \notin A)\}$

Рис.1.1

Рис.1.2

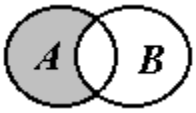


Рис.1.3

Рис.1.4

Рис.1.5

Рис.1.6

5. ***Дополнением*** множества A до множества U (обозначается $\overline{A}$) называется множество всех элементов U, не принадлежащих множеству A (рис.1.6).

Основные свойства операций над множествами. Для всех множеств A, B, C и универсального множества U справедливы следующие равенства:

1. $A \cup B = B \cup A$;
2. $A \cup (B \cup C) = (A \cup B) \cup C$;
3. $A \cap B = B \cap A$;
4. $A \cap (B \cap C) = (A \cap B) \cap C$;
5. $A \cap A = A$;
6. $A \cup A = A$;
7. $A \cup \varnothing = A$;
8. $A \cap \varnothing = \varnothing$;
9. $A \cup U = U$;
10. $A \cap U = A$;
11. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$;
12. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
13. $\overline{\overline{A}} = A$;
14. $\overline{\varnothing} = U$;
15. $\overline{U} = \varnothing$;
16. $A \cup \overline{A} = U$;
17. $A \cap \overline{A} = \varnothing$;
18. $\overline{A \cup B} = \overline{A} \cap \overline{B}$;
19. $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

1.2. Прямое произведение множеств

Вектором *(кортежем)* называется упорядоченный набор элементов. Элементы, образующие вектор, называются ***координатами*** или ***компонентами***. Координаты нумеруются слева направо. Число координат называется ***длиной*** или ***размерностью*** вектора. В отличие от элементов множества координаты вектора могут совпадать. Обозначение вектора: (a, b, c), где a, b, c – координаты вектора. ***Два вектора равны,*** если они имеют одинаковую длину и равны их соответствующие координаты.

Прямым или ***декартовым произведением*** множеств A и B (обозначается $A \times B$), называется множество всех упорядоченных пар (a, b) таких, что $a \in A$ и $b \in B$, т.е. $A \times B = \{(a, b) \mid a \in A; b \in B\}$.

Частный случай: $A = B$. Тогда $A \times B = A \times A$. $A \times A$ обозначается A^2.

Прямым произведением $A_1 \times A_2 \times A_3 \times ... \times A_n$ называется множество всех векторов $(a_1, a_2, a_3, ..., a_n)$ длины n таких, что $a_1 \in A_1;\ a_2 \in A_2; ...;\ a_n \in A_n$.

Пример.

1) $A_1 = \{0, 1\},\ A_2 = \{x, y, z\}$

$A_1 \times A_2 = \{(0, x), (0, y), (0, z), (1, x), (1, y), (1, z)\}$;

2) $A = \{a, b, c, d, e, f, g, h\},\ B = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$A \times B = \{(a, 1), (a, 2), ..., (h, 7), (h, 8)\}$.

Пусть имеется множество A, элементы которого являются символами (буквы, цифры, знаки). Такое множество называется **алфавитом**. Элементы множества A^n – слова. Множество $R \times R = R^2$ – это множество точек (пар координат) плоскости (здесь R – множество всех действительных чисел).

Теорема 1.1. Пусть $A_1, A_2, ..., A_n$ – конечные множества и их мощности известны: $|A_1| = m_1, |A_2| = m_2, ..., |A_n| = m_n$. Тогда $|A_1 \times A_2 \times ... \times A_n| = m_1 \cdot m_2 \cdot ... \cdot m_n$. Частный случай: $|A^n| = |A|^n$.

Проекцией вектора $A = (a_1, a_2, ..., a_n)$ на ось i (обозначается

$пр_i A$) называется его компонента a_i. ***Проекцией вектора*** A на оси $i_1 \ldots i_k$ $(пр_i A)$ называется вектор $\left(a_{i_1}, a_{i_2}, \ldots, a_{i_k}\right)$ длины k.

Если V – множество векторов A_j одинаковой длины, то проекцией V на *i-ю* ось называется *множество проекций* всех A_j *на эту ось*:

$$пр_i V = \left\{ пр_i A_j \mid A_j \in V \right\}.$$

2. ОСНОВНЫЕ ПОЛОЖЕНИЯ БУЛЕВОЙ АЛГЕБРЫ

2.1. Булева алгебра и ее применение

2.1.1. Определение булевой алгебры

Название этого раздела математики связано с именем его основателя – Джорджа Буля (1815 – 1864 г.г.). Используя классическое понятие алгебры, булеву алгебру можно определить как систему $A=(B,\varphi_1,\varphi_2,\ldots,\varphi_n)$, в которой несущим множеством является двухэлементное множество двоичных чисел $B=\{0,1\}$, а $\Omega=\{\varphi_1,\varphi_2,\ldots,\varphi_n\}$ – заданные на этом множестве логические операции, сущность которых раскрывается ниже (см. разд. 2.2.2).

Основные логические операции, - дизъюнкция, конъюнкция и отрицание, - можно интерпретировать как операции, введенные в теории множеств: свойства указанных операций аналогичны свойствам операций объединения, пересечения и дополнения множеств соответственно (см. разд. 1.1.2). Однако логические операции имеют несколько иной смысл; они позволяют формировать простые и сложные высказывания. Все множество логических операций обозначается E_2.

Как правило, существует логическая интерпретация элементов множества B: 1 – истинно; 0 – ложно. В ряде случаев такой смысл не придается, и в качестве элемента множества рассматривается двоичная переменная (ее называют также логическая или булева переменная) x, которая принимает значения x = 0 или x = 1.

2.1.2. Области применения булевой алгебры

Булева алгебра применяется:

1) как средство алгоритмического описания в языках программирования для определения логических условий;

2) как средство формирования логических высказываний в математической логике, лингвистике, теории искусственного интеллекта;

3) как средство разработки и описания дискретных технических систем.

Алгебра логики позволяет производить анализ и синтез логических устройств. *Анализ* – это поиск аналитического выражения, которое описывает работу системы. *Синтез* – обратная задача: создание технического устройства на основе математического описания средствами булевой алгебры.

2.1.3. Высказывания

Одним из базовых понятий в булевой алгебре является понятие ***высказывания***.

Высказывание – это любое повествовательное предложение, в отношении которого имеет смысл утверждение о его истинности или ложности. Обычно высказывания обозначаются буквами латинского алфавита: a, b, c, A, B, C. Для каждого высказывания вводится значение ***истинности***, которое может принимать одно из двух возможных: значений *1*– истина, *0* – ложь.

Пример. Рассмотрим справедливость утверждений:

a) число 4 – четное;

b) снег – красный;

c) $2 \times 2 = 5$.

Значения истинности данных высказываний следующие: $a = 1, b = 0, c = 0$.

Два высказывания A и B называются ***эквивалентными,*** если их значения истинности совпадают. Значение истинности может быть постоянным либо изменяется в зависимости от

обстоятельств. Изменяемое высказывание может рассматриваться как переменный параметр – двоичная переменная, принимающая одно из двух значений (обозначается x, y, z).

2.2. Функции алгебры логики

2.2.1. Понятие функции и способы ее задания

Пусть имеется n двоичных переменных $x_1, x_2, ..., x_n$. Каждая из них в некотором конкретном случае может принимать значение *0* или *1*. Полученный набор элементов есть двоичный вектор длины n. Каждому конкретному набору можно поставить в соответствии одно из значений *0* или *1*.

Функция f, задающая однозначное отображение множества всевозможных наборов значений двоичных переменных $x_1, x_2, ..., x_n$ во множество $\{0, 1\}$ называется ***функцией алгебры логики*** (или логической функцией, булевой функцией, переключательной функцией):

$$f(x_1, x_2, ..., x_n) = y;\ (y = 0 \text{ или } y = 1).$$

Таким образом, логическая *функция* – это зависимость, которая устанавливает связь между сочетанием значений входных двоичных переменных и двоичным значением этой функции.

Способы задания функции. Логическая функция может быть задана:

1) математическим выражением (формулой);

2) таблицей.

Таблица является наиболее общим и универсальным способом задания функции. В её левой части перечисляют всевозможные наборы значений двоичных переменных, а в правой — значения функции на этих наборах.

Такие таблицы, описывающие функции, называют *таблицами истинности*. В таблицах 2.1 и 2.2 приведены примеры таблиц истинности.

Оценим число возможных наборов (число строк входных переменных).

Таблица 2.1

x_1	x_2	...	x_n	$f(x_1, x_2, ..., x_n)$
0	0		0	$f(0,0,...,0)$
1	0		1	$f(1,0,...,1)$
...	...	...	...	...
1	1		1	$f(1,1,...,1)$

Таблица 2.2

x_1	x_2	x_3	$y = f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Конкретный набор – это вектор значений $(b_1, b_2, ..., b_n)$. Количество наборов – это мощность прямого произведения n двухэлементных множеств B: $a = \left|B^n\right| = \left|B\right|^n = 2^n$, где n – число входных элементов.

Оценим возможное количество вариантов логических функций от n переменных. Множество вариантов логической функции можно представить как прямое произведение: $M = B_1 \times B_2 \times ... \times B_i \times ... \times B_a = B^a$, где B_i – значение функции на наборе i.

Таким образом, общее количество функций от n переменных

$m = \left|B^a\right| = \left|B\right|^a = 2^a$, где $a = 2^n$.

Наборы, на которых функция равна единице, называют *единичными* наборами, а наборы, на которых функция равна нулю, называют *нулевыми.*

Тождественно истинные, тождественно ложные, частично определенные функции. Если функция при любых значениях аргументов принимает значение *0*, то такую функцию называют *нулевой* или *константой 0* (тождественно ложная функция).

Функция, которая на всех наборах равна *1*, называется *единичной* или *константой 1* (тождественно истинная функция).

Если функция определена не на всех наборах аргументов, то она называется *не полностью определенной* или *частично определенной*.

Две булевы функции $f(x_1, x_2, ..., x_n)$ и $\varphi(x_1, x_2, ..., x_n)$ называют *равными,* если для всех возможных наборов значений аргументов они принимают одинаковые значения и таким образом имеют одну и ту же таблицу истинности, и записывают: $f(x_1, x_2, ..., x_n) = \varphi(x_1, x_2, ..., x_n)$.

Говорят, что булева функция $f(x_1, x_2, ..., x_n)$ существенно зависит от аргумента x_i, если

$$f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \neq f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n),$$

хотя бы для одного набора остальных аргументов. В противном случае x_i называется ***несущественной*** или ***фиктивной*** переменной (ее можно исключить).

2.2.2. Элементарные логические операции

Из множества логических функций выделяется ряд наиболее простых операций, которые имеют ясную логическую интерпретацию:

1) ***отрицание*** (инверсия)

$f_1(x) = \bar{x}$ (читается: не x).

Отрицание – это функция, выражающая высказывание, которое истинно, если высказывание x ложно, и ложно, если x истинно (табл.2.3);

2) ***дизъюнкция*** (логическое сложение)

$f_2(x, y) = x \vee y$ (читается: "x или y").

Дизъюнкция – это функция, выражающая высказывание, которое истинно тогда и только тогда, когда, по крайней мере, одно из высказываний x или y является истинным (табл.2.4);

3) ***конъюнкция*** (логическое умножение)

$f_3(x, y) = x \wedge y$ (читается: "x и y").

Для этой операции применяются также следующие формы записи:

$$f_3(x,y)=xy=x\&y.$$

Конъюнкция – это функция, выражающая высказывание, которое истинно только в том случае, если истинны оба высказывания x и y (табл.2.5);

4) **и*мпликация***

$$f_4(x, y) = x \to y \quad \text{(читается : “если } x \text{, то } y\text{”).}$$

Функция f_4 принимает значение ложно только тогда, когда x истинно, а y ложно (табл.2.6);

Таблица 2.3

x	$\bar{x}$
0	1
1	0

Таблица 2.4

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.5

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 2.6

x_1	x_2	$x_1 \to x_2$
0	0	1
0	1	1
1	0	0
1	1	1

5) ***эквивалентность*** (равнозначность)

$$f_5(x, y) = x \sim y \quad \text{(читается: “} x \text{ равно } y\text{”).}$$

Функция $f_5 = 1$ тогда и только тогда, когда значения обоих аргументов совпадают (табл.2.7);

6) ***сложение по модулю два*** (неравнозначность)

$$f_6(x, y) = x \oplus y.$$

Функция f_6 истинна тогда и только тогда, когда значения аргументов различны (табл.2.8). Как следует из табл. 2.7 и 2.8, функция f_6 является обратной к f_5;

7) ***штрих Шеффера***

$$f_7(x, y) = x \mid y.$$

Операция обратная по отношению к конъюнкции (функция ложна, только если оба аргумента истинны) табл.2.9;

8) ***стрелка Пирса***

$$f_8(x, y) = x \downarrow y.$$

Функция обратная дизъюнкции (f_8 истинно, только когда x и y ложны) табл.2.10;

Таблица 2.7

x_1	x_2	$x_1 \sim x_2$
0	0	1
0	1	0
1	0	0
1	1	1

Таблица 2.8

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 2.9

x_1	x_2	$x_1 \mid x_2$
0	0	1
0	1	1
1	0	1
1	1	0

Таблица 2.10

x_1	x_2	$x_1 \downarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	0

Наиболее важными функциями являются первые три. Остальные могут быть выражены через эти три функции. С использованием трех основных функций (дизъюнкции, конъюнкции и отрицания) могут образовываться более сложные функции. Поэтому можно дать еще одно определение булевой алгебры. ***Булевой алгеброй*** называется алгебра типа $\left(B, \vee, \wedge, \bar{}\ \right)$, несущим множеством которой является множество двоичных чисел, а операциями - конъюнкция, дизъюнкция и отрицание.

2.2.3. Свойства основных логических функций

Основные логические функции обладают следующими свойствами:

1) *коммутативность:* $a \vee b = b \vee a$; $a \cdot b = b \cdot a$;

2) *ассоциативность:* $a \vee (b \vee c) = (a \vee b) \vee c$; $a \cdot (b \cdot c) = (a \cdot b) \cdot c$;

3) *идемпотентность конъюнкции и дизъюнкции:*

$a \vee a = a$; $a \cdot a = a$;

4) *дистрибутивность:*

а) *конъюнкции относительно дизъюнкции:*

$a \cdot (b \vee c) = (a \cdot b) \vee (a \cdot c)$;

б) *дизъюнкции относительно конъюнкции:*

$a \vee (b \cdot c) = (a \vee b) \cdot (a \vee c)$;

5) *двойное отрицание:* $\bar{\bar{a}} = a$;

6) *правило де Моргана:* $\overline{a \cdot b} = \bar{a} \vee \bar{b}$; $\overline{a \vee b} = \bar{a} \cdot \bar{b}$;

7) *правило склеивания:* $a \cdot b \vee a \cdot \bar{b} = a$; $(a \vee b) \cdot (a \vee \bar{b}) = a$;

8) *правило поглощения:* $a \vee ab = a$; $a \cdot (a \vee b) = a$;

$a \vee \bar{a} b = a \vee b$; $a \cdot (\bar{a} \vee b) = a \cdot b$;

9) *действия с константами*:

$$a \vee 0 = a; \qquad a \cdot 0 = 0;$$
$$a \vee 1 = 1; \qquad a \cdot 1 = a;$$
$$a \vee \bar{a} = 1; \qquad a \cdot \bar{a} = 0.$$

Свойства основных булевых функций доказываются либо путем преобразования выражений, либо на основе сопоставления таблиц истинности правой и левой части равенства.

Пример. Доказать, что $\overline{a \vee b} = \bar{a} \cdot \bar{b}$.

С учетом таблиц истинности элементарных логических операций определяем последовательно значения функций, указанных в верхней строке для всех возможных значений аргументов a и b, т.е. построим для них соответствующие им таблицы истинности (табл.2.11).

Таблица 2.11

a	b	$a \vee b$	$\overline{a \vee b}$	$\bar{a}$	$\bar{b}$	$\bar{a} \cdot \bar{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Так как значения функций $\overline{a \vee b}$ и $\bar{a} \cdot \bar{b}$ на всех наборах совпадают, то эти функции равны.

2.2.4. Задание функции формулой. Эквивалентные преобразования логических выражений

Понятие формулы вводится для формализации представления и записи простого или сложного высказывания. Формула рассматривается как некоторый способ реализации функции и вводится индуктивно в соответствии со следующим правилом: если A и B – высказывания (простые или сложные, постоянные или переменные), то запись значения истинности каждого из этих высказываний – есть формула; если A и B – формулы, то выражения «A * B» и «$\bar{A}$» (где символ * обозначает знак одной из рассмотренных выше элементарных логических операций) – тоже формулы. Таким образом, рассмотренные выше выражения, которыми описывались элементарные логические

операции и свойства основных логических операций, - суть формулы. Применение по отношению к ним указанного правила позволяет получить новые формулы, соответствующие более сложным высказываниям.

Новые формулы могут быть получены на основе использования понятия суперпозиции функций. Суперпозицией функций $f_1, f_2, \ldots, f_n$ называется функция f, полученная путем подстановки функций $f_1, f_2, \ldots, f_n$ друг в друга и переименования переменных.

Пусть имеется множество логических функций, заданных формулами $E=\{f_1, f_2, \ldots, f_m\}$; при этом говорят, что имеет место множество формул над Е. Тогда новую формулу можно получить используя следующее правило:

Пусть $f_0(x_1,x_2,\ldots,x_n)$ – некоторая формула над E. *Если* $A_1, A_2, \ldots, A_n$ – либо символы переменных, либо другие формулы над E, то $f_0(A_1,A_2,\ldots,A_n)$ – тоже формула над E.

Пример. Пусть функция задана формулой $f_0(z_1, z_2) = z_1 \to z_2$, и при этом имеет место равенство $z_1 = A_1 = x_1 \vee x_2$, $z_2 = A_2 = x_3$. Тогда новую формулу над E можно получить путем подстановки A_1 и A_2 в исходную формулу: $f_0(x_1, x_2, x_3) = (x_1 \vee x_2) \to x_3$.

Полученную формулу вновь представим как исходную, и, полагая далее $A_1 = f_0(x_1, x_2, x_3)$, $A_2 = x_3$, $A_3 = x_1 \cdot x_2$, делаем вновь подстановку.

Тогда новая формула над E:

$$f_0(x_1, x_2, x_3) = \big(\big((x_1 \vee x_2) \to x_3\big) \vee x_3\big) \to x_1 \cdot x_2.$$

Логические операции обладают различным приоритетом, с точки зрения порядка выполнения их в выражении. Принят следующий порядок выполнения операций в булевой алгебре: в первую очередь вычисляются выражения, над которыми стоит знак отрицания, далее выполняются операции конъюнкции $(\wedge)$, а затем дизъюнкции $(\vee)$.

Если выражение, заключенное в скобках, представляет конъюнкцию или имеет общий знак отрицания, то скобки опускаются.

Сопоставляя введенные выше понятия логической функции и формулы, следует иметь ввиду, что логическая функция - это зависимость между логическими переменными, однозначно определяемая таблицей истинности, а формула это выражение, которое используется для описания логической функции, причем одна и та же логическая функция может описываться несколькими формулами.

Пример. Рассмотрим две формулы:

$f(x_1,x_2)=x_1 \mid x_2$ и $f(x_1,x_2)=\overline{x_1 \cdot x_2}$.

Несложно показать, что обе формулы представляют одну и ту же функцию, так как таблицы истинности у них одинаковы.

Формулы, соответствующие одной и той же функции, называются ***эквивалентными*** или ***равносильными.***

Две формулы U и B называются ***эквивалентными (равносильными)***, если они реализуют одну и ту же функцию. При этом записывают: $U = B$.

Эквивалентные преобразования логических выражений. ***Эквивалентные преобразования*** – это такие преобразования формул, при которых сохраняется их эквивалентность. Преобразование называется ***эквивалентным***, если исходная формула $U=(x_1, x_2, ..., x_n)$ и полученная в результате преобразования формула $B=(x_1, x_2, ..., x_n)$ принимают одинаковые значения на каждом наборе значений аргументов.

Эквивалентное преобразование осуществляется на основе сопоставления таблиц истинности, либо на основе применения свойств основных логических операций.

Покажем примеры эквивалентных преобразований, которые позволяют получить новые формулы для описания функций f_4 - f_8 (см. п. 1.2.2), используя только знаки операций конъюнкции, дизъюнкции и отрицания.

1. Преобразование формулы, описывающей функцию $f_4 : a \to b = \bar{a} \vee b$. Справедливость преобразования доказывается соответствующей таблицей истинности.

a	b	$\bar{a}$	$a \to b$	$\bar{a} \vee b$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

2.Преобразование формулы, описывающей функцию $f_5 : a \sim b = ab \vee \bar{a}\bar{b} = (a \to b)(b \to a)$. Справедливость преобразования доказывается соответствующей таблицей истинности.

a	b	$a \sim b$	ab	$\bar{a}\bar{b}$	$ab \vee \bar{a}\bar{b}$
0	0	1	0	1	1
0	1	0	0	0	0
1	0	0	0	0	0
0	0	1	1	0	1

3. Функция f_6:
$a \oplus b = \overline{a \sim b} = \overline{ab \vee \bar{a}\bar{b}} = \overline{ab} \cdot \overline{\bar{a}\bar{b}} = (\bar{a} \vee \bar{b}) \cdot (a \vee b)$.

4. Функция $f_7 : a \mid b = \overline{a \cdot b}$.

5. Функция $f_8 : a \downarrow b = \overline{a \vee b}$.

Формулы, из которых построена некоторая исходная формула, называются ***подформулами***.

Чаще всего эквивалентные преобразования основаны на замене подформул на эквивалентные им подформулы. Если в формуле U заменить подформулу B на эквивалентную ей под формулу B', то формула U перейдет в эквивалентную ей формулу U'.

2.2.5. Двойственные функции

Логическая функция $f^*(x_1, x_2, ..., x_n)$, равная $\overline{f}(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)$, называется ***двойственной*** по отношению к функции $f(x_1, x_2, ..., x_n)$.

Очевидно свойство взаимности двойственных функций:

$$f^{**}(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n).$$

Теорема 2.1. Если некоторая формула Φ реализует функцию f, то формула Φ^*, реализующая двойственную функцию f^*, может быть получена из Φ путем замены всех подформул на двойственные им. То есть если
$\Phi(x_1,x_2,...,x_n) = f[f_1(x_1,x_2,...,x_n), f_2(x_1,x_2,...,x_n),...,f_k(x_1,x_2,...,x_n)]$, то
$\Phi^*(x_1,x_2,...,x_n) = f^*[f_1^*(x_1,x_2,...,x_n), f_2^*(x_1,x_2,...,x_n),...,f_k^*(x_1,x_2,...,x_n)]$.

В частном случае из теоремы 2.1 вытекает следующее правило.

Если формула Φ содержит только операции $\wedge, \vee, \bar{}$ и константы $0, 1$, то для получения формулы Φ^*, двойственной к Φ, необходимо, сохраняя порядок выполнения операций, везде заменить 0 на 1 (1 на 0), операцию $\vee$ на & (операцию & на $\vee$).

Можно сформулировать следующие принципы двойственности для формул:

1) если две формулы эквивалентны, то есть $\Phi_1(x_1,x_2,...,x_n) = \Phi_2(x_1,x_2,...,x_n)$,
то эквивалентны и отрицания этих формул $\bar{\Phi}_1(x_1,x_2,...,x_n) = \bar{\Phi}_2(x_1,x_2,...,x_n)$;

2) если формулы Φ_1 и Φ_2 эквивалентны, то и двойственные им функции тоже эквивалентны $\Phi_1^*(x_1,x_2,...,x_n) = \Phi_2^*(x_1,x_2,...,x_n)$;

3) если две формулы эквивалентны, то будут эквивалентны и формулы, полученные из исходных путем замены аргументов на их отрицания

$$\Phi_1(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) = \Phi_2(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n).$$

2.3. Специальные разложения логических функций

2.3.1. Конъюнктивная и дизъюнктивная нормальные формы

Любая функция булевой алгебры может быть представлена некоторыми формулами специального вида. ***Нормальные формы*** – это некоторое стандартное представление функции с помощью элементарных конъюнкций и элементарных дизъюнкций.

Элементарной дизъюнкцией (ЭД) называют дизъюнкцию нескольких переменных или их отрицаний. Например, $d_i = a_1 \vee a_2 \vee \bar{a}_3$.

Элементарной конъюнкцией (ЭК) называют конъюнкцию нескольких переменных или их отрицаний. Например, $k_i = a_1 \cdot \bar{a}_2 \cdot a_3$.

Дизъюнктивной нормальной формой (ДНФ) некоторой заданной функции называется формула, которая имеет вид дизъюнкции элементарных конъюнкций, например, $a_1\,\bar{a}_2\,a_3 \vee \bar{a}_1\,\bar{a}_2\,a_3$.

Конъюнктивной нормальной формой (КНФ) некоторой заданной функции называется формула, которая имеет вид конъюнкции элементарных дизъюнкций.

Если заданная функция уже представлена некоторой формулой, то путем эквивалентных преобразований эта формула может быть приведена к равносильной ей формуле в соответствующей нормальной форме (КНФ или ДНФ).

Порядок действий, при приведении исходной формулы к нормальной форме, следующий:

1) все функции выражаются через дизъюнкцию, конъюнкцию и отрицание;

2) если в выражении над несколькими элементами имеются знаки отрицания, то следует, используя формулы де Моргана, изменить выражение так, чтобы отрицание относилось только к одной переменной;

3) дальнейшее преобразование производится с использованием свойств элементарных операций.

Пример. $A \cdot \overline{\overline{B} \cdot C} \cdot (M \vee \overline{N}) = A(B \vee \overline{C})(M \vee \overline{N}) = (AB \vee A\overline{C})(M \vee \overline{N}) =$
$= ABM \vee AB\overline{N} \vee A\overline{C}M \vee A\overline{C}\,\overline{N}$.

Упрощение логических формул на основе применения понятий тождественно истинных и тождественно ложных форм. После приведения к нормальной форме можно существенно упрощать выражения логических функций.

Утверждение 1. Для того чтобы элементарная дизъюнкция была тождественно истинной, необходимо и достаточно, чтобы среди её слагаемых нашлась хотя бы одна переменная и её отрицание ($x_i \vee \bar{x}_i = 1$).

Утверждение 2. Для того чтобы элементарная конъюнкция была тождественно ложной, необходимо и достаточно, чтобы среди её сомножителей оказалась хотя бы одна переменная и её отрицание ($x_i \,\&\, \bar{x}_i = 0$).

Указанные два утверждения используются для упрощения выражений следующим образом:

В случае принадлежности логической формулы к КНФ рассматривается каждый ее сомножитель, и если в какой-то из них входят вместе x_i и $\bar{x}_i$, то он равен единице и его можно исключить. Если все сомножители равны единице, то такая функция тождественно истинна.

В случае принадлежности формулы к ДНФ рассматривается каждое слагаемое. Если в каком-либо слагаемом встречается произведение $x_i \cdot \bar{x}_i$, то это слагаемое равно нулю и его можно исключить.

Пример. Установить характер истинности формулы

$$(P \cdot (P \to Q)) \to Q = \overline{P \cdot (\bar{P} \vee Q)} \vee Q = \bar{P} \vee \overline{(\bar{P} \vee Q)} \vee Q = \bar{P} \vee (P \cdot \bar{Q}) \vee Q =$$

$$= \bar{P} \vee \bar{Q} \vee Q = 1.$$

3.3.2. Совершенно нормальные конъюнктивная и дизъюнктивная формы

Понятие совершенно нормальных конъюнктивных и дизъюнктивных форм. Пусть имеется n переменных: $x_1, x_2, ..., x_n$. Будем формировать из них строки так, чтобы выполнялись три условия:

1) в каждую строку входят все n переменных;

2) каждая переменная входит в строку только один раз;

3) в строку входит либо сама переменная, либо её отрицание, но не одновременно то и другое.

Число таких строк 2^n.

Элементарная дизъюнкция, сформированная в соответствии с указанными требованиями, образует логическую конструкцию вида

$d_i = x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee ... \vee x_n$ и называется ***макстермом*** .

Элементарная конъюнкция образует логическую конструкцию вида

$k_i = x_1 \cdot x_2 \cdot \bar{x}_3 \cdot ... \cdot x_n$ и называется ***минтермом*** .

Совершенной дизъюнктивной нормальной формой (СДНФ) логической функции $f(x_1, x_2, ..., x_n)$ называют представление её в виде дизъюнкции минтермов, построенных из аргументов $x_1, x_2, ..., x_n$:

$$f(x_1, x_2, ..., x_n) = x_1 \cdot x_2 \cdot ... \cdot x_n \vee \bar{x}_1 \cdot x_2 \cdot ... \cdot x_n \vee x_1 \bar{x}_2 ... x_n \vee ... \vee \bar{x}_1 \bar{x}_2 ... x_n$$

Совершенной конъюнктивной нормальной формой (СКНФ) логической функции $f(x_1, x_2, ..., x_n)$ называют представление её в виде конъюнкции макстермов, построенных из аргументов рассматриваемой функции:

$$f(x_1, x_2, ..., x_n) = (x_1 \vee x_2 \vee ... \vee x_n) \cdot (\bar{x}_1 \vee x_2 \vee ... \vee x_n) \cdot ... \cdot (\bar{x}_1 \vee \bar{x}_2 \vee ... \vee \bar{x}_n)$$

Каждая логическая функция может быть представлена единственным образом в совершенной дизъюнктивной нормальной форме и совершенной конъюнктивной нормальной форме. В полученной формуле заданной функции могут присутствовать или все термы, которые можно построить из n переменных, или часть из них, но дважды один терм входить в выражение не может.

Приведение функции к совершенно нормальной форме называют ее ***разложением по всем переменным или по термам***.

Если имеется формула, описывающая некоторую заданную функцию, то с применением эквивалентных преобразований ее можно привести к совершенной дизъюнктивной нормальной форме, применяя следующую последовательность действий:

1. Логическая формула $f(x_1, x_2, ..., x_n)$ приводится к дизъюнктивной нормальной форме

$$f(x_1, x_2, ..., x_n) = \varphi_1(x_1, x_2, ..., x_n) \vee \varphi_2(x_1, x_2, ..., x_n) \vee ... .$$

При этом каждое слагаемое представляет собой конъюнкцию некоторого числа переменных, но не обязательно всех.

2. Пусть слагаемое $\varphi_j(x_1, x_2, ..., x_n)$ не содержит ни x_i, ни $\bar{x}_i$. В этом случае к нему конъюнктивно присоединяют тождественно-истинную форму $(x_i \vee \bar{x}_i)$:

$$\varphi_j(x_1, x_2, ..., x_n) \cdot (x_i \vee \bar{x}_i) = \varphi_j(x_1, x_2, ..., x_n) x_i \vee \varphi_j(x_1, x_2, ..., x_n) \bar{x}_i .$$

Если отсутствуют несколько переменных, то операцию нужно повторить для всех отсутствующих сомножителей. По окончании слагаемое будет удовлетворять первому условию (см. п.2.3.2).

3. Если в каком-либо слагаемом имеется несколько одинаковых сомножителей, то из них оставляется один.

4. Если слагаемое содержит $x_i \cdot \bar{x}_i$, то это слагаемое можно исключить.

5. Если среди всех слагаемых окажется два или несколько одинаковых, то они заменяются одним.

В результате указанных операций формула будет приведена к СДНФ.

При приведении к совершенной конъюнктивной нормальной форме последовательность действий остается той же, но все действия заменяются на двойственные.

Существует еще один метод приведения функции к совершенной нормальной форме.

Выражение в форме СКНФ включает столько сомножителей, сколько в таблице истинности содержится наборов, на которых функция равна нулю. Каждый сомножитель содержит дизъюнкцию всех входных переменных.

При этом если $a_1, a_2, ..., a_n$ – набор, где функция равна нулю, то в элементарную дизъюнкцию включают x_i, если $a_i = 0$ и $\bar{x}_i$, если $a_i = 1$.

Правила для построения совершенной дизъюнктивной нормальной формы аналогичны, только значения заменяются на двойственные.

Пример. Рассмотрим пример приведения заданной логической функции к форме СДНФ, с использованием обоих известных способов.

$$f(x,y,z)=(x\oplus y)\to \bar{y}\,\&\,\bar{x}\vee x\,\&\,z\,\&\,(y\vee \bar{x})=\overline{(\bar{x}\vee\bar{y})\,\&\,(x\vee y)}\vee$$
$$\vee\,\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z\vee x\,\&\,z\,\&\,\bar{x}=\overline{\bar{x}\vee\bar{y}}\vee\overline{x\vee y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\,\&\,(z\vee\bar{z})\vee\bar{x}\,\&\,\bar{y}\,\&\,(z\vee\bar{z})\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\,\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{z}\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\,\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{z}$$

x	y	z	$x\oplus y$	$\bar{x}\,\&\,\bar{y}$	$x\,\&\,z\,\&\,(y\vee\bar{x})$	$f(x,y,z)$
0	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	1	0	0	0
1	0	0	1	0	0	0
0	1	1	1	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	0	1	1

$$f(x,y,z)=\bar{x}\bar{y}\bar{z}\vee\bar{x}y\bar{z}\vee xy\bar{z}\vee xyz.$$

Пример. Рассмотрим пример приведения заданной логической функции к форме СКНФ, с использованием обоих известных способов.

x	y	z	$z\oplus\bar{x}$	$\bar{y}\vee\bar{z}\vee x$	$\bar{x}\to zy$	$f(x,y,z)$
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	1	0	1	1	0	0
1	0	0	0	0	1	0
0	1	1	0	1	1	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	1	1	1	1	1

$$f(x,y,z)=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z)$$

$$f(x,y,z)=(z\oplus\bar{x})(\bar{y}\vee\bar{z}\vee x)(\bar{x}\to zy)=(z\vee\bar{x})(\bar{z}\vee\bar{\bar{x}})(x\vee\bar{y}\vee\bar{z})(\bar{\bar{x}}\vee zy)=$$

$$=(z\vee\bar{x})(\bar{z}\vee x)(x\vee\bar{y}\vee\bar{z})(x\vee z)(x\vee y)=(z\vee\bar{x}\vee y\bar{y})(\bar{z}\vee x\vee y\bar{y})(x\vee\bar{y}\vee\bar{z})\&$$

$$\&(x\vee z\vee y\bar{y})(x\vee y\vee z\bar{z})=(z\vee\bar{x}\vee\bar{y})(z\vee\bar{x}\vee y)(\bar{z}\vee x\vee\bar{y})(\bar{z}\vee x\vee y)(x\vee\bar{y}\vee\bar{z})\&$$

$$\&(x\vee z\vee y)(x\vee z\vee\bar{y})(x\vee y\vee z)(x\vee y\vee\bar{z})=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)\&$$

$$\&(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z).$$

2.4. Минимизация булевых функций

2.4.1. Понятие минимизации

Задача минимизации булевой функции состоит в построении такой ДНФ (или КНФ) для некоторой заданной функции, которая реализует эту функцию и имеет наименьшее суммарное число операций и символов в формуле, т.е. имеет минимальную сложность. Решение этой задачи важно, когда логические функции реализуются техническими устройствами. Одной из основных целей минимизации является упрощение логических устройств и достижение максимальной экономичности разрабатываемых систем.

Для минимизации булевых функций в целях получения самых простых выражений используется ряд методов, среди которых наибольшее применение находят:

1) метод неопределенных коэффициентов;
2) метод Квайна-Мак Класки;
3) метод карт Карно.

2.4.2. Метод неопределенных коэффициентов

Сущность метода заключается в представлении функции в самом общем виде в ДНФ и введении коэффициентов a_i, значения которых (0 или 1) подбираются таким образом, чтобы формула была минимальной. Пусть дана функция трех переменных. Представим формулу этой функции в виде ДНФ самого общего вида:

$$f(x_1, x_2, ..., x_n) = a_1x_1 \vee a_2\bar{x}_1 \vee a_3x_2 \vee a_4\bar{x}_2 \vee a_5x_3 \vee a_6\bar{x}_3 \vee a_7x_1x_2 \vee$$
$$\vee a_8\bar{x}_1x_2 \vee a_9x_1\bar{x}_2 \vee a_{10}\bar{x}_1\bar{x}_2 \vee a_{11}x_1x_3 \vee a_{12}\bar{x}_1x_3 \vee a_{13}x_1\bar{x}_3 \vee a_{14}\bar{x}_1\bar{x}_3 \vee$$
$$\vee a_{15}x_2x_3 \vee a_{16}\bar{x}_2x_3 \vee a_{17}x_2\bar{x}_3 \vee a_{18}\bar{x}_2\bar{x}_3 \vee a_{19}\bar{x}_1\bar{x}_2\bar{x}_3 \vee a_{20}x_1\bar{x}_2\bar{x}_3 \vee \quad (2.1)$$
$$\vee a_{21}\bar{x}_1x_2\bar{x}_3 \vee a_{22}\bar{x}_1\bar{x}_2x_3 \vee a_{23}\bar{x}_1x_2x_3 \vee a_{24}x_1\bar{x}_2x_3 \vee a_{25}x_1x_2\bar{x}_3 \vee a_{26}x_1x_2x_3.$$

Термы, содержащие k переменных, будем называть термами ранга k. В выражении (2.1) представлены все возможные формы термов.

Если записать уравнение (2.1) для всех возможных значений аргументов x_1, x_2, x_3, то получим систему 2^n уравнений (в нашем случае $n=3$):

1) $f(1,1,1) = a_1 \vee a_3 \vee a_5 \vee a_7 \vee a_{11} \vee a_{15} \vee a_{26}$;
2) $f(1,1,0) = a_1 \vee a_3 \vee a_6 \vee a_7 \vee a_{13} \vee a_{17} \vee a_{25}$;
3) $f(1,0,1) = a_1 \vee a_4 \vee a_5 \vee a_9 \vee a_{11} \vee a_{16} \vee a_{24}$;
4) $f(0,1,1) = a_2 \vee a_3 \vee a_5 \vee a_8 \vee a_{12} \vee a_{15} \vee a_{23}$; (2.2)
5) $f(1,0,0) = a_1 \vee a_4 \vee a_6 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20}$;
6) $f(0,1,0) = a_2 \vee a_3 \vee a_6 \vee a_8 \vee a_{14} \vee a_{17} \vee a_{21}$;
7) $f(0,0,1) = a_2 \vee a_4 \vee a_5 \vee a_{10} \vee a_{12} \vee a_{16} \vee a_{22}$;
8) $f(0,0,0) = a_2 \vee a_4 \vee a_6 \vee a_{10} \vee a_{14} \vee a_{18} \vee a_{19}$.

Для заданной конкретной функции конкретные значения левой части (2.2) нам известны. Если набор x_1, x_2, x_3 такой, что функция на нем принимает значение *0*, то все коэффициенты в правой части равны *0*. Эти нулевые коэффициенты вычеркиваются и в остальных уравнениях. В каждом оставшемся уравнении приравниваем единице коэффициент, определяющий конъюнкцию наименьшего ранга (ранг – число переменных), а остальные коэффициенты в правой части уравнения приравниваем нулю с учетом ранее сделанных подстановок. После подстановки коэффициентов в уравнение (2.1) получаем результирующее выражение булевой функции.

Пример. Минимизировать заданное выражение

$$f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3.$$

После вычисления значений функции на всех наборах и подстановки в (2.2) получим:

1) $a_1 \vee a_3 \vee a_5 \vee a_7 \vee a_{11} \vee a_{15} \vee a_{26} = 1$;
2) $a_1 \vee a_3 \vee a_6 \vee a_7 \vee a_{13} \vee a_{17} \vee a_{25} = 1$;

3) $a_1 \vee a_4 \vee a_5 \vee a_9 \vee a_{11} \vee a_{16} \vee a_{24} = 1$;

4) ~~$a_2 \vee a_3 \vee a_5 \vee a_8 \vee a_{12} \vee a_{15} \vee a_{23} = 0$~~;

5) $a_1 \vee a_4 \vee a_6 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20} = 1$;

6) ~~$a_2 \vee a_3 \vee a_6 \vee a_8 \vee a_{14} \vee a_{17} \vee a_{21} = 0$~~;

7) ~~$a_2 \vee a_4 \vee a_5 \vee a_{10} \vee a_{12} \vee a_{16} \vee a_{22} = 0$~~;

8) $a_2 \vee a_4 \vee a_6 \vee a_{10} \vee a_{14} \vee a_{18} \vee a_{19} = 1$.

Все коэффициенты, входящие в нулевые наборы, должны быть равны нулю, т.е.

$a_2 = a_3 = a_4 = a_5 = a_6 = a_8 = a_{10} = a_{12} = a_{14} = a_{15} = a_{16} = a_{17} = a_{21} = a_{22} = a_{23} = 0$.

Остается рассмотреть единичные наборы:

1) $a_1 \vee a_7 \vee a_{11} \vee a_{26} = 1$;

2) $a_1 \vee a_7 \vee a_{13} \vee a_{25} = 1$;

3) $a_1 \vee a_9 \vee a_{11} \vee a_{24} = 1$;

5) $a_1 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20} = 1$;

8) $a_{18} \vee a_{19} = 1$.

В первых четырех уравнениях конъюнкция наименьшего ранга – a_1, а в последнем – a_{18}. Таким образом, принимаем $a_1 = 1,\ a_{18} = 1$. Остальные коэффициенты принимаем равными нулю.

Подставляя коэффициенты в выражение (2.1), получаем минимальную
ДНФ заданной функции:

$$f(x, y, z) = x_1 \vee \bar{x}_2 \bar{x}_3.$$

2.4.3. Метод Квайна – Мак Класки

Метод Квайна. При минимизации методом Квайна исходная функция задается в СДНФ. Сущность метода состоит в поэтапном упрощении выражений на основе операций склеивания.

Шаг 1. Нахождение первичных импликант. Все термы сравниваются между собой попарно. Если два терма имеют вид $m_i = a x_i$, а $m_j = a\bar{x}_i$ (где a – конъюнкция нескольких переменных), тогда вместо m_i и m_j выписываются a, которые являются термом $(n-1)$ порядка. Исключаемые термы помечаются. Далее

сравниваются все полученные термы $(n-1)$ ранга. В результате склеивания выписываются термы $(n-2)$ ранга. Процесс продолжается до тех пор, пока дальнейшее склеивание становится невозможным. Не исключенные в процессе выполнения указанной процедуры термы исходной функции, а также термы, которые были получены в результате склеивания, будем называть первичными или простыми импликантами..

Пример. Пусть задано следующее выражение в форме СДНФ

$$f(x_1,x_2,x_3)=\bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee \bar{x}_1x_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee$$
$$\vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1x_2\bar{x}_3x_4.$$

Пронумеруем все входящие в него минтермы

$$\bar{x}_1\bar{x}_2x_3x_4-1,\ \bar{x}_1x_2\bar{x}_3\bar{x}_4-2,\ \bar{x}_1x_2\bar{x}_3x_4-3,\ \bar{x}_1x_2x_3x_4-4,\ x_1\bar{x}_2\bar{x}_3x_4-5,$$
$$x_1\bar{x}_2x_3x_4-6,\ x_1x_2\bar{x}_3\bar{x}_4-7,\ x_1x_2\bar{x}_3x_4-8.$$

Производим склеивания следующим образом:

склеиваемые термы 4-го ранга: полученный терм 3-го ранга:

$1 — 4 \to (x_2)\ \ \bar{x}_1x_3x_4$ (1); $3 — 8 \to (x_2)\ \ x_2\bar{x}_3x_4$ (6);

$1 — 6 \to (x_1)\ \ \bar{x}_2x_3x_4$ (2); $5 — 6 \to (x_2)\ \ x_1\bar{x}_2x_4$ (7);

$2 — 3 \to (x_4)\ \ \bar{x}_1x_2\bar{x}_3$ (3); $5 — 8 \to (x_2)\ \ x_1\bar{x}_3x_4$ (8);

$2 — 7 \to (x_1)\ \ x_2\bar{x}_3\bar{x}_4$ (4); $7 — 8 \to (x_2)\ \ x_1x_2\bar{x}_3$ (9).

$3 — 4 \to (x_3)\ \ \bar{x}_1x_2x_4$ (5);

склеиваемые термы 2-го ранга:

$3 — 9 \to (x_1)\ \ x_2\bar{x}_3$;

$4 — 6 \to (x_4)\ \ x_2\bar{x}_3$.

Дальнейшее склеивание невозможно, выписываем простые импликанты: $\bar{x}_1x_3x_4, \bar{x}_2x_3x_4, \bar{x}_1x_2x_4, x_1\bar{x}_2x_4, x_1\bar{x}_3x_4, x_2\bar{x}_3$

Таблица 2.12

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
$\bar{x}_1 x_3 x_4$	*v*			*v*				
$\bar{x}_2 x_3 x_4$	*v*					*v*		
$\bar{x}_1 x_2 x_4$			*v*	*v*				
$x_1 \bar{x}_2 x_4$					*v*	*v*		
$x_1 \bar{x}_3 x_4$					*v*			*v*
$x_2 \bar{x}_3$		*v*	*v*				*v*	*v*

Шаг 2. Составление таблицы. В колонках табл.2.12 записываем термы исходного выражения (они обозначены порядковыми номерами), а в строках - простые импликанты.

Если в исходный терм входит какой-либо импликант, то на пересечении соответствующего столбца и строки ставится метка.

Шаг 3. Нахождение существенных импликант. Если в каком-либо из столбцов таблицы имеется одна метка, то импликанта, соответствующая этой строке, является существенной. Она обязательно входит в конечный результат. Из таблицы для дальнейшего анализа исключаются строки, соответствующие существенным импликантам, и покрываемые ими столбцы.

Шаг 4. Вычеркивание лишних столбцов. Из двух столбцов, имеющих метки в одинаковых строках, один вычеркивается (в нашем примере таких нет).

Шаг 5. Вычеркивание лишних импликант. Если после шага 4 в таблице появятся строки, в которых нет ни одной метки, то импликанты, соответствующие этим строкам, из рассмотрения исключаются (в нашем примере нет). Результате выполнения описанных выше действий приведены в табл.2.13.

Таблица 2.13.

		1	*4*	*5*	*6*
1	$\bar{x}_1 x_3 x_4$		*v*		
2	$\bar{x}_2 x_3 x_4$	*v*			*v*
3	$\bar{x}_1 x_2 x_4$		*v*		
4	$x_1 \bar{x}_2 x_4$			*v*	*v*
5	$x_1 \bar{x}_3 x_4$			*v*	

Шаг 6. Выбор конечного результата. В результирующее выражение включается совокупность импликант, которые имеют метки во всех столбцах. Предпочтение отдается тому варианту, в котором минимально суммарное число литер (букв), входящих в конечный результат.

Выбирая *1-ю* и *4-ю* импликанты, которые в совокупности покрывают все столбцы, окончательно получаем:

$$f(x_1, x_2, x_3, x_4) = x_2\bar{x}_3 \vee \bar{x}_1 x_3 x_4 \vee x_1\bar{x}_2 x_4 .$$

Метод Мак Класки. Мак Класки предложил модернизировать метод Квайна следующим образом:

1) все термы кодируются в виде двоичных последовательностей: переменной соответствует *1*; ее отрицанию – *0*, например, $x_1\bar{x}_2x_3 = 101$;

2) все последовательности разбиваются на группы по числу единиц в последовательности (в i-ю группу попадает терм, который имеет i единиц);

3) при сравнении сопоставляются только соседние группы, т.к. только они имеют отличие в одном разряде;

4) при исключении переменной вместо нее записывается прочерк.

Порядок формирования минимизированного выражения логической функции такой же, как в методе Квайна.

2.4.4. Метод карт Карно

Карта Карно – это графическое представление таблицы истинности. Она представляет собой совокупность ячеек, определяемых системой вертикальных и горизонтальных координат; каждой ячейке соответствует набор значений входных переменных. Запись в ячейке – это значение функции на соответствующем наборе.

Таким образом, имеется взаимно однозначное соответствие между ячейками Карно и строками таблицы истинности.

Пример. Функции $f(x, y, z) = \bar{x}(\bar{y} \vee \bar{z}) \vee xz$ соответствуют представленные ниже таблица истинности и карта Карно.

Карта Карно

x \ yz	00	01	11	10
0	1	1	0	1
1	0	1	1	0

Таблица истинности

x	y	z	$f(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	1
1	0	0	0
1	0	1	1
0	1	1	0
1	1	0	0
1	1	1	1

Сущность метода заключается в выделении в карте Карно прямоугольных ячеек (блоков), содержащих одно и то же значение функции. Любой блок может иметь размер $2^a \times 2^b$, где a, b – целые.

Правила формирования выражения минимальной булевой функции. Основные правила заключаются в следующем:

1) каждая ячейка, содержащая единицу, должна войти в какой-то из блоков;

2) результат записывается в виде логической суммы термов, соответствующих сформированным блокам;

3) терм, описывающий блок, записывается в форме произведения тех переменных, которые на координатной сетке не изменяют свое значение в пределах блока; если координата равна нулю, то соответствующая ей переменная входит в терм с отрицанием, а если - единице, то без отрицания.

Степень сложности булевой функции оценивается числом слагаемых и числом букв (литер). Выражение, которое получено с применением метода карт Карно, на основе термов, сформированных из групп с единичным значением логической функции, при минимальном количестве литер называется ***минимальной суммой***.

При формировании групп для получения минимальных сумм необходимо руководствоваться двумя принципами:

1) группа должна быть как можно больше;

2) число групп должно быть как можно меньше.

Карту Карно следует рассматривать как трехмерную, представляя ее в виде тора (склеивая правую и левую, а так же нижнюю и верхнюю границы).

Пример. Карте Карно, представленной на рис. 2.1, соответствует функция $f(w, x, y, z) = x\bar{z} \vee \bar{x}yz$.

Общая последовательность действий при минимизации:

1) в таблице Карно выбирается ячейка с единицей, не являющаяся подмножеством уже сформированного блока;

2) формируется наибольшая группа, содержащая эту ячейку;

3) выбирается следующая ячейка, не вошедшая в предшествующие группы, и для нее повторяются те же действия;

4) процесс повторяется, пока не останутся единицы, не включенные в какие-либо группы;

5) записывается выражение минимальной функции по правилу, которое было установлено выше.

Пример. Карте Карно, представленной на рис. 2.2, соответствует функция $f(w, x, y, z) = \bar{x}\,\bar{y} \vee w\,y\,\bar{z} \vee \bar{x}\,\bar{z}$ **.**

Существует еще один способ записи минимальной формулы на основе метода минимальных произведений. При этом по тому же правилу объединяются ячейки с нулем, логическая формула записывается как конъюнкция элементарных дизъюнкций, а правила формирования переменных в дизъюнкции обратны описанным выше.

Минимизация частично определенных булевых функций. В случае, если некоторые входные наборы невозможны либо значение функции на них несущественно, то говорят о недоопределенных условиях. При этом особенность использования карт Карно следующая: недоопределенное условие на карте Карно обозначается прочерком и его можно либо присоединить, либо не присоединить к любой группе.

Пример. Карте Карно, представленной на рис. 2.3, соответствует функция $f(w, x, y, z) = y\bar{z} \vee \overline{w}x\bar{y}$.

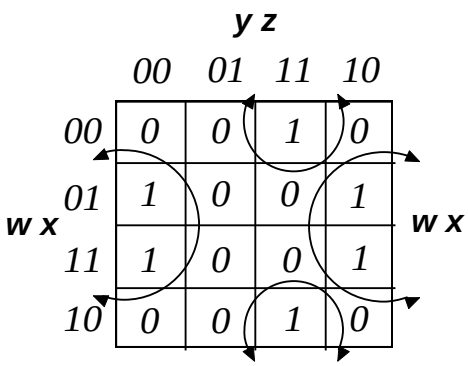


Рис.2.1

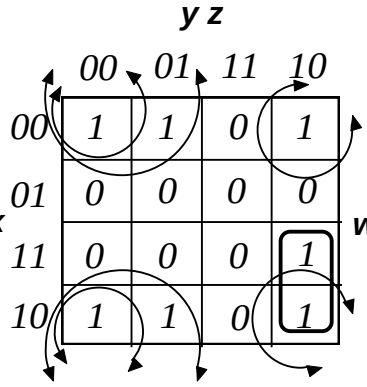


Рис.2.2

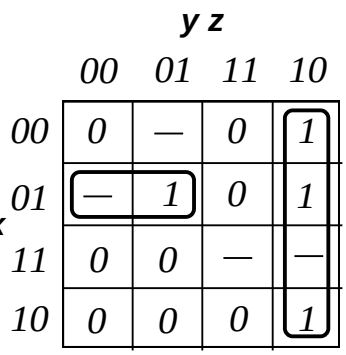


Рис.2.3

2.5. Полнота и замкнутость множества Булевых функций

2.5.1. Понятие функционально полной системы

Система функций $\{f_1, f_2, ..., f_n\}$ называется ***функционально полной***, если любая булева функция может быть записана в виде формул через функции этой системы. Одним из способов получения полных систем является использование некоторых существующих, ранее выявленных полных систем.

Теорема. Пусть даны две системы функций: $B = \{f_1, f_2, ..., f_n\}$ и $D = \{q_1, q_2, ..., q_m\}$. Относительно этих функций известно, что B – полная система и каждая ее функция может быть выражена в виде формул через функции второй системы. В этом случае система D – тоже полная.

Доказательство. Пусть h – это произвольная функция. В силу полноты B функция h может быть выражена в виде формул над B, т.е. $h = F(f_1, f_2, ..., f_n)$. Но любая из функций B по условию может быть выражена через функции D, т.е. $f_1 = \varphi_1(q_1, q_2, ...), f_2 = \varphi_2(q_1, q_2, ...)$.

Следовательно,

$h = F(\varphi_1(q_1, q_2, ...), \varphi_2(q_1, q_2, ...)) = F(q_1, q_2, ..., q_n)$.

Таким образом, произвольная функция h может быть выражена через функции системы D. Поэтому система D – полная.

Доказанную теорему можно использовать для доказательства полноты двух следующих систем: $S_1 = \{\wedge, \bar{}\}$ и $S_2 = \{\vee, \bar{}\}$. Известно, что $S_0 = \{\wedge, \vee, \bar{}\}$ – полная система (это вытекает из того факта, что любую булеву функцию можно представить в КНФ и ДНФ).

Для доказательства полноты S_1 и S_2 нужно установить, что недостающая относительно S_0 операция (в S_1 – дизъюнкция, в S_2 – конъюнкция) может быть выражена через две остальные. Такое подтверждение дают правила де Моргана и двойного отрицания: $x_1 \vee x_2 = \overline{\bar{x}_1 \cdot \bar{x}_2}$, $x_1 \wedge x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}$. Следовательно, S_1 и S_2 — полные системы и их называют соответственно ***конюнктивным и дизъюнктивным базисом Буля***.

Полной является также система $S_3 = \{\wedge, \oplus, 1\}$, которая сводится к конъюнктивному базису, т.к. отрицание в S_1 может быть выражено через операции S_3 следующим образом: $\bar{x} = x \oplus 1$ (проверяемся по таблице истинности функции f_6, см. табл.2.8).

2.5.2. Алгебра Жегалкина

Алгебра над множеством логических функций с двумя бинарными операциями (конъюнкция и сложение по модулю два) называется ***алгеброй Жегалкина***. В этой алгебре выполняются следующие соотношения:

1) $x \oplus y = y \oplus x$;
2) $x(y \oplus z) = xy \oplus xz$;
3) $x \oplus x = 0$;
4) $x \oplus 0 = x$;
5) $\bar{x} = x \oplus 1$.

Для подтверждения полноты алгебры Жегалкина представим дизъюнкцию через функции этой алгебры: $x \vee y = \overline{\bar{x} \cdot \bar{y}} = xy \oplus x \oplus y$.

Полученная формула, имеющая вид суммы произведений, называется ***полиномом по модулю два*** или ***полиномом Жегалкина***. Для каждой логической функции может быть получен полином Жегалкина, причем единственный для данной функции. Для этого следует привести заданное выражение к

форме СДНФ, исключить операции дизъюнкции, отрицания и раскрыть скобки. В частном случае, если $x \cdot y = 0$, то $x \vee y = x \oplus y$.

Пример. $x_1x_2 \vee \bar{x}_1\bar{x}_2 = x_1x_2 \oplus \bar{x}_1\bar{x}_2 = x_1x_2 \oplus (x_1 \oplus 1)(x_2 \oplus 1) =$
$= x_1x_2 \oplus x_1x_2 \oplus x_1 \oplus x_2 \oplus 1 = x_1 \oplus x_2 \oplus 1.$

Существует ещё один способ приведения функции к полиному Жегалкина. Запишем предполагаемый результат в виде выражения общего вида с неопределенными коэффициентами.

Пример. $x_1x_2 \vee \bar{x}_1\bar{x}_2 = ax_1x_2 \oplus bx_1 \oplus cx_2 \oplus d.$

Для определения значений коэффициентов вычислим левую и правую часть для четырех возможных наборов входных переменных:

1) $x_1 = 0, x_2 = 0$ $\quad 1 = d;$ $\quad d = 1;$
2) $x_1 = 0, x_2 = 1$ $\quad 0 = c \oplus d = c \oplus 1;$ $\quad c = 1;$
3) $x_1 = 1, x_2 = 0$ $\quad 0 = b \oplus d = b \oplus 1;$ $\quad b = 1;$
4) $x_1 = 1, x_2 = 1$ $\quad 1 = a \oplus b \oplus c \oplus d = a \oplus 1 \oplus 1 \oplus 1;$ $\quad a = 0.$

Отсюда. $x_1x_2 \vee \bar{x}_1\bar{x}_2 = x_1 \oplus x_2 \oplus 1.$

2.5.3. Замыкание и замкнутые классы

Замыканием множества функции M называют множество всех булевых функций, представляемых в виде формул через функции множества M. Замыкание множества M будем обозначать $[M]$. Очевидно, что $M \subseteq [M]$.

Множество M называют ***замкнутым классом***, если при замыкании M не происходит его дальнейшее расширение, то есть $[M] = M$.

Ответ на вопрос, какую систему булевых функций можно считать функционально полной, дают две теоремы.

Теорема о функциональной полноте. Для того чтобы система функций B была полной, необходимо и достаточно, чтобы она не содержалась полностью ни в одном из пяти замкнутых классов: T_0, T_1, S, M, L, где T_0 – класс замкнутых булевых функций, сохраняющих константу 0, т.е. функций, для которых $f(0,0,...,0) = 0$

T_1 – замкнутый класс булевых функций, сохраняющих константу *1*, т.е. таких функций, что $f(1,1,...,1)=1$;

S – замкнутый класс всех самодвойственных функций, т.е. функций, для которых $\overline{f}(\bar{x}_1,\bar{x}_2,...,\bar{x}_n)=f(x_1,x_2,...,x_n)$;

M – замкнутый класс всех монотонных функций;

L – замкнутый класс всех линейных функций.

Функция $f(x_1, x_2, ..., x_n)$ называется *монотонной*, если из условия $\alpha \le \beta$ следует $f(\alpha) \le f(\beta)$, где α и β – наборы переменных $\alpha=(\alpha_1,\alpha_2,...,\alpha_n)$, $\beta=(\beta_1,\beta_2,...,\beta_n)$ и введено отношение порядка $\le$, означающее $\alpha \le \beta$, если $\alpha_1 \le \beta_1, \alpha_2 \le \beta_2, ..., \alpha_n \le \beta_n$.

Линейной функцией называется функция, у которой полином Жегалкина имеет вид $c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus ..... \oplus c_ix_i \oplus ..... \oplus c_nx_n$, где c_0 и c_i – коэффициенты (равны *1* или *0*). В линейной функции отсутствует произведение переменных.

Существует еще одно, близкое по содержанию определение теоремы о полноте.

Теорема Поста. Система $B=\{f_1, f_2, ..., f_n\}$ является ***полной*** тогда, когда выполняются следующие условия:

1) $f_i \notin T_0$;
2) $f_j \notin T_1$;
3) $f_k \notin S$;
4) $f_m \notin M$;
5) $f_l \notin L$.

То есть хотя бы одна функция не должна принадлежать какому-либо замкнутому классу, т.к. из функций замкнутого класса нельзя построить функцию, не входящую в данный класс.

Примеры функционально полных базисов. Класс N функций из E_2 называется ***предполным,*** если он неполный, а для любой функции f, такой, что $f \in E_2$, но $f \notin N$, класс $N \cup \{f\}$ – полный.

Система B называется ***полной,*** если любая функция $f \in E_2$ может быть представлена в виде суперпозиций функций этой системы, и она называется ***базисом,*** если теряется полнота при удалении хотя бы одной функции.

Состав систем, которые относятся к базису, можно определить на основе табл. 2.14.

Таблица 2.14

N	Функция	T_0	T_1	L	S	M
1	$x_1 \wedge x_2$	—	—	+	+	—
2	$x_1 \oplus x_2$	—	+	—	+	+
3	$x_1 \vee x_2$	—	—	+	+	—
4	$x_1 \sim x_2$	+	—	—	+	+
5	$\bar{x}$	+	+	—	—	+
6	$x_1 \to x_2$	+	—	+	+	+
7	$x_1 \mid x_2$	+	+	+	+	+
8	1	+	—	—	+	—

Символом «+» обозначается факт непринадлежности функции к замкнутому классу.

Согласно теореме Поста, к базису можно отнести минимальный набор функций, строки которого будут содержать хотя бы один символ “+” в каждом столбце (T_0, T_1, S, M, L).

Из табл.2.14 видно, что к функционально полным системам можно отнести, например, системы $\{\wedge, \bar{}\}, \{\mid\}, \{\oplus, \to\}$.

3. МАТЕМАТИЧЕСКАЯ ЛОГИКА

3.1. Общие сведения о формальных и аксиоматических системах

Во введении уже говорилось, что в основе любой логики лежит определенная система, задающая правила формирования логических выражений, используемые механизмы построения рассуждений и т.п. Системы такого рода называются формальными. Рассмотрим общие положения, которые лежат в основе построения таких систем.

Определение. ***Формальная система*** представляет собой совокупность чисто абстрактных объектов, не связанных с внешним миром, в которой представлены правила оперирования множеством символов только в синтаксической трактовке без учета смыслового содержания.

Всякая формальная система строится на основе формализованного языка (как средства формирования и изложения выражений, имеющих смысл в данной теории) и совокупности теорем. Так же, как в естественном языке, средством выражения мысли является предложение, построенное по определенным правилам, в математике средством выражения является формула, построенная из заданного набора символов. В формальной теории все формулы доказываются.

Под теоремой в формальной системе понимают высказывание, истинное в данной системе, – это некоторое обоснованное и строгое утверждение, которое строится на основе определенных логических правил и доказательства. Доказательство – это способ получения одних выражений из других с помощью операций над символами и построения обоснованной аргументации, следствием которой и является теорема.

При построении любой формальной теории (системы) в качестве исходных посылок всегда используются некоторые неопределяемые термины и аксиомы. Неопределяемые термины – это те термины и понятия, смысл и содержание которых считается уже известным, и через них вводятся все новые

понятия и термины. Совершенно аналогично вводится некоторая часть постулатов (формул), которые, как считается в данной теории, не требуют доказательства. Обычно это утверждения, правильность которых не вызывает сомнения, и они принимаются как очевидные истины. Такие выражения (формулы) называют аксиомами, а системы, в основе построения которых лежит использование аксиом, называются аксиоматическими системами.

Формирование строгой формальной теории осуществляется в следующем порядке:

1. Задается конечное множество символов, которые образуют алфавит формальной системы.
2. Устанавливаются процедуры построения формул формальной системы.
3. Устанавливается множество аксиом, т.е. формул, истинность которых не требует доказательства. Обычно к ним относят те утверждения, которые полагаются очевидными по самой природе рассматриваемых понятий.
4. Устанавливается конечное множество правил вывода, которые позволяют получать новые формулы из некоторого множества известных формул. В общем случае эти правила могут быть представлены в следующем виде $H_1 \& H_2 \& \ldots \& H_m \to M$, что означает: из множества истинных формул $H_1, H_2, \ldots, H_m$, указанных в левой части выражения, следует истинность формул правой части выражения. Говорят, что формула *M*, полученная в результате применения правила вывода, выводима в данной теории. Таким образом, правила вывода позволяют, последовательно применяя их, получать новые истинные формулы или теоремы из аксиом и ранее выведенных формул.

Теперь с учетом понятия правила вывода уточним понятия формального доказательства и теоремы.

Определение. ***Формальным доказательством***, или просто ***доказательством,*** называется последовательность формул $M_1, M_2, \ldots, M_n$ такая, что каждая формула M_i является либо аксиомой, либо выводима из предшествующих ей формул.

Определение. Формула t называется ***теоремой***, если существует доказательство, в котором она является последней.

Задаваемые при описании формальной системы правила вывода называют также ***правилами вывода заключений***, т.е. они позволяют определить, является ли данная формула теоремой данной формальной системы или нет.

Различают два типа правил вывода.

1. Правила, применяемые к формулам, рассматриваемым как единое целое, в этом случае их называют ***продукционными правилами***.

Пример. $x < y$ и $y < z \rightarrow x < z$ – это продукция с двумя посылками.

2. Правила, которые могут применяться к любой отдельной части формулы, причем сами эти части являются формулами, входящими в состав формальной системы. В этом случае их называют ***правилами переписывания***.

Пример. $x - x = 0$, это выражение имеет смысл при любом значении входящей в него в качестве подвыражения переменной x.

Определение. ***Правило подстановки*** заключается в замещении всех вхождений какой-либо переменной на формулу из формальной системы, которая не содержит этой переменной.

Пример. Рассмотрим формальную систему следующего вида:

1. Алфавит = $\{a, b, w\}$.
2. Формулы – символ или последовательность символов a, b или w.
3. Аксиома awa.
4. Правило вывода: $c_1 w c_2 \rightarrow b c_1 w c_2 b$ (продукция).

Символы c_1 и c_2 не принадлежат алфавиту формальной системы (ФС), они служат посредниками для формализации правил вывода. То есть c_1 и c_2 обозначают какие-либо последовательности символов a или b формальной системы и могут быть замещены любыми последовательностями символов a или b.

Учитывая способ образования формул, можно сказать, что незамещаемые символы a и b называются константами, а символ w - оператором.

Из определения ФС вытекает и способ получения допустимых формул, т.е. формул, выводимых согласно правилу вывода путем его последовательного применения к аксиоме:

awa
bawab
bbawabb
bbbawabbb и т.д.

Определение. Формальная система называется ***разрешимой,*** если существует хорошо определенный способ действия, который за конечное число шагов для любой формулы формальной системы позволяет определить – выводима она в данной системе или нет. Сам способ действий, если он существует, называют ***процедурой разрешения***.

Так как формальные системы всегда представляют собой модель какой-то реальности, то возникает вопрос об установлении взаимосвязи между объектами формальной системы и этой реальностью, для этого вводится понятие интерпретации.

Определение. ***Интерпретация*** представляет собой распространение исходных положений какой-либо формальной системы на реальный мир. Интерпретация придает смысл каждому символу формальной системы и устанавливает взаимно однозначное соответствие между символами формальной системы и реальными объектами. Теоремы формальной системы, будучи интерпретированы, становятся после этого утверждениями в обычном смысле слова, и в этом случае уже можно делать выводы об их истинности или ложности.

Следует отметить, что при интерпретации речь идет о замыкании или логическом завершении математического подхода, который в общем случае можно описать в виде следующей последовательности действий:

- в начале математик изучает реальность, конструируя некоторое абстрактное представление о ней, т.е. некоторую формальную систему;
- затем строится доказательство теорем формальной системы. Вся польза и удобства формальных систем заключаются

в их абстрагировании от конкретной реальности. Благодаря этому одна и та же формальная система может служить моделью многочисленных различных конкретных ситуаций;

- происходит возвращение к начальной точке всего построения и осуществляется интерпретация теорем, полученных при формализации.

Замечание. Изучение аксиом и теорем как абстрактных выражений, представленных в некоторой форме, называется *синтаксическим изучением* ***аксиоматических систем*** (АС*)*; изучение и рассмотрение смысла этих выражений называется *семантическим изучением АС*. Отметим, что с синтаксическим аспектом АС и связано понятие формальной системы.

Формальную теорию часто называют исчислением. Под исчислением понимают формальное представление теории, которое позволяет оперировать с объектами без учета формального смысла выражений.

В рамках создания формальной системы, как правило, решаются (должны быть решены) следующие проблемы:

1. **Проблема противоречивости**. Логическое исчисление называется *непротиворечивым*, если в нем недоказуемы никакие две формулы, из которых одна является отрицанием другой.

2. **Проблема полноты.** Система исчисления считается полной, если любая теорема теории может быть доказана или опровергнута.

3. **Проблема независимости аксиом.** Для начала введем понятие независимой аксиомы. Аксиома называется *независимой*, если она не может быть выведена из других аксиом. Система аксиом исчисления называется *независимой*, если каждая аксиома в ней независима.

4. **Проблема разрешимости.** Система исчисления называется разрешимой, если существует алгоритм, позволяющий по каждому утверждению выяснить, является ли оно истинным или ложным.

3.2. Исчисление высказываний

Определение. ***Исчисление высказываний*** (ИВ), т.е. логика высказываний – это формальная система, интерпретацией

которой является алгебра высказываний. Под **высказыванием** понимается **повествовательное** предложение, о котором можно сказать, что оно истинно или ложно. Основной задачей исчисления высказываний является порождение общелогических законов – тождественно истинных высказываний, то есть высказываний (в том числе составных), которые всегда истинны независимо от входящих в них элементарных высказываний.

Как и любая формальная система, исчисление высказываний строится на основе четырех основных процедур: задания алфавита, установления правил построения формул, аксиом и правил вывода. Определим все эти четыре компонента:

1. Алфавит, который состоит из символов трех категорий:

а) бесконечное счетное множество высказываний (или переменных высказываний), которые обычно обозначаются буквами: *x, y, z, a, b, c,* x_1, x_2 и т.д.;

б) логические операторы (или логические связки), которые обозначают символы логических операций ($\vee$, &, $\rightarrow$ и т.д.);

в) открывающиеся и закрывающиеся скобки: (,) .

Других символов в ИВ нет.

2. Правила построения формул. Для обозначения формул обычно используют заглавные буквы латинского алфавита. Эти буквы не являются символами исчисления и служат для условного обозначения формул. Формулы в ИВ определяются следующим образом.

Формулы в исчислении высказываний однозначно получаются с помощью правил, которые описываются базисом и индуктивным шагом:

базис: всякое высказывание есть формула;

индуктивный шаг: если X и Y – формулы, то $\overline{X}$, $(X \vee Y)$, $(X \rightarrow Y)$, $(X \,\&\, Y)$ и т.д. – также формулы.

Никакая другая последовательность символов не является формулой.

Пример. Если *x, y, z* – формулы в соответствии с правилом базиса, то $(x \rightarrow y)$, $(x \,\&\, z)$ – формулы в соответствии с правилом индуктивного шага. Очевидно, что не будут формулами: *&x,* $(x \vee z$, т.к. они не удовлетворяют вышеуказанным правилам (в

первом случае в бинарной операции используется один операнд, а во втором – отсутствует закрывающая скобка).

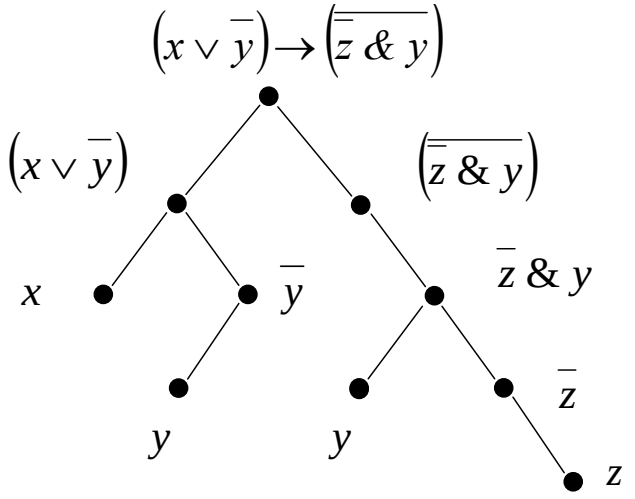
С введением понятия формулы вводится и понятие *подформулы* или части формулы, делается это следующим образом.

1. Подформулой элементарной формулы является только она сама.
2. Если $\overline{X}$ – формула, то ее подформулами будут: она сама, X и все подформулы X.
3. Обозначим w – любую из логических операций $(\vee, \&, \rightarrow)$, кроме отрицания. Тогда если (XwY) – формула, то ее подформулы: она сама, формулы X и Y и все подформулы X и Y.

Пример. Пусть задана формула $\left(x \vee \overline{y}\right) \rightarrow \left(\overline{\overline{z} \,\&\, y}\right)$, определим ее подформулы и глубину их вложенности.

Подформула	Глубина
$\left(x \vee \overline{y}\right) \rightarrow \left(\overline{\overline{z} \,\&\, y}\right)$	0
$\left(x \vee \overline{y}\right),\ \left(\overline{\overline{z} \,\&\, y}\right)$	1
$x, \overline{y}, \left(\overline{z} \,\&\, y\right)$	2
$y, \overline{z}$	3
z	4

Кроме табличной формы каждая правильная формула может быть представлена в виде дерева, ветви которого – исходные и



промежуточные формулы:

Из примера видно, что на самом низком уровне находятся только элементарные формулы, однако они могут находиться и на более высоких уровнях.

Для упрощения записи формул ИВ используются те же соглашения, что и в алгебре логики, которые позволяют учитывать приоритеты операций и избавиться от излишних скобок.

Пример. $((X \,\&\, Y) \vee C) = X \,\&\, Y \vee C$;

$\overline{(X \vee Y)} = \overline{X} \vee \overline{Y}$.

3. Аксиомы. Существует несколько вариантов подбора аксиом как исходных тождественно истинных формул. Эти наборы эквивалентны в том смысле, что они определяют один и тот же класс выводимых формул. Один из наиболее часто применяемых вариантов включает 11 аксиом:

1. $x \to (y \to x)$.
2. $(x \to (y \to z)) \to ((x \to y) \to (x \to z))$.
3. $x \,\&\, y \to x$.
4. $x \,\&\, y \to y$.
5. $(z \to x) \to ((z \to y) \to (z \to x \,\&\, y))$.
6. $x \to x \vee y$.
7. $y \to x \vee y$.
8. $(x \to z) \to ((y \to z) \to (x \vee y \to z))$.
9. $(x \to y) \to (\bar{y} \to \bar{x})$.
10. $x \to \bar{\bar{x}}$.
11. $\bar{\bar{x}} \to x$.

Тождественную истинность аксиом можно проверить либо прямым вычислением значения формулы на каждом наборе, либо приведением их к константе 1 путем эквивалентных преобразований, применяемых в булевой алгебре. Например, докажем методом эквивалентных преобразований истинность формулы (2):

$$(x \to (y \to z)) \to ((x \to y) \to (x \to z)) = \overline{(\bar{x} \vee \bar{y} \vee z)} \vee (\overline{\bar{x} \vee y} \vee \bar{x} \vee z) =$$
$$= xy\bar{z} \vee x\bar{y} \vee \bar{x} \vee z = xy\bar{z} \vee \bar{y} \vee \bar{x} \vee z = x\bar{z} \vee \bar{y} \vee \bar{x} \vee z = \bar{z} \vee \bar{y} \vee \bar{x} \vee z =$$
$$= \bar{y} \vee \bar{x} \vee 1 = 1.$$

4. Правила вывода. Правила вывода устанавливают отношения на множестве формул исчисления высказываний. Правила вывода обычно представляются как отношения на множестве формул исчисления высказываний. Над чертой записываются формулы, которые играют роль посылки (уже известные истинные выражения), а под чертой – выводимая формула, истинность которой утверждается данным правилом. Она называется следствием или заключением.

В исчислении высказываний используется два правила вывода:

1) *правило заключения (modus ponens).* Если A и $A \to B$ – это выводимые формулы, то B также является выводимой формулой. Записывается это правило так:

$$\frac{A,\ A \to B}{B};$$

2) *правило подстановки.* Его формальная запись имеет вид

$$\frac{A}{A(x_1, x_2, \ldots, x_n \| B_1, B_2, \ldots, B_n)},$$

где $A, B_1, B_2, \ldots, B_n$ - это формулы, $x_1, x_2, \ldots, x_n$ – попарно различные переменные высказывания; через запись $A(x_1, x_2, \ldots, x_n \| B_1, B_2, \ldots, B_n)$ обозначен результат одновременной замены всех вхождений переменных $x_1, x_2, \ldots, x_n$ в A на формулы $B_1, B_2, \ldots, B_n$.

Справедливость правил вывода исчисления высказываний подтверждается применением методов булевой алгебры.

В частности, тождественную истинность выводимой формулы B можно установить следующим образом: если A и $A \to B$ – тождественно истинные формулы, то есть $A = 1$ и $A \to B = 1$, следовательно $\bar{A} \vee B = 1$. Так как в последнем выражении $\bar{A} = 0$, а значение логической суммы равно 1, то B должно быть равно 1 (то есть быть истинным).

Правило подстановки сохраняет тождественную истинность выражения, так как любая подстановка в тождественно истинную формулу также дает тождественно истинную формулу.

Используя два приведенных правила, можно формально порождать новые формулы без учета семантического смысла используемых выражений.

Кроме двух приведенных выше правил вывода, можно получить и другие правила, позволяющие строить новые доказуемые формулы. Но так как они реализуются с помощью правила подстановки и заключения, то они получили название *производных правил* вывода. Наиболее распространенные из них следующие:

Правило сложного заключения. Если $A_1, A_2, \ldots, A_n$ – формулы и $A_1 \to (A_2 \to (A_3 \to (\ldots (A_m \to B)\ldots)))$ – теорема, то формула B – также теорема.

Правило двойного отрицания. Если $A \to \overline{\overline{B}}$ и $\overline{\overline{A}} \to B$ – теоремы, то будет теоремой и формула $A \to B$, иначе $\dfrac{A \to \overline{\overline{B}}}{A \to B}$ и $\dfrac{\overline{\overline{A}} \to B}{A \to B}$.

Правило силлогизма (замыкания). Если $A \to B$ и $B \to C$ – теоремы, то $A \to C$ также теорема, иначе $\dfrac{A \to B,\ B \to C}{A \to C}$.

Правило композиции. Если $A \to B$ теорема, то $\overline{B} \to \overline{A}$ так же теорема, иначе $\dfrac{A \to B}{\overline{\overline{B}} \to \overline{\overline{A}}}$.

Правила вывода можно рассматривать и как результат логического анализа некоторых человеческих рассуждений. Например, рассмотрим примеры для приведенных выше правил.

1. Правилу заключения соответствует следующая схема рассуждений:

ИСХОДНЫЕ ПОСЫЛКИ. Если данный многоугольник правильный (*A=1*), то в него можно вписать окружность (*A* → *B*). Возьмем правильный многоугольник (*A=1*).

ВЫВОД. В данный многоугольник можно вписать окружность (*B = 1*).

2. Интерпретацией правила силлогизма может быть следующая схема рассуждений:

ИСХОДНЫЕ ПОСЫЛКИ. Если треугольник равнобедренный (*A = 1*), то две его стороны равны (*B = 1*). Если две стороны треугольника равны (*B = 1*), то два его угла равны С = 1).

ВЫВОД. Если треугольник равнобедренный, то два его угла равны (*A*→*C*).

3. Правило композиции иллюстрируется следующей схемой рассуждений:

ИСХОДНАЯ ПОСЫЛКА. Если число рациональное (*A = 1*), то оно может быть представлено в виде отношения двух целых чисел (*B = 1*).

ВЫВОД. Если число не может быть представлено в виде отношения двух целых чисел, то оно нерациональное ($\overline{B} \to \overline{A}$).

Как отмечалось выше, формулы исчисления высказываний можно интерпретировать как формулы алгебры высказываний (АВ). Для этого следует переменные ИВ трактовать как переменные АВ, т.е. переменные, которые могут принимать только значение 0 и 1 (ложь и истина). Все логические операции ($\vee, \&, \to$ и т.д.), которые используются в ИВ, интерпретируются так же, как и в АВ, т.е. на основе тех же самых таблиц истинности. Очевидно, что между формулами ИВ и АВ существует взаимнооднозначное соответствие. Однако формализма, реализованного в АВ, не всегда достаточно для реализации построения доказательств в ИВ, поэтому существует и множество других методов.

Перед рассмотрением методов установления факта общезначимости формул рассмотрим используемые в исчислении высказываний следующие термины и определения:

1. Формула ***выполнима***, если она может принимать значение «истина» (например, $p \& q,\ \overline{p},\ p \vee q$) .

2. Формула ***невыполнима***, если ни при каких значениях составляющих ее высказываний она не может быть истинной (например, $p \& \overline{p}$).

3. Формула ***общезначима***, если она принимает значение «истина» независимо от истинности ее составляющих (например, $p \vee \overline{p}$).

4. Формула ***нейтральна***, если она не общезначима и не является невыполнимой.

5. ***Тавтологиями*** называются общезначимые формулы. Если формула $A \equiv 1$, т.е. A – тавтология, то это можно записать так $\models A$.

Логический вывод на основе множества гипотез можно определить следующим образом.

Пусть E – это множество формул, тогда запись $E \models A$ означает, что если все формулы из E истинны, то будет истинной формула A. В этом случае A – называется логическим следствием из E.

Если $E = \{H_1, H_2, \ldots, H_n\}$ и $E \models A$, то можно записать $\{H_1, H_2, \ldots, H_n\} \models A$. Формулы H_i называются *гипотезами*, а формула A – *заключением*.

Определение. ***Принцип дедукции*** состоит в следующем. Формула A является логическим следствием конечного множества E тогда и только тогда, когда $E \cup \{\overline{A}\}$ содержит невыполнимые формулы.

В силу того, что для высказываний справедливы все свойства логических операций, которые были определены в алгебре логики, то, используя законы де Моргана, можно ввести понятие ***прямой и обратной дедукции.***

Действительно, если A есть логическое следствие гипотез $H_1, \ldots, H_n$, то, учитывая сформулированный выше принцип дедукции, можно считать справедливой следующую запись: $H_1 \,\&\, H_2 \,\&\, \ldots \,\&\, H_n \,\&\, \overline{A} = 0$. Это правило называется *правилом прямой дедукции.* Возьмем отрицание от этого выражения, тогда по правилу де Моргана получим $\overline{H}_1 \vee \overline{H}_2 \vee \ldots \vee \overline{H}_n \vee A = 1$. Это правило называется *правилом обратной дедукции.*

Т.к. для распознавания выполнимости и общезначимости формул существует множество различных методов, рассмотрим некоторые из них.

Замечание. Следует учитывать, что не всегда алгоритм дедукции оказывается более эффективным по сравнению с тривиальным подходом, который основывается на

алгебраическом (использующим эквивалентные преобразования из АВ) представлении вывода.

3.3. Методы, используемые для определения общезначимости формул исчисления высказываний

3.3.1. Алгоритм редукции

Этот алгоритм позволяет доказывать общезначимость формул исчисления высказываний путем приведения к абсурду. Рассмотрим на примере. Пусть требуется доказать общезначимость следующей формулы:

$$((p \,\&\, q) \to r) \to (p \to (q \to r)).$$

Предположим, что при некоторой интерпретации эта формула принимает значение «ложь». Из определения функции импликации известно, что значение «ложь» она принимает только в том случае, когда посылка истинна, а заключение ложно. Учитывая указанное свойство, представляем исходную формулу в виде двух интерпретаций следующего вида:

$$p \to (q \to r) = 0; \qquad (1.1)$$

$$(p \,\&\, q) \to r = 1. \qquad (1.2)$$

Применив ранее использованные рассуждения к строке (1.1), получим следующие значения переменных: $p = 1, q = 1, r = 0.$ Если подставить полученные значения в строку (1.2), то получится противоречие. Значит, предположение о том, что существует некоторая интерпретация, при которой исходная формула принимает значение «ложь», неверно, и это означает общезначимость исходной формулы.

Пример. Используя алгоритм редукции, доказать общезначимость следующей формулы: $(a \to b) \to ((c \to a) \to (c \to b))$.

Пусть при некоторой интерпретации

$(a \to b) \to ((c \to a) \to (c \to b)) = 0$.

Это возможно только, если

$$a \to b = 1; \qquad (1.3)$$

$$(c \to a) \to (c \to b) = 0. \qquad (1.4)$$

Тогда из (1.3) следует, что возможна одна из следующих комбинаций значений переменных a и b:

$$a=0 \quad b=0;$$
$$a=1 \quad b=1;$$
$$a=0 \quad b=1.$$

Из (1.4) следует $(c \to a)=1$, это означает, что возможны следующие значения c и a:

$$c=0 \quad a=0;$$
$$c=1 \quad a=1;$$
$$c=0 \quad a=1.$$

Из $c \to b=0$ имеем $c=1, \quad b=0$. Это единственно допустимые для c и b значения, при которых формула принимает значение «ложь».

Сопоставляем полученные результаты с ранее рассмотренными возможными значениями переменных. Оказывается, что при $c=1$ единственное допустимое значение для а – это $a=1$, а при $b=0$ единственное допустимое значение $a=0$. То есть переменная а должна принимать взаимно исключающие значения, что невозможно. Следовательно, предположение о существовании интерпретации, при которой формула $(a \to b) \to ((c \to a) \to (c \to b))$ принимает значение «ложь» неверно и это означает ее общезначимость.

3.3.2. Метод резолюций

Для порождения логических следствий будет использована очень простая схема рассуждений. Пусть A, B и X – формулы. Предположим, что две формулы $(A \vee X) и (B \vee \overline{X})$ истинны. Тогда, если X истина, то отсюда следует, что B тоже истина. Наоборот, если X – ложь, то следует, что A – истина. Получаем правило $\{A \vee X, B \vee \overline{X}\} \models (A \vee B)$, которое можно записать в виде $\{\overline{X} \to A, X \to B\} \models (A \vee B)$.

В том частном случае, когда X – высказывание, а A и B – дизъюнкты, это правило называется ***правилом резолюций.***

Рассмотрим, каким образом это правило может быть использовано для построения доказательства.

В методе резолюций применяется также приведенное выше правило прямой дедукции: для того чтобы доказать, что формула C является логическим следствием из гипотез $H_1,H_2,\ldots,H_n$, следует доказать, что $H_1\&H_2\&\ldots\&H_n\&\overline{C}=0$. Так как левая часть последнего равенства представляет собой конъюнкцию, для его выполнения достаточно доказать равенство нулю любой входящей в уравнение формулы. Таким образом, для доказательства выводимости исходной формулы C необходимо доказать, что в множестве $\{H_1,H_2,\ldots,H_n,\overline{C}\}$ имеется хотя бы одна невыполнимая формула. Для этого каждый элемент указанного множества рассматривается как элементарная дизъюнкция (дизъюнкт). Применение метода резолюций предусматривает порождение новых дизъюнктов на основе следующей леммы, в основе которой лежит правило резолюций.

Лемма. Пусть S_1 и S_2 – дизъюнкты нормальной формы S, а l – литера. Если $l\in S_1$ и $\bar{l}\in S_2$, то дизъюнкт $r=(S_1\setminus\{l\})\cup(S_2\setminus\{\bar{l}\})$ является логическим следствием нормальной формы S.

Следствие. Нормальные формы S и $S\cup\{r\}$ эквивалентны.

Замечание. Дизъюнкт r называется ***резольвентой*** дизъюнктов S_1 и S_2.

Для доказательства приведенных выше утверждений о выполнимости формулы C необходимо, как отмечалось выше, доказать, что хотя бы один дизъюнкт из рассматриваемого множества исходных и порожденных дизъюнктов тождественно равен нулю. В этом случае говорят, что получен пустой дизъюнкт.

Таким образом, принцип резолюций заключается в использовании того факта, что множество дизъюнктов S не выполнимо, если пустой дизъюнкт является логическим следствием из него. То есть для доказательства невыполнимости множества S необходимо строить логические следствия из него до тех пор, пока не будет получен пустой дизъюнкт или дальнейшее построение логических следствий окажется невозможным.

Метод резолюций выгодно отличается от других методов тем, что он дает возможность использовать средства

автоматического доказательства, применяемые в логическом программировании.

Прежде чем перейти к описанию алгоритма напомним несколько терминов, известных из курса дискретной математики, которые будут использованы при описании алгоритма метода резолюций.

Определение. ***Литера*** – это элементарное высказывание или его отрицание. Например, $x, \bar{y}, z$.

Определение. ***Дизъюнкт*** или ***элементарная дизъюнкция*** – это совокупность различных литер, связанных дизъюнктивной связью. Например, $x \vee \bar{y}, \bar{x} \vee y \vee \bar{z}, x$.

Определение. ***Конъюнктивной нормальной формой*** (КНФ) некоторой формулы называется равносильная ей формула, представляющая конъюнкций элементарных дизъюнкций. Например, $(x \vee \bar{y}) \& (\bar{x} \vee z \vee y) \& (x \vee z)$.

Так как для того, чтобы выражение в форме КНФ было тождественно истинным, необходимо и достаточно, чтобы был истиной каждый дизъюнкт в него входящий, то при построении логического вывода можно перейти от КНФ к множеству дизъюнктов, в котором каждый элемент множества имеет значение «истино».

Определение. ***Пустой дизъюнкт*** – это такой дизъюнкт, значение которого тождественно ложно.

Итак, невыполнимость формул, из которых формируется конечное множество дизъюнктов *S*, доказывается с помощью следующего алгоритма.

Шаг 1. Проверка множества S на невыполнимость. Если пустой дизъюнкт принадлежит множеству *S* (он может либо присутствовать изначально или получается из-за того, что в множестве одновременно присутствует некоторая литера и ее отрицание), это означает, что множество *S* невыполнимо и алгоритм свою работу закончил. Иначе переходим на шаг 2.

Шаг 2. Построение резольвенты. Выбираем *l*, S_1, S_2, такие, что $l \in S_1$ и $\bar{l} \in S_2$ и вычисляем резольвенту *r*. Если невозможно выбрать *l*, S_1, S_2 , соответствующие указанным условиям, то алгоритм свою работу закончил и можно сказать, что исходное множество выполнимо. Иначе переходим на шаг 3.

Шаг 3.Обновление множества дизъюнктов. Заменяем множество дизъюнктов S *на* $S \cup \{r\}$, т.е. добавляем к существующим дизъюнктам новый дизъюнкт – резольвенту, полученную на предыдущем шаге. После чего переходим на шаг 1.

Пример. Доказать, используя метод резолюций, невыполнимость следующего множества дизъюнктов $S = \{ p \vee q, p \vee r, \bar{q} \vee \bar{r}, \bar{p} \}$. Пронумеруем дизъюнкты. Это делается для того, чтобы при построении резольвенты можно было сослаться на дизъюнкты, которые для этого использовались:

1. $p \vee q$
2. $p \vee r$
3. $\bar{q} \vee \bar{r}$
4. $\bar{p}$

Порождаем логические следствия, при порождении следствия будем указывать, номера участвовавших в построении дизъюнктов:

5. $p \vee \bar{r}$ *(1,3)*
6. q *(1,4)*
7. $p \vee \bar{q}$ *(2,3)*
8. r *(2,4)*
9. p *(2,5)*
10. $\bar{r}$ *(3,6)*
11. $\bar{q}$ *(3,8)*
12. $\bar{r}$ *(4,5)*
13. $\bar{q}$ *(4,7)*
14. *Пустой дизъюнкт (4,9)*

Замечание. Алгоритм проверки невыполнимости недетерминирован. Вообще говоря, возможен не один вариант выбора значений для *l*, S_1 и S_2. В приведенном примере дизъюнкты выбирались в лексико-графическом порядке номеров. Такая стратегия не оптимальна, так как некоторые резольвенты оказались не нужны, а также вычислялись более одного раза. Этот же пример с минимумом резолюций будет выглядеть так:

5. q (1,4)
6. r (2,4)
7. $\bar{q}$ (3,6)
8. *Пустой дизъюнкт* (5,7)

Ясно, что принятая стратегия может существенно влиять на процесс выполнения алгоритма. Тем не менее существуют два свойства, не зависящие от используемой стратегии.

Свойство 1. Если множество *S* не содержит ни одной пары дизъюнктов, допускающих резольвенту, то оно выполнимо (за исключением случая, когда оно содержит пустой дизъюнкт).

Свойство 2. Если выполнение этого алгоритма закончилось нормально после порождения пустого дизъюнкта, то установлена невыполнимость исходного множества *S*.

В заключение рассмотрим несколько примеров применения метода резолюций в логике высказываний.

Пример. Доказать, используя метод резолюций, что *S* является логическим следствием множества гипотез *H*, где $H = \left\{a \vee (b \to c), \overline{c \,\&\, d} \vee e, f \vee \left(\overline{\bar{d} \vee e}\right)\right\}$, а $S = \bar{a} \vee \bar{b} \vee f$. Сначала преобразуем множество гипотез в множество дизъюнктов:

1. $\bar{a} \vee (b \to c) = \bar{a} \vee \bar{b} \vee c$.
2. $\overline{c \,\&\, d} \vee e = \bar{c} \vee \bar{d} \vee e$.
3. $f \vee \left(\overline{\bar{d} \vee e}\right) = f \vee d\bar{e} = (f \vee d)(f \vee \bar{e})$.
4. $\bar{S} = \overline{\bar{a} \vee \bar{b} \vee f} = a \,\&\, b \,\&\, \bar{f}$.

Для доказательства того, что $H \models S$ необходимо и достаточно доказать невыполнимость следующего множества дизъюнктов

$$H = \{\bar{a} \vee \bar{b} \vee c, \bar{c} \vee \bar{d} \vee e, f \vee d, f \vee \bar{e}, a, b, \bar{f}\}:$$

1. $\bar{a} \vee \bar{b} \vee c$
2. $\bar{c} \vee \bar{d} \vee e$
3. $f \vee d$
4. $f \vee \bar{e}$
5. a
6. b
7. $\bar{f}$
8. d *(3 – 7)*
9. $\bar{b} \vee c$ *(1 – 5)*

10. c *(6 – 9)*
11. $\bar{e}$ *(4 – 7)*
12. $\bar{c} \vee \bar{d}$ *(2 – 11)*
13. $\bar{d}$ *(10 – 12)*
14. Пустой дизъюнкт (8 – 13)

Пример. Пусть дано множество утверждений, сформулированных на естественном языке, и некоторое заключение, которое из них следует. Требуется превратить их в множество высказываний и доказать справедливость рассуждений, используя метод прямой дедукции. Набор рассуждений следующий:

1) если пойти на первую пару, то надо встать рано, а если играть в преферанс, то лечь придется поздно;

2) если лечь поздно и рано встать, то спать придется мало;

3) мало спать нельзя.

Заключение: надо или не играть в преферанс, или не идти на первую пару.

Введем следующие обозначения для высказываний:

g – встать рано;

d – играть в преферанс;

c – идти на первую пару;

s – лечь поздно спать;

e – мало спать.

Используя введенные обозначения, перейдем от утверждений к следующему набору гипотез: $H_1 = (c \to g) \& (d \to s)$, $H_2 = s \& g \to e$, $H_3 = \bar{e}$. Следствие примет вид $A = \bar{c} \vee \bar{d}$. При построении доказательства по дедукции в качестве механизма воспользуемся методом эквивалентных преобразований и методом резолюций. Требуется доказать невыполнимость следующего множества: $\{H_1, H_2, H_3, \bar{A}\}$, где $\bar{A} = \overline{\bar{c} \vee \bar{d}} = cd$.

Используя метод эквивалентных преобразований, требуется доказать, что имеет место равенство $cd(c \to g)(d \to s)(sg \to e)\bar{e} = 0$. Докажем, что

$$cd(c \to g)\&(d \to s)(sg \to e)\bar{e} = cd(\bar{c} \vee g)(\bar{d} \vee s)(\bar{s} \vee \bar{g} \vee e)\bar{e} =$$
$$= (cd\bar{c} \vee cdg)(\bar{d} \vee s)(\bar{e}\bar{s} \vee \bar{g}\bar{e} \vee e\bar{e}) = (cd\bar{d}g \vee cdgs)(\bar{e}\bar{s} \vee \bar{e}\bar{g}) = cdgs\bar{e}\bar{s} \vee cdgs\bar{e}\bar{g} = 0.$$

Теперь построим доказательство, используя метод резолюций. Для этого приведем имеющиеся гипотезы к форме дизъюнктов:

$H_1 = (c \to g)\,\&\,(d \to s) = (\bar{c} \vee g)(\bar{d} \vee s)$;

$H_2 = s\,\&\,g \to e = \overline{s\,\&\,g} \vee e = \bar{s} \vee \bar{g} \vee e$;

$H_3 = \bar{e}$.

Отрицание следствия будет иметь вид $\bar{A} = \overline{\bar{c} \vee \bar{d}} = cd$, а цепочка доказательства:

1. $\bar{c} \vee g$
2. $\bar{d} \vee s$
3. $\bar{s} \vee \bar{g} \vee e$
4. $\bar{e}$
5. c
6. d
7. g (1-5)
8. s (2-6)
9. $\bar{s} \vee e$ (3-7)
10. e
11. *Пустой дизъюнкт (4-10)*

3.4. Логика предикатов

3.4.1. Основные понятия логики предикатов

В алгебре логики высказывания рассматриваются как единый объект с точки зрения истинности или ложности. Структура и содержание высказываний не рассматриваются. Однако на практике для построения полноценного логического вывода важно иметь представление о структуре и содержании используемых в выводе высказываний. Поэтому логика предикатов является, по сути, расширением логики

высказывания, которую включает в себя в качестве составной части.

Определение. Одноместным предикатом $P(x)$ называется произвольная функция переменной x, определенная на множестве M и принимающая значение из множества $\{0, 1\}$.

Обобщим это определение.

Определение. Предикатом P называется n-местная функция, определенная на производном множестве M и принимающая в качестве значений элементы из двухэлементного множества $\{0, 1\}$, где 0 и 1 интерпретируются как ложь и истина соответственно. Выражение вида $P(x_1, x_2, \ldots, x_n)$ можно трактовать так, что переменные $x_1, x_2, \ldots, x_n$ связаны отношением P.

Используя функциональную форму записи для предикатов, можно сказать, что предикатом $P(x_1, x_2, \ldots, x_n)$ называется функция $P: M^n \rightarrow B$, где B – двоичное множество, а M – произвольное множество.

Таким образом, n-местный предикат, – это двузначная функция от n аргументов, определенная на произвольном множестве M, принимающая значение 0 или 1 (0 и 1 интерпретируются как ложь и истина соответственно). Область определения M называется предметной областью предиката, а $x_1, x_2, \ldots, x_n$ – предметными переменными.

Предикат может выражать высказывание либо о свойствах объекта, либо об отношении между объектами. Возможность описывать с помощью предикатов не только функции, но и отношения, определяется следующим:

а) если $a_1, a_2, \ldots, a_n$ – элементы множества M, то каждому n-местному отношению R соответствует предикат P такой, что $P(a_1, a_2, \ldots, a_n) = 1$ тогда и только тогда, когда $(a_1, a_2, \ldots, a_n) \in R$;

б) всякий предикат $P(x_1, x_2, \ldots x_n)$ определяет отношение R, такое, что $(a_1 a_2 \ldots a_n) \in R$, если и только если $P(a_1, a_2, \ldots a_n) = 1$. При этом R задает область истинности предиката P.

Предикат можно поставить в соответствие и непрерывной функции типа $F: M^n \rightarrow M$. Такой функции можно поставить в соответствие $(n + 1)$ - местный предикат P такой, что $P(a_1, a_2, \ldots, a_n, a_{n+1}) = 1$, если и только если $f(a_1, a_2, \ldots, a_n) = a_{n+1}$.

Таким образом, в общем случае предикат P – двоичная переменная, то есть переменное высказывание, истинность которого определяется значениями аргументов $(x_1, x_2, \ldots, x_n)$, а аргументы x_i – чаще нелогические переменные. После подстановки вместо x_i конкретных элементов множества M предикат $P(a_1,a_2,\ldots,a_n)$ перестает быть переменной и принимает одно из двух возможных значений (0 или 1).

Пример.

1. Рассмотрим утверждение «x – целое число». Введем предикат I, обозначающий отношение «быть целым числом», тогда в виде предикатного выражения утверждение может быть записано так : $I(x)$.
2. Рассмотрим утверждение $x < y$. Введем предикат S с двумя аргументами, первый из которых меньше второго, тогда $S(x, y)$ соответствует введенному утверждению.
3. Элементы x_i множества M – города. Предикат $P(x)$ устанавливается таким образом: x – это столица Украины. Тогда $P(Воронеж)=0$, а $P(Киев)=1$.
4. Задана функция $z=x+y$, где x, y, z – действительные числа. Пусть предикат $P(x,y,z)$ соответствует этой функции. Тогда $P(2, 3, 5)=1$.

Если вместо переменных в предикат подставить объекты $a_1, a_2, \ldots, a_m \in M$, где M – множество, на котором определено P, то значение $P(a_1, a_2, \ldots, a_m)$ можно рассматривать как истинное или ложное.

Пример. (для предикатов определенных в предыдущем примере). Пусть в обоих случаях предикаты определены на множестве R – действительных чисел. Тогда

1) если $x = 5$, то предикат $I(5) = 1$;
   если $x = 7.3$; то $I(7.3) = 0$;
2) если $x = 5$; $y = 10.5$, то $S(5; 10.5) = 1$;
   если $x = 27.1$; $y = 4.3$, то $S(27.1; 4.3) = 0$.

Определение. Для предиката $P(x_1, x_2, \ldots, x_n)$ можно выделить множество наборов $b_i = (a_{i1}, a_{i2}, \ldots, a_{in})$ таких, что $a_{i1}, a_{i2}, \ldots, a_{in} \in M$, которые, будучи подставленными в предикат P, приводят его к значению истина. Объединение всех этих наборов

называется множеством истинных наборов предиката *P,* обозначим его I_p.

Пример. Для предиката $S(x, y)$, определенного ранее, множество I_p может быть определено следующим образом:

$$I_p = \{(x, y) \mid x \in R, y \in R, x < y\}.$$

Определение. Предикат $P(x_1, x_2, \ldots, x_n)$ называется ***тождественно истинным***, если его множество I_p образуют все возможные наборы, которые можно определить на множестве *M*.

Определение. Предикат $P(x_1, x_2, \ldots, x_n)$ называется ***тождественно ложным***, если его множество $I_p = \varnothing$.

Так как предикаты, как и высказывания, могут принимать значения истина или ложь (1 или 0), то к ним применимы все операции логики высказываний.

Пример. Рассмотрим предикаты *I(x)* и *S(x,y)* из предыдущих примеров. Пусть *I* и *S* определены на множестве *R*, тогда:

1) при *x* = 3, *y* =10 : *I(3) = 1, S(3, 10) = 1*, имеем *I(3)&S(3, 10) = 1&1 = 1, S(3, 10)→I(3) = 1→1 = 1*;
2) при *x*=7.5, *y*=11: *I(7.5)=0, S(7.5,11)=1*, имеем *I(7.5)&S(7.5,11)=0&1=0, I(7.5) → S(7.5, 11) = 0 → 1 = 1*;
3) задана вычислительная процедура: «Повторять цикл, пока переменные *x* и *y* не станут равными, либо прекратить вычисления после 100 повторений. Область определения: *x* и *y* – действительные числа; *i* – целое число, которое в каждом шаге увеличивается на единицу. Предикат *P* задается условием: $\overline{(x = y) \vee (i > 100)}$. При этом процедура может задаваться условием: «Повторять цикл, если *P=1*» .

Кроме логических операций, в логике предикатов вводятся *кванторные* операции (или просто *кванторы*). Наличие кванторов в выражении позволяет существенно повысить выразительную мощность предикатных предложений. Кванторов всего два.

1. **Квантор всеобщности (∀).** Пусть *P(x)* это предикат, определенный на множестве М. Тогда под выражением $\forall x P(x)$ – понимается высказывание, которое истинно для любого элемента $x \in M$. Соответствующее этому высказыванию предложение

можно сформулировать так: «Для любого x выражение $P(x)$ истинно».

2. **Квантор существования ($\exists$).** Пусть $P(x)$ это предикат, определенный на множестве M. Тогда под выражением $\exists x P(x)$ понимается высказывание, которое истинно, если существует элемент $x \in M$, для которого $P(x)$ истинно, и ложно в противном случае. Соответствующее ему предложение: «Существует x, при котором значение $P(x)$ истинно».

Пример. Рассмотрим предикат $I(x)$ означающий "x – целое число". Пусть предикат I определен на множестве R – действительных чисел. Тогда выражение $\forall x I(x)$ – соответствует утверждению: "Все действительные числа целые". Очевидно, что это утверждение ложно. Выражение $\exists x I(x)$ – соответствует утверждению: «Существуют действительные числа, являющееся целыми» – это истинное утверждение. Теперь предположим, что предикат I определен на множестве N – натуральных чисел. Тогда оба выражения $\forall x I(x)$ и $\exists x I(x)$ будут истинными.

Кванторы могут применяться и к предикатам, определенным относительно n переменных, при этом к одному предикату может быть применено несколько кванторов, относящихся к разным переменным. Порядок следования кванторов существенно влияет на истинность предиката.

Пример. Пусть предикат $P(x, y)$ - "$x \le y$" определен на множестве N.

Рассмотрим предикатные выражения, приведенные в таблице.

№	Предикатная запись	Предложение	Значение истинности
1.	$\exists x \forall y P(x, y)$	Существует x, который меньше любого y	1
2.	$\exists x \exists y P(x, y)$	Существуют такие x и y, что $x \le y$	1
3.	$\forall x \forall y P(x, y)$	Для любого x и любого y имеет место $x \le y$	0

Входящие в предикатное выражение переменные могут быть *связанными* или *свободными* и это зависит от того, каким образом определена *область действия квантора*, входящего в формулу. Например, для выражения вида $\forall x \exists y(P(x, y) \to \forall z R(x, z))$ область действия кванторов определена следующим образом: для квантора $\forall x$ - $\exists y(P(x, y) \to \forall z R(x, z))$, для квантора $\exists y$ – $P(x, y) \to \forall z R(x, z)$, а для квантора $\forall z$ – $R(x, z)$.

Определение. Вхождение переменной x в формулу называется ***связанным*** тогда и только тогда, когда оно совпадает с вхождением в квантор $(\forall x)$ или $(\exists y)$ и находится в области действия квантора. Вхождение переменной в формулу свободно тогда и только тогда, когда оно не является связанным.

Определение. Переменная свободна в формуле, если хотя бы одно ее вхождение в эту формулу свободно. Отметим, что переменная в формуле может быть свободной и связанной одновременно.

Определение. Переменная *называется* ***квантифицированной*** , если она используема в кванторных операциях, т.е. $\forall x$ или $\exists x$.

Пример.

$P(x)$ – x свободная;

$\forall x P(x)$, $\exists x P(x)$ – x связанная;

$\exists x P(x, y)$ – x связанная, y свободная;

$\exists x \forall y P(x, y)$ – x и y связанные.

3.4.2. Логика предикатов как формальная система

Как и в случае логики высказываний, логику предикатов можно рассматривать как формальную систему. Для этого необходимо определить четыре основные компоненты, лежащие в основе любой формальной системы.

1. Алфавит. В логике предикатов обычно используют следующие категории символов:

- предметные переменные – x, y, z и т.п., которые принимают значение из множества M;

- индивидные константы – *a, b, c* и т.п., то есть нульместные функциональные константы или константы предметной области;
- функциональные константы – *f, g, h* и т.п., используются для обозначения функций;
- высказывания – *p, q, r* и т.п.;
- предикатные константы – *P, R*, H, *Q* и т.п.;
- символы логических операций – &, $\vee$, $\rightarrow$ и т.д.;
- символы кванторных операций – $\forall, \exists$;
- вспомогательные символы – (,).

2. Правила построения формул. Для начала введем несколько определений, которыми будем оперировать при определении правил построения формул.

- ***Термом*** является всякая переменная и всякая функциональная форма.
- ***Функциональная форма*** – это функциональная константа, соединенная с некоторым числом термов. Если *f* – функциональная константа (*n*-местная) и $t_1, \ldots, t_n$ – термы, то соответствующая форма записывается так $f(t_1, t_2, \ldots, t_n)$. Если *n = 0*, то *f* превращается в индивидную константу.
- ***Предикатная форма*** – это предикатная константа, соединенная с соответствующим числом термов. Если *P* – предикатная константа *m*-местная константа и $t_1, \ldots, t_m$ – термы, то соответствующая форма записывается так $P(t_1, t_2, \ldots, t_m)$. Если *m = 0*, то пишут *P*.

Замечание. Следует помнить, что функциональная форма без аргументов это просто индивидуальная константа, а предикатная форма без аргументов – просто высказывание.

- ***Атом*** – это предикатная форма или некоторое равенство (выражение вида *s=t*, где *s* и *t* – термы). Для равенства характерно то же, что и для предикатной формы, т.е. о нем можно сказать, что оно истинно или ложно.

Теперь перейдем к определению понятия формулы, которое делается рекурсивно по следующим правилам:

1. Атом есть формула.
2. Если *A* – формула, то $\overline{A}$ – формула.

3. Если A и B – формулы, то $(A \vee B), (A \to B),$, *(A&B), (A~B)* – формулы.

4. Если A – формула и x – переменная, то $\forall xA, \exists xA$ – формулы.

Замечание. Возможно опускать лишние скобки так же, как и в логике высказываний, если это допускает контекст.

3. Определение аксиом. Выбор системы аксиом в исчислении предикатов может быть осуществлен по-разному. Один из наиболее простых способов состоит в том, что берутся уже ранее определенные аксиомы из исчисления высказываний и дополняются еще двумя, связанными с использованием кванторов:

$\forall x F(x) \to F(y)$;

$F(t) \to \exists x\, F(x)$, где t – не содержит переменной x.

4. Правила вывода. Правила вывода также полностью заимствуются из логики высказываний, кроме того, к ним добавляются еще два правила следующего вида:

- правило введения квантора существования: $\dfrac{P(x) \to F}{\exists x\, P(x) \to F}$;
- правило введения квантора общности: $\dfrac{F \to P(x)}{F \to \forall x\, P(x)}$.

При этом в обоих случаях считается, что F не зависит от x.

Замечание 1. Понятия вывода и доказуемой формулы базируются на введенных ранее этих понятиях для произвольной ФС.

Замечание 2. Любые формальные системы, имеющие в качестве аксиом аксиомы, совпадающие с исчислением предикатов, называется *формальными системами первого порядка*. Обычно они отличаются только правилами словообразования и могут содержать дополнительные аксиомы. Термин «система первого порядка» означает, что в их формулах действие квантора может распространяться только на предикатные переменные и внутренние символы в формулах. К системам второго порядка относятся те, в которых действие квантора распространяется и на предикаты. В системах более

высокого порядка действие квантора может распространяться и на предикаты от предикатов и так далее.

В заключение приведем несколько примеров построения предикатных выражений.

Примеры.

1. Утверждение: «Все слоны серые».

Вводимые предикаты: *слон(x)* – «*x* – слон»; *цвет(x,y)* – «*x* цвета *y*».

Предикатное выражение: $\forall x(\text{слон}(x) \to \text{цвет}(x, \text{серый}))$.

2. Утверждение: «Для любого множества *x* существует множество *y* такое, что мощность *y* больше, чем мощность *x*».

Вводимые предикаты: *множество(x)* – «*x* – множество»; *мощность(x,u)* – «мощность множества *x* равна *u*»; *больше(x,u)* – «значение *x* больше значения *u*».

Предикатное выражение:

$$\forall x(\text{множество}(x) \to (\exists y)(\exists u)(\exists v)((\text{множество}(y) \,\&\, \text{мощность}(x,u) \,\&\, \& \,\text{мощность}(y,v) \,\&\, \text{больше}(v,u))).$$

3. Утверждение: «Все кубики, находящиеся на кубиках, которые были сдвинуты или были скреплены с кубиками, которые сдвигались, тоже будут сдвинуты».

Вводимые предикаты: *куб(x)* – «*x* – куб»; *сверху(x,y)* – «*x* расположен сверху *y*»; *скреплен(x,y)* – «*x* скреплен с *y*»; *сдвинут(x)* – «*x* – сдвинут»;

Предикатное выражение:

$$(\forall x)(\forall y)(\text{куб}(x) \,\&\, \text{куб}(y) \,\&\, (\text{сверху}(x,y) \vee \text{скреплен}(x,y)) \,\&\, \& \,\text{сдвинут}(y)) \to \text{сдвинут}(x).$$

3.4.3. Определение значения истинности предикатных формул

О логическом значении формулы логики предикатов можно говорить лишь тогда, когда задано множество *M*, на котором определены входящие в формулу предикаты. При задании конкретных значений аргументам предикатов предикаты становятся высказываниями и можно говорить об их истинности или ложности.

Как и в алгебре высказываний в логике предикатов имеет место понятие равносильности предикатных выражений.

Определение. Две формулы логики предикатов считаются ***равносильными в области M***, если они принимают одинаковые логические значения при всех входящих в них переменных, отнесенных к области *M*.

Определение. Две формулы логики предикатов называются ***равносильными***, если они равносильны на всякой области.

На понятие равносильности формул опираются и правила эквивалентных преобразований предикатных выражений, которые позволяют получить из предикатного выражения одного вида равносильное ему выражение другого вида. Очевидно, что в тех случаях, когда преобразования не изменяют ограничений, накладываемых кванторами, можно использовать все известные правила из АВ, если заменить в них высказывания на предикаты. Кроме того, в исчислении предикатов существуют и собственные правила, регламентирующие преобразование выражений, содержащих кванторы.

Пусть *A(x), A(x,y) и B(x) –* предикаты, *p –* высказывание, тогда правила имеют следующий вид:

1. $\overline{\forall x\, A(x)} = \exists x\, \overline{A}(x)$.
2. $\overline{\exists x\, A(x)} = \forall x\, \overline{A}(x)$.
3. $\forall x\, A(x) = \overline{\exists x\, \overline{A}(x)}$.
4. $\exists x\, A(x) = \overline{\forall x\, \overline{A}(x)}$.
5. $\forall x\,(A(x)\,\&\,B(x)) = \forall x\, A(x)\,\&\,\forall x\, B(x)$.
6. $\forall x\,(A(x)\vee B(x)) = \forall x\, A(x)\vee \forall x\, B(x)$.
7. $p\,\&\,\forall x\, A(x) = \forall x\,(p\,\&\,A(x))$.
8. $p\vee \forall x\, A(x) = \forall x\,(p\vee A(x))$.
9. $p\to \forall x\, B(x) = \forall x\,(p\to B(x))$.
10. $\forall x\,(B(x)\to p) = \exists x\, B(x)\to p$.
11. $\exists x\,(A(x)\vee B(x)) = \exists x\, A(x)\vee \exists x\, B(x)$.
12. $\exists x\,(A(x)\,\&\,B(x)) = \exists x\, A(x)\,\&\,\exists x\, B(x)$.
13. $p\vee \exists x\, B(x) = \exists x\,(p\vee B(x))$.
14. $p\,\&\,\exists x\, B(x) = \exists x\,(p\,\&\,B(x))$.
15. $\exists x\, A(x)\,\&\,\exists y\, B(x) = \exists x\,\exists y\,(A(x)\,\&\,B(x))$.

16. $\exists x\, \exists y\, A(x, y) = \exists y\, \exists x\, A(x, y)$.

17. $\forall x\, \forall y\, A(x, y) = \forall y\, \forall x\, A(x, y)$.

18. $\exists x\, (B(x) \to p) = \forall x\, B(x) \to p$.

Таким образом, в случае если не определены значения предметных переменных и нельзя установить значение истинности предикатов, считается, что невозможно установить значение истинности предикатных формул. Однако в некоторых частных случаях это возможно сделать путем эквивалентных преобразований, базирующихся на ранее определенных формулах эквивалентности.

Пример. Доказать тождественную истинность заданного предикатного выражения

$$\forall x\,(P(x) \to (P(x) \vee P(y))) = \forall x\,\left(\overline{P}(x) \vee P(x) \vee P(y)\right) = \forall x\,(1 \vee P(y)) = \forall x\,(1) = 1.$$

Пример. Доказать тождественную ложность заданного предикатного выражения

$$\exists x \exists y\left((F(x) \to F(y)) \,\&\, \left(F(x) \to \overline{F}(y)\right) \,\&\, F(x)\right) =$$
$$= \exists x \exists y\left(\left(\overline{F}(x) \vee F(y)\right) \,\&\, \left(\overline{F}(x) \vee \overline{F}(y)\right) \,\&\, F(x)\right) =$$
$$= \exists x \exists y(\left(\overline{F}(x) \vee F(y)\right) \,\&\, \left(\overline{F}(x) \,\&\, F(x) \vee \overline{F}(y) \,\&\, F(x)\right) =$$
$$= \exists x \exists y(\left(\overline{F}(x) \vee F(y)\right) \,\&\, \left(0 \vee \overline{F}(y) \,\&\, F(x)\right) =$$
$$= \exists x \exists y\left(\left(\overline{F}(x) \vee F(y)\right) \,\&\, \overline{F}(y) \,\&\, F(x)\right) =$$
$$= \exists x \exists y(\overline{F}(x) \,\&\, \overline{F}(y) \,\&\, F(x) \vee F(y) \,\&\, \overline{F}(y) \,\&\, F(x)) = \exists x \exists y(0 \vee 0) = 0.$$

В большинстве случаев, особенно если в выражении присутствуют кванторы, найти универсальный метод определения выполнимости такого выражения невозможно, поэтому считается, что исчисление предикатов неразрешимо, однако существует большое множество задач в логике предикатов, для которых существуют эффективные алгоритмы, позволяющие определить их общезначимость или вычислимость.

Одним из наиболее эффективных методов определения разрешимости множества предикатных формул является метод резолюций, который, по сути, является развитием уже известного из АВ метода применительно к исчислению предикатов.

3.4.4. Методы резолюций для логики предикатов

Как и ранее, под резолюцией будет пониматься правило вывода, которое может быть применено к определенному классу формул – *дизъюнктам.* Принцип резолюции состоит в том, что, будучи примененным к двум формулам, позволяет получить новую формулу как их следствие.

Однако прежде чем перейти к рассмотрению самого метода, следует ознакомиться с некоторыми понятиями, без которых невозможна его реализация.

Унификация. Так как анализ выражений в формальных системах осуществляется в чисто синтаксической форме, без учета семантики, то возникает необходимость приведения отдельных выражений к единой форме. Эти приведения базируются на использовании унификации.

Например, для создания $W_2(A)$ из $\forall x(W_1(x) \to W_2(x))$ и $W_1(A)$ необходимо найти подстановку «*A вместо x*», которая сделает идентичными $W_1(A)$ и $W_1(x)$. Поиск подстановок термов на место переменных называется ***унификацией***. Унификация основывается на другом понятии – ***подстановка***.

Определение. ***Подстановочным частным случаем*** называется подстановка в некоторое выражение термов вместо переменных.

Пример. Для выражения $P(x, f(y), B)$ имеется четыре частных случая подстановки:

1) $P(z, f(\omega), B)$ – алфавитная, просто замена одних переменных на другие;
2) $P(x, f(A), B)$ – константу подставили в функцию f;
3) $P(g(z), f(A), B)$ – функциональное выражение вместо переменной x;
4) $P(C, f(A), B)$ – основной частный случай, т.к. везде подставлены константы вместо переменных.

Любую подстановку можно представить с помощью множества упорядоченных пар вида $S = \{t_1/v_1, t_2/v_2, \ldots, t_n/v_n\}$, где пара t_i/v_i означает, что переменная v_i заменяется на терм t_i. Следует отметить, что при выполнении подстановки должны соблюдаться два правила:

– каждое вхождение переменной заменяется на один и тот же терм;

– переменную нельзя заменить на терм, содержащий ту же самую переменную.

Для предыдущего примера подстановки описанные с помощью введенного формализма, имеют следующий вид:

1) $S_1 = \{z/x, \omega/y\};$
2) $S_2 = \{A/y\};$
3) $S_3 = \{g(z)/x, A/y\};$
4) $S_4 = \{C/x, A/y\}.$

Для обозначения подстановки часто используется следующая запись. Если S – подстановка, а E – выражение, к которому она применяется, то пишут E_S .

Если подстановка S применяется к каждому элементу некоторого множества $\{E_i\}$, то множество подстановочных примеров обозначается $\{E_i\}_S$.

Определение. Говорят, что ***множество*** $\boldsymbol{E=\{E_1, E_2, \ldots, E_n\}}$ ***унифицируемо***, если существует такая подстановка S, что $E_{1S} = E_{2S} = \ldots = E_{nS}$. В этом случае подстановка S называется ***унификатором*** для множества E, т.к. после ее применения множество склеивается в один элемент.

Пример. Пусть $A=\{P(x, f(y), B), P(x, f(B), B)\}$, рассмотрим подстановку $S=\{C/x, B/y\}$, унифицируем и получаем $A = \{P(A, f(B), B)\}$ (в подстановке C необходимости не было).

Пример. $P(x, f(y), B)_{S_1} = P(z, f(\omega), B)$.

Унификация производится при следующих условиях:

1. Если термы – константы, то они унифицируемы тогда и только тогда, когда они совпадают.
2. Если в первом дизъюнкте терм – переменная, а во втором – константа, то они унифицируемы, при этом вместо переменной подставляется константа.
3. Если терм в первом дизъюнкте – переменная и во втором дизъюнкте терм тоже переменная, то они унифицируемы.
4. Если в первом дизъюнкте терм – переменная, а во втором – употребление функции, то они унифицируемы, при этом вместо переменной подставляется употребление функции.

5. Унифицируются между собой термы, стоящие на одинаковых местах в одинаковых предикатах.

Пример. Рассмотрим дизъюнкты:

1. *Q(a, b, c)* и *Q(a, d, l).* Дизъюнкты не унифицируемы.
2. *Q(a, b, c)* и *Q(x, y, z).* Дизъюнкты унифицируемы. Унификатор – *Q(a,b,c).*

Если необходимо последовательно выполнить несколько подстановок: S_1, S_2, то можно записать $E_{S_1S_2}$. На этом действии базируется такое понятие, как ***композиция подстановок.*** Если S и S' являются двумя множествами подстановок, то их композиция S и S' (пишется SS') получается после применения S' к элементам S и добавления результата к S. Можно сказать, что композиция подстановок – это метод, с помощью которого объединяются подстановки унификации. Обобщая сказанное, можно ввести следующее определение понятия композиции подстановок.

Определение. Композиция подстановок *g* и *s* есть функция *gs*, определяемая следующим образом: (*gs*) [*t*]=*g*[*s*[*t*]], где *t* – терм, *g* и *s* – подстановки, а *s*[*t*] – терм, который получается из *t* путем применения к нему подстановки *s*.

Пример. Пусть задана последовательность подстановок *{x/y,w/z}, {v/x}, {A/v, f(B)/w}*, они эквивалентны одной подстановке *{A/y,f(B)/z}.* Последняя подстановка была выведена путем компоновки *{x/y,w/z}* и *{v/x}* для получения *{v/y,w/z}* и компоновки результата с *{A/v, f(B)/w}* для получения *{A/y,f(B)/z}.*

Основное требование алгоритма унификации состоит в том, что унификатор должен быть максимально общим, т.е. для любых двух выражений должен быть найден ***наиболее общий унификатор*** (НОУ). Это важно, т.к. при потере общности в процессе решения уменьшается вероятность достижения окончательного решения или такая возможность исчезает полностью. Например, предикаты *P(x)* и *P(y)*, определенные на множестве целых чисел, можно унифицировать любым константным выражением вида *{4/x, 4/y}.* Однако число 4 в данном случае не является НОУ, используя в качестве унификатора любую переменную можно получить более общее выражение: *{z/x, z/y}.* Решения, полученные при использовании

первой подстановки, всегда будут ограничены содержащейся в них константой 4, лимитирующей логический вывод. Следовательно, константу 4 можно использовать в качестве унификатора, но это снижает универсальность результата.

Определение. Если s – произвольный унификатор выражения E, а g – ***наиболее общий унификатор*** этого набора выражений, то в случае применения s к E будет существовать еще один унификатор s' такой, что $Es = Egs'$, где Es и Egs' – композиции унификация, примененные к выражению E.

Очевидно, что прежде чем приводить выражения к некоторому общему виду, необходимо выявить их различия, т.е. *рассогласования*.

Определение. ***Множество рассогласований*** непустого множества дизъюнктов $\{E_1, \ldots, E_n\}$ получается путем выявления первой (слева) позиции, на которой не для всех дизъюнктов из E стоит один и тот же символ, и выписывания из каждого дизъюнкта терма, который начинается с символа, занимающего данную позицию. Множество термов и есть множество рассогласований в E.

Пример. Рассмотрим дизъюнкты: $\{P(x, f(y, z)), P(x, a), P(x, g(h(k(x))))\}$.

Множество рассогласований состоит из термов, которые начинаются с пятой позиции, и представляет собой множество $\{f(x, y), a, g(h(k(x)))\}$.

Рассмотрим алгоритм унификации для нахождения наиболее общего унификатора. Пусть E – множество дизъюнктов, D – множество рассогласований, k – номер итерации, g_k – наиболее общий унификатор на k-й итерации.

Шаг 1. Пусть $k=0$, $g_k=e$ (пустой унификатор), $E_k=E$.

Шаг 2. Если для E_k не существует множества рассогласований D_k, то алгоритм завершает работу и g_k – наиболее общий унификатор для E. Иначе найдем множество рассогласований D_k.

Шаг 3. Если существуют такие элементы v_k и t_k в D_k, что v_k – переменная, не входящая в терм t_k, то перейдем к шагу 4. В противном случае алгоритм завершает работу и E не унифицируемо.

Шаг 4. Пусть $g_{k+1}=g_k \{ t_k / v_k\}$, заменим во всех дизъюнктах E_k терм t_k на v_k.

Шаг 5. $k=k+1$. Перейти к шагу 2.

Пример. Рассмотрим дизъюнкты: $E=\{P(f(a), g(x)), P(y, y)\}$.

Итерация I

Шаг 1. $E_0=E$, $k=0$, $g_0=e$.

Шаг 2. $D_0=\{f(a),y\}$.

Шаг 3. $v_0=y$, $t_0=f(a)$.

Шаг 4. $g_1=\{f(a)/y\}$, $E_1=\{P(f(a), g(x)), P(f(a), f(a))\}$. Переход на шаг 2.

Итерация II

Шаг 2. $D_1=\{g(x),f(a)\}$.

Шаг 3. Так как нет переменной в множестве рассогласований D_1, то, следовательно, алгоритм унификации завершается, множество E – не унифицируемо.

Обычно унификацию используют для приведения в соответствие различных выражений. Если в одном из выражений подставлены вместо переменных константы, а другие выражения приводятся в соответствие с ним**,** то это называется ***сравнением с образом.***

Как отмечалось ранее, метод резолюций применим к множеству дизъюнктов. Поэтому рассмотрим ***последовательность действий, которую необходимо реализовать, для приведения любой формулы логики предикатов к множеству дизъюнктов***. Проиллюстрируем процесс приведения примером. Пусть задано следующее предикатное выражение:

$$(\forall x)(P(x) \to (\forall y)(P(y) \to P(f(x,y))) \,\&\, \overline{\forall y(Q(x,y) \to P(y))}).$$

Его следует привести к множеству предложений. На каждом шаге будем приводить пример, который демонстрирует результат применения действий соответствующего шага к заданному выражению.

Шаг 1. Используя правила эквивалентных преобразований, в исходном выражении все логические операции записывают через операции дизъюнкции, конъюнкции и отрицания. То есть результат будет записан как выражение, определенное в рамках

следующей функционально полной системы логических функций $\{\vee, \&, \bar{}\}$.

Пример. $\forall x(\overline{P}(x) \vee \forall y(\overline{P}(y) \vee P(f(x,y))) \& \overline{\forall y(\overline{Q}(x,y) \vee P(y))})$.

Шаг 2. Используя правила де Моргана, выполняют преобразования, обеспечивающие применение операции отрицания только к литералам.

Пример. $\forall x(\overline{P}(x) \vee \forall y(\overline{P}(y) \vee P(f(x,y))) \& \exists y(Q(x,y) \& \overline{P}(y)))$.

Шаг 3. Разделение переменных. Так как в пределах области действия квантора можно заменить любую переменную на другую и от этого значение истинности формулы не изменится, можно преобразовать формулу так, чтобы каждый квантор имел свою собственную переменную, например вместо $(\forall x)(P(x) \to \exists x Q(x))$ пишется $(\forall x)(P(x) \to \exists y Q(y))$.

Пример.

$(\forall x)(\overline{P}(x) \vee \forall y(\overline{P}(y) \vee P(f(x,y))) \& \exists \omega(Q(x,\omega) \& \overline{P}(\omega)))$.

Шаг 4. Исключение кванторов существования. Рассмотрим формулу $(\forall x)(\exists x P(x,y))$. В этой формуле получается, что все выражение выполняется для любого y и некоторого x, который, возможно, зависит от y. Эту зависимость можно обозначить явно с помощью некоторой функции $g(y)$,отображающей каждое значение y в то x, которое существует. Такая функция называется ***сколемовской функцией***. Если заменить на нее x, то квантор существования можно убрать и переписать выражение в новом виде $\forall y P(g(y),y)$. Пусть требуется привести к сколемовской форме следующую формулу: $(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)$ $(P(x, y, z, u, v, w)$. В этой формуле левее (∃x) нет никаких кванторов всеобщности, левее $(\exists u)$ стоят $(\forall y)$ и $(\forall z)$, а левее $(\exists w)$ стоят $(\forall y)$, $(\forall z)$ и $(\forall v)$. Следовательно, можно заменить переменную x на константу a, переменную u – на двухместную $f(y, z)$, переменную w – на трехместную функцию $g(y, z, v)$. Таким образом, получаем следующую форму:

$(\forall y)(\forall z)(\forall v)(P(a, y, z, f(y, z), g(y, z, v))$. Общее правило исключения квантора существования состоит в замене каждого вхождения переменной, относящейся к квантору существования, на сколемовскую функцию, аргументы которой представляют собой те переменные, которые связаны с кванторами общности, в

область действия которых попал квантор существования. Функциональные символы, используемые в сколемовских функциях, должны быть новыми, т.е. они не должны встречаться ранее в формуле. Например, исключаем $\exists z$ из следующего выражения $\forall \omega Q(\omega) \to \forall x(\forall y(\exists z(P(x,y,z) \to \forall uR(x,y,u,z))))$, получим $\forall \omega Q(\omega) \to \forall x(\forall y(P(x,y,g(x,y)) \to \forall uR(x,y,u,g(x,y))))$. Если квантор существования не входит в область действия никакого квантора общности, то применяется сколемовская функция без аргументов, т.е. константа.

Пример.

$\forall x(\overline{P}(x) \vee (\forall y(\overline{P}(y) \vee P(f(x,y)) \,\&\, (Q(x,g(x)) \,\&\, \overline{P}(g(x))))$.

Шаг 5. Преобразование выражения в предваренную (префиксную) форму. Так как в формуле кванторы существования отсутствуют, то все кванторы общности можно переместить в начало формулы. Формула в таком виде называется ***формулой в предварительной форме***, цепочка кванторов перед ней – ***префиксом***, а следующее за ним бескванторное выражение – матрицей.

Пример.

$\forall x \forall y((\overline{P}(x) \vee \overline{P}(y) \vee P(f(x,y))) \,\&\, (\overline{P}(x) \vee Q(x,g(x))) \,\&\, (\overline{P}(x) \vee \overline{P}(g(x))))$

.

Шаг 6. Приведение матрицы выражения к форме КНФ. Известно, что любая логическая функция может быть представлена в виде КНФ. Для этого чаще всего используется метод эквивалентных преобразований.

Пример. Из алгебры логики известны следующие равенства

$$x_1 \vee x_2 x_3 = (x_1 \vee x_2)(x_1 \vee x_3) \quad ,$$

$$x_1 x_2 \vee x_3 x_4 = (x_1 \vee x_3 x_4)(x_2 \vee x_3 x_4) \, .$$

Применим их к полученному после шага 5 выражению:

$$\forall x \forall y(\underbrace{\overline{P}(x)}_{x_1} \vee \underbrace{((\overline{P}(y) \vee P(f(x,y)))}_{x_2} \,\&\, \underbrace{Q(x,g(x)) \,\&\, \overline{P}(g(x))))}_{x_3};$$

$$(\overline{P}(x) \vee \overline{P}(y) \vee P(f(x,y))) \,\&\, (\underbrace{\overline{P}(x)}_{x_1} \vee \underbrace{Q(x,g(x))}_{x_2} \,\&\, \underbrace{\overline{P}(g(x))}_{x_3}) =$$

$$= \forall x \forall y(\overline{P}(x) \vee \overline{P}(y) \vee P(f(x,y)) \,\&\, (\overline{P}(x) \vee Q(x,g(x))) \,\&\, (\overline{P}(x) \vee \overline{P}(g(x))).$$

Шаг 7. Исключение кванторов общности. Так как в выражении остались кванторы общности, а их порядок несущественен, то можно эти кванторы опустить, то есть удалить у формулы префикс.

Пример.

$$\overline{P}(x) \vee \overline{P}(y) \vee P(f(x,y)) \,\&\, (\overline{P}(x) \vee Q(x, g(x))) \,\&\, (\overline{P}(x) \vee \overline{P}(g(x)).$$

Шаг 8. Переход от выражения в виде КНФ к множеству предложений. Для этого требуется убрать все операции конъюнкции, а каждый дизъюнкт представить как отдельное предложение, т.е. перейти от выражения вида $x_1 \& x_2$ к выражению $\{x_1, x_2\}$, в котором каждый элемент – предложение.

Пример.

$$\{\overline{P}(x) \vee \overline{P}(y) \vee P(f(x,y)), \overline{P}(x) \vee Q(x, g(x)), \overline{P}(x) \vee \overline{P}(g(x))\}.$$

Шаг 9. Переименование переменных. Символы переменных должны быть изменены так, чтобы каждый появлялся не более чем в одном предложении. Этот процесс называется ***разделением переменных***.

Пример.

$$\{\overline{P}(x_1) \vee \overline{P}(y) \vee P(f(x_1,y)), \overline{P}(x_2) \vee Q(x_2, g(x_2)), \overline{P}(x_3) \vee \overline{P}(g(x_3))\}$$

.

Примечание. Когда резолюция используется в качестве правила вывода, то множество формул, которое будет использоваться в качестве исходного для доказательства, должно быть представлено в виде множества предложений.

При построении вывода с помощью метода резолюций следует учесть следующее:

1. Если во множестве присутствуют только предложения, содержащие предикаты без переменных, то вывод строится, как и в случае с высказываниями, т.к. логика высказываний – это частный случай логики предикатов.

2. Для того чтобы применить резолюцию к предложениям, содержащим переменные, необходимо иметь возможность найти такую подстановку, которая, будучи применена к родительским предложениям, приведет к тому, что они будут содержать дополнительные литералы.

Известно, что правило резолюций предполагает нахождение пары дизъюнктов, допускающих построение резольвенты. Для

дизъюнктов логики высказываний это очень просто. Для дизъюнктов логики предикатов процесс усложняется, так как дизъюнкты могут содержать функции, переменные и константы, а следовательно, их предварительно необходимо унифицировать.

Пример. Рассмотрим дизъюнкты:

C_1: $P(x) \vee Q(x)$,

C_2: $R(x) \vee \overline{P}(f(x))$.

Так как аргументы предиката P в C_1 и C_2. различны, то невозможно построить на их основе резольвенту, но если подставить $f(a)$ вместо x в C_1 и a вместо x в C_2, то исходные дизъюнкты примут вид

C_1': $P(f(a)) \vee Q(f(a))$,

C_2': $R(a) \vee \overline{P}(f(a))$

И, следовательно, можно получить резольвенту C_3': $Q(f(a)) \vee R(a)$.

В общем случае, подставив $f(x)$ вместо x в C_1, получим C_1'': $P(f(x)) \vee Q(f(x))$.

Предикат $P(f(x))$ в C_1'' и предикат $\overline{P}(f(x))$ в C_2 позволяют получить резольвенту C_3: $Q(f(x)) \vee R(x)$.

Таким образом, если подставлять подходящие термы вместо переменных в исходные дизъюнкты, можно порождать новые дизъюнкты. Отметим, что дизъюнкт C_3 из примера является наиболее общим дизъюнктом в том смысле, что все другие дизъюнкты, порожденные правилом резолюции, будут частным случаем данного дизъюнкта.

Учитывая введенное понятие унификации, можно распространить метод резолюций на исчисление предикатов. При унификации возникает одна трудность: если один их термов есть переменная x, а другой терм содержит x, но не сводится к x, унификация невозможна. Проблема решается путем переименования переменных таким образом, чтобы унифицируемые дизъюнкты не содержали одинаковых переменных.

Определение. Если два или более предиката (с одинаковым знаком) дизъюнкта C имеют наиболее общий унификаторg, то Cg – называется *склейкой* C.

Пример. Рассмотрим дизъюнкт $C= P(x)\vee P(f(y))\vee\overline{Q}(x)$. В этом дизъюнкте первый и второй предикаты имеют наиболее общий унификатор $g=\{f(y)/x\}$. Следовательно, $Cg=P(f(y))\vee \overline{Q}(f(y))$ есть склейка C.

Определение. Пусть C_1 и C_2 – два дизъюнкта, которые не имеют никаких общих переменных. Пусть L_1 и L_2 – два предиката в C_1 и C_2. Если L_1 и $\overline{L_2}$ имеют наиболее общий унификатор g, то дизъюнкт $(C_1g\backslash L_1g) \cup (C_2g\backslash L_2g)$ называется ***резольвентой*** C_1 и C_2.

Пример. Пусть $C_1= P(x)\vee Q(x)$ и $C_2=\overline{P}(a)\vee R(x)$. Так как x входит в C_1 и C_2, то заменим переменную в C_2 и пусть $C_2=\overline{P}(a) \vee R(y)$. Выбираем $L_1= P(x)$ и $L_2=\overline{P}(a)$. L_1 и L_2 имеют наиболее общий унификатор $g=\{a/x\}$. Следовательно, $Q(a)\vee R(y)$ – резольвента C_1 и C_2.

Пример. Пусть задано множество дизъюнктов

$\{P(x,f(A))\vee P(x,f(y))\vee Q(y),\ \overline{P}(z,f(A))\vee\overline{Q}(z)\}$, тогда

при $L_1=\{P(x,f(A))\}$ и $L_2=\{\overline{P}(z,f(A))\}$ полагаем $g=\{z/x\}$ и получаем резольвенту $P(z,f(y))\vee\overline{Q}(z)\vee Q(y)$, а при

$L_1=\{P(x,f(A)),P(x,f(y))\}$ и $L_2=\{\overline{P}(z,f(A))\}$ полагаем $g=\{z/x,A/y\}$ и получаем резольвенту $Q(A)\vee\overline{Q}(z)$.

Очевидно, что для двух дизъюнктов может существовать более одной резольвенты.

Переформулируем теперь уже известный алгоритм метода резолюций применительно к логике предикатов.

Шаг 1. Если в S есть пустой дизъюнкт, то множество S не выполнимо и алгоритм завершает работу, иначе переходим к шагу 2.

Шаг 2. Найти в исходном множестве S такие дизъюнкты или склейки дизъюнктов C_1 и C_2, которые содержат унифицируемые литералы $L_1\in C_1$ и $L_2\in C_2$. Если таких дизъюнктов нет, то исходное множество выполнимо и алгоритм завершает работу, иначе переходим к шагу 3.

Шаг 3. Вычислить резольвенту C_1 и C_2 , добавить ее в множество S и перейти к шагу 1.

Пример. Докажем с помощью метода резолюций справедливость следующих рассуждений.

- Некоторые руководители с уважением относятся ко всем программистам.
- Ни один руководитель не уважает бездельников.
- Следовательно, ни один программист не является бездельником.

Введем следующие предикаты:

C(x) – "x – руководитель";

P(x) – "x – программист";

B(x) – "x – бездельник";

U(x,y) – "x уважает y".

Тогда посылки, записанные в виде предикатных выражений будут выглядеть так:

- $\exists x(C(x) \,\&\, \forall y(P(y) \to U(x,y)))$;
- $\forall x(C(x) \to \forall y(B(y) \to \overline{U}(x,y)))$.

Заключение примет следующий вид:

- $\forall y(P(y) \to \overline{B}(y))$.

Преобразовав посылки и следствие, которое надо взять с отрицанием, в совокупность дизъюнктов, получим множество предложений, невыполнимость которого докажем, используя метод резолюций. Имеем

$$\{C(a), \overline{P}(y) \cup U(x,y), \overline{C}(x) \vee \overline{B}(y) \vee \overline{U}(x,y), P(b), B(b)\}.$$

1) $C(a)$

2) $\overline{P}(y) \cup U(x,y)$

3) $\overline{C}(x) \vee \overline{B}(y) \vee \overline{U}(x,y)$

4) $P(b)$

5) $B(b)$

6) $\overline{B}(y) \vee \overline{U}(a,y)$ (1,3) $/ s = \{a/x\}$

7) $U(a,b)$ (2,4) $/ s = \{b/y\}$

8) $\overline{B}(b)$ (6,7) $/ s = \{b/y\}$

9) *Пустой дизъюнкт* (5,8)

В заключение введем еще два определения.

Определение. ***Сколемовской формой*** предикатного выражения называется формула логики предикатов, в которой все вхождения переменных, у которых кванторы существования находятся в области действия кванторов общности, заменены на сколемовские функции.

Определение. ***Клаузальной формой*** называется такая сколемовская форма, матрица которой имеет вид КНФ. Любая сколемовская форма допускает эквивалентную ей клаузальную форму.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Кузнецов О.П., Адельсон-Вельский Г.Н. Дискретная математика для инженера. М.: Энергоатомиздат, 1988.
2. Тейз А., Грибомон П. Логический подход к искусственному интеллекту: от классической ложки к логическому программированию. М.: Мир, 1990.
3. Лорьер Ж.-Л. Системы искусственного интеллекта. М.: Мир, 1991.
4. Лихтарников Л.М., Сукачева Т.Г. Математическая логика: Курс лекций. Задачник-практикум и решения. СПб.: Лань, 1998.
5. Воротников С.М. Введение в математическую логику/ Комсомольский-на-Амуре гос.техн.ун-т. Комсомольск-на-Амуре, 1996.
6. Тэрано Т., Асаи К., Сугэно М. Прикладные нечеткие системы. М.: Мир, 1993.
7. Клини С.К. Математическая логика. М.: Мир,1973.
8. Мендельсон Э. Введение в математическую логику. М.: Наука, 1984.
9. Ершов Ю.Л., Палютин Е.А. Математическая логика. М.: Наука, 1987.
10. Раца М.Ф. Выразимость в исчислениях высказываний. Кишинев: Штиинца, 1991.
11. Мелихов А.Н., Чефранов А.Г. Применение ЭВМ для решения задач математической логики/ ТРТИ. Таганрог, 1988.
12. Лавров И.А., Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов. М.: Наука, 1984.

13. Потоцкий С.И., Гринченков Д.В., Гордиенко А.В. Дискретная математика/ ЮРГТУ(НПИ). Новочеркасск, 2001.

14. Метакидес Г., Нероуд А. Принципы логики и логического программирования. М.:Факториал, 1998.

15. Джордж Ф.Люгер. Искусственный интеллект. Стратегии и методы решения сложных проблем. М.:Вильямс, 2003.

16. Кондаков Н.И. Логический словарь-справочник. М.:Наука, 1975.

17. Новиков Ф.А. Дискретная математика для программистов. СПб.:Питер, 2001.

18. Гуц А.К. Математическая логика и теория алгоритмов: Учеб. пособие. Омск: Диалог-Сибирь, 2003.

19. Шапиро С.И. Решение логических и игровых задач. М.:Радио и связь, 1984.

20. Адаменко А., Кучуков А. Логическое программирование и Visual Prolog. СПб.:БХВ-Петербург, 2003.

21. Иванченко А.Н., Гринченков Д.В., Потоцкий С.И. Функциональное и логическое программирование./ ЮРГТУ(НПИ). Новочеркасск, 2006.

22. И.Братко Алгоритмы искусственного интеллекта на языке PROLOG. М.:Вильямс, 2004.

23. Игошин В.И. Математическая логика и теория алгоритмов, М.:Академия, 2008.

24. Аляев Ю.А. Тюрин С.Ф. Дискретная математика и математическая логика, М.:Академия, 2006.

ОГЛАВЛЕНИЕ

1. ТЕОРИЯ МНОЖЕСТВ 3
1.1. Основные понятия теории множеств 3
1.1.1. Множества, способы задания множеств 3
1.1.2. Основные операции над множествами и их свойства 4
1.2. Прямое произведение множеств 6
2. ОСНОВНЫЕ ПОЛОЖЕНИЯ БУЛЕВОЙ АЛГЕБРЫ . 7
2.1. Булева алгебра и ее применение 7
2.1.1. Определение булевой алгебры 7
2.1.2. Области применения булевой алгебры 8
2.1.3. Высказывания 8
2.2. Функции алгебры логики 9
2.2.1. Понятие функции и способы ее задания 9
2.2.2. Элементарные логические операции 11
2.2.3. Свойства основных логических функций 13
2.2.4. Задание функции формулой. Эквивалентные преобразования логических выражений 14
2.2.5. Двойственные функции 17
2.3.Специальные разложения логических функций 18
2.3.1. Конънктивная и дизъюнктивная нормальные формы. 18
2.3.2. Совершенно нормальные конъюнктивная и дизъюнктивная формы 20
2.4. Минимизация булевых функций 24
2.4.1.Понятие минимизации *24*
2.4.2 .Метод неопределенных коэффициентов 24
2.4.3. Метод Квайна-Мак Класки 26
2.4.4. Метод карт Карно 29
2.5. Полнота и замкнутость множества булевых функций.... 32
2.5.1. Понятие функционально полной системы 32
2.5.2. Алгебра Жегалкина 33
2.5.3. Замыкание и замкнутые классы 34
3. МАТЕМАТИЧЕСКАЯ ЛОГИКА 37
3.1. Общие сведения о формальных и аксиоматических системах 37
3.2. Исчисление высказываний 41
3.3. Методы, используемые для определения общезначимости формул исчисления высказываний 49

3.3.1. Алгоритм редукции 49
3.3.2. Метод резолюций..... 50
3.4. Логика предикатов..... 56
3.4.1. Основные понятия логики предикатов 56
3.4.2. Логика предикатов как формальная система 61
3.4.3. Определение значения истинности предикатных формул 64
3.4.4. Методы резолюций для логики предикатов …… 67
БИБЛИОГРАФИЧЕСКИЙ СПИСОК…….. 78

Учебное издание

Гринченков Дмитрий Валерьевич
Кущий Дарья Николаевна

ЛОГИКА ВЫСКАЗЫВАНИЙ И БУЛЕВЫ АЛГЕБРЫ

Учебное пособие

Редактор *Н.А.Юшко*
Подписано в печать 10.08.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 4,42. Уч.-изд.л. 4,75 . Тираж 100 экз. Заказ ____.

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова

Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

МИРОВЫЕ ИНФОРМАЦИОННЫЕ РЕСУРСЫ

Методические указания к лабораторным работам и самостоятельной работе по курсу

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 004.9 (076.5)
ББК 32.81

Рецензент – кандидат технических наук, доцент, доцент кафедры «Информационные и измерительные системы и технологии» **Скоба Андрей Николаевич**

Кузнецова А.В., Кущий Д.Н.

Мировые информационные ресурсы: методические указания к лабораторным работам и самостоятельной работе по курсу / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 24 с.

Издание содержит методические материалы по освоению навыков ведения деловой электронной переписки, поиска технической литературы в каталогах и информационно-поисковых системах, реализации фрагментов web-страниц. Приведена программа лабораторных работ, варианты заданий, а также контрольные вопросы для проверки знаний.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 004.9 (076.5)

1. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Изучение электронного этикета	2	Защита отчета	10-15.10
2	Знакомство с информационными ресурсами вуза	2	Защита отчета	10-15.10
3	Организация эффективного поиска в сети Интернет	2	Защита отчета	10-15.10
4	Создание сложных документов средствами текстового редактора	4	Защита отчета	15-20.11
5	Изучение элементов языка HTML	4	Защита отчета	15-20.12
6	Создание информационного ресурса для WWW с элементами интерактивности	4	Защита отчета	15-20.12

Лабораторная работа №1

Цель работы: закрепление навыков использования e-mail, изучение этикета проведения электронной переписки.

Программа работы

1. Создать ящик электронной почты на одном из бесплатных почтовых серверов.
2. Изучить правила электронного этикета (*netiquette*).
3. Составить электронное письмо с кратким рассказом о себе. Прикрепить к письму фотографию.
4. Отправить письмо преподавателю.
5. Ответить на контрольные вопросы. Оформить отчет.

Контрольные вопросы

1. Что такое электронная почта?
2. Какую информацию о себе необходимо ввести для регистрации на почтовом сервере? Какая информация не является обязательной?

3. Какова структура электронного почтового ящика?
4. Перечислите основные части электронного письма.
5. Услугами какого почтового сервера Вы воспользовались?
6. Какие ограничения на отправку электронных писем накладывает эта служба?
7. Какие еще почтовые службы Вы знаете (в том числе и на зарубежных серверах)?
8. Какие программы почтовых клиентов вы знаете?

Лабораторная работа №2

Цель работы: изучить информационные ресурсы, предоставляемые учебным заведением, освоить навыки поиска технической литературы.

Программа работы

1. Ознакомиться со структурой сайта библиотеки ЮРГПУ(НПИ) lib.npi-tu.ru
2. Провести поиск новых поступлений на дату по варианту(а). В случае отсутствия новых поступлений на эту дату изменить месяц и повторить поиск. Включить результаты в отчет.
3. Посетить электронную выставку издательства «Юрайт», включить в отчет сведения о понравившейся книге.
4. Провести поиск авторефератов и диссертаций в электронном каталоге в разделе «Диссертации» по ключевым словам (б). Включить результаты в отчет.
5. Провести поиск литературы по вариантам (в). Включить в поиск следующие ресурсы:
    а) электронный каталог книг НТБ ЮРГПУ(НПИ) (полный список книг);
    б) электронные сетевые ресурсы. Режим доступа: http://lib.npi-tu.ru/infresource/el_fulltext_res2:
    – коллекция электронно-библиотечной системы «КнигаФонд» (названия двух-трех книг). Режим доступа: www.knigafund.ru;

- коллекция электронно-библиотечной системы «Лань» (названия двух-трех книг). Режим доступа: http://e.lanbook.com;
- научная электронная библиотека Научная Электронная библиотека eLibrary (Библиографические сведения следующих источников (по 3 примера): учебник, статья, автореферат диссертации, монография, патент. Воспользоваться функцией расширенного поиска. Включить библиографические сведения в отчет. Режим доступа: http://www.elibrary.ru.
- коллекция журналов The Springer, найти и включить в отчет сведения о книге и статье на иностранном языке. Режим доступа: http://www.springerlink.com.

6. Ответить на контрольные вопросы. Оформить отчет.

Варианты заданий

№	Дата (а)	Ключевое слово (б)	Тема (в)
1	Февраль 2013	Метод	Программирование
2	Март 2013	Алгоритм	Сети ЭВМ
3	Апрель 2013	Механизм	Базы данных
4	Май 2013	инструментарий	Алгоритмизация
5	Сентярь 2013	Организация	Объектно-ориентированное программирование
6	Октябрь 2012	Метод	Автоматизированные системы
7	Ноябрь 2012	Алгоритм	Криптография
8	Декабрь 2012	Механизм	Электроника
9	Февраль2012	инструментарий	Операционные системы
10	Март 2012	Организация	Защита информации
11	Апрель 2012	Метод	Phyton
12	Май 2012	Алгоритм	Схемотехника
13	Сентябрь 2012	Механизм	Информационные ситсемы
14	Октябрь 2012	инструментарий	Архитектура ЭВМ
15	Ноябрь 2012	Организация	Компьютерная графика
16	Декабрь 2012	Метод	Вычислительная математика
17	Февраль 2014	Алгоритм	Теория алгоритмов
18	Март 2014	Механизм	Pascal
19	Апрель 2014	инструментарий	CMS -системы

20	Май 2014	Организация	CASE –средства
21	Сентябрь 2014	Метод	Организация ЭВМ
22	Октябрь 2014	Алгоритм	СУБД
23	Ноябрь 2014	Механизм	Искусственный интеллект
24	Декабрь 2014	инструментарий	Java
25	Февраль 2015	Организация	Программная инженерия

Контрольные вопросы

1. Приведите классификацию мировых информационных ресурсов.
2. Какие виды документированных информационных ресурсов вы изучили, выполняя лабораторную работу?
3. Приведите пример оформленной соответственно ГОСТ ссылки на книгу, статью и электронный ресурс.

Лабораторная работа №3

Цель работы: освоить навыки поиска информации в информационно-поисковых системах (ИПС).

Программа работы

1. Провести поиск информации в соответствии с вариантом в нескольких информационно-поисковых системах (выбрать 5, из них обязательно 1-2 англоязычные): Yandex, Rambler, Google, Поиск Mail.ru, Nigma, Bing, MSN Search и др. После осуществления поиска, для каждого найденного сайта на странице Excel заполнить таблицу вида:

Поисковая система 1	Степень совпадения 1	…	Поисковая система 5	Степень совпадения 5
Адрес сайта 1	Да	…	Адрес сайта 1	Нет
…		…	…	…
Адрес сайта 10	Частично	…	Адрес сайта 10	Да

Затем подсчитать число совпадений относительно той поисковой системы, которую Вы выбрали в качестве базовой.

Полученные результаты отобразить на диаграмме в MS Excel.

2. Ознакомиться с одним из перечисленных каталогов (Yahoo! (www.yahoo.com), Open Directory (dmoz.org), About (www.about.com), Апорт (www.aport.ru), Улитка (www.ulitka.ru), а также с каталогами ИПС (Яндекс-каталог и др.). Осуществить поиск информации, сделать вывод о возможностях поиска в каталогах.
3. Познакомиться с языком запросов поисковой системы Яндекс. Режим доступа: https://yandex.ru/support/search/query-language/search-context.xml. Исследовать эффективность поиска при применении логических операций. Сконструировать не менее 10 выражений, оценить эффективность поиска.

№	Запрос ИПС Yandex	Результаты поиска
1.	Мировые информационные ресурсы	Поиск информации, 9 млн ответов
2.	+Мировые+информационные+ресурсы	В документе встречаются все три слова (90 тыс. ответов)
3.	+Мировые+информационные+ресурсы -реферат	Исключить из результатов поиска документы, где есть слово «реферат»
…	…..	….

4. Изучить документные операторы языка запросов ИПС Яндекс. Режим доступа: https://yandex.ru/support/search/query-language/search-operators.xml. Исследовать эффективность поиска при применении операторов *«title:», «mime:», «lang:», «date: », «cat:»*. Описать результаты.
5. Ознакомиться с функциями расширенного поиска ИПС Яндекс. Привести твердые копии экранов.
6. Освоить функцию ИПС «Поиск в найденном».
7. Отобрать и скачать с указанием URL не менее 10 полнотекстовых релевантных ресурса по выданной теме (статьи, учебники, монографии, электронные учебники, web-порталы). Не использовать рефераты, статьи без авторства, неофициальную информацию. Использовать возможности elibrary (провести регистрацию для доступа к ресурсам).
8. Составить библиографический список выбранных ресурсов.

9. Найти информацию об авторе статьи по заданной теме. Описать действия по поиску человека и организации.
10. Ответить на контрольные вопросы. Оформить отчет.

Варианты заданий

1. Классификация мировых информационных ресурсов.
2. Способы хранения информации
3. Электронные библиотеки и ЭБС как часть мировой информационной инфраструктуры.
4. Информационно-поисковые машины.
5. Построение и организация глобальных сетей на примере Интернета.
6. Библиотечные ресурсы мира.
7. Архивные ресурсы мира
8. Социальные сети
9. Система DNS
10. Протоколы Интернета
11. Проблемы кроссбраузерности.
12. SEO (Поисковая оптимизация).
13. Браузеры. Назначение. Примеры. Настройка.
14. Скриптовые языки. Назначение. Характеристики. Сравнение.
15. Поставщики информации на рынке информационных услуг.
16. Модель OSI.Семиуровневая модель сетевого обмена. Задачи. Функции. Протоколы.
17. Образовательные ресурсы сети Интернет.
18. Службы Интернет.
19. Системы E-Learning.
20. Геоинформационные системы.
21. Облачные технологии.
22. Языки разметки гипертекста.
23. Формы и виды информационных ресурсов.
24. Информационные ресурсы предприятия.
25. Национальные информационные ресурсы.
26. Структура глобальных информационных ресурсов.

27. Плагиат. Этические нормы использования электронных информационных ресурсов. Системы проверки на плагиат.
28. Законодательные аспекты работы с информацией.
29. Проблемы информатизации общества.
30. Информационные ресурсы пользователя.
31. Физическая среда документальных информационных ресурсов.
32. Физическая среда электронных информационных ресурсов.
33. Экспертные системы как инструмент выработки недокументированных знаний.
34. Мировые ресурсы научно-технической информации.
35. Типологическая характеристика мировых информационных ресурсов.
36. Поиск знаний в сети Интернет.
37. Редакторы HTML-кода.
38. Вебинары. Организация. Этикет.
39. Grid-системы.
40. Криптография.
41. Массовая потребительская информация.
42. Участники рынка информационных услуг.
43. Государственные информационные ресурсы.
44. IP-адрес. Классы IP-адресов.
45. Принципы системы универсальных идентификаторов ресурсов URL. Составляющие формата URL. Основные адресные схемы формата URL.
46. Преступления в сфере компьютерных технологий, связанные с DDoS-атаками и ответственность за них.
47. Электронная коммерция.
48. Организация эффективного поиска в сети Internet.
49. Продвижение интернет-проектов.
50. Стеганография.

Контрольные вопросы

1. Дайте определения: каталог, ИПС, релевантность.
2. Опишите историю создания, характеристики, рейтинг поисковой системы по варианту (вариант определяется как остаток от

целочисленного деления на 5 порядкового номера в журнале посещаемости):

1) Яндекс; 2) Google; 3) Rambler; 4) Nigma; 5) Апорт.

3. Назовите отличия каталогов от поисковых машин.
4. Дайте определение метапоисковой системы.
5. Познакомьтесь с одной из систем ускоренного поиска. Приведите примеры.
6. Какими ИПС, каталогами и метапоисковиками Вы воспользовались при выполнении лабораторной работы?
7. Какие правила повышения эффективности поиска информации в глобальной сети были применены?

Лабораторная работа №4

Цель работы: изучение возможностей текстового процессора Microsoft Word при создании и форматировании документов.

Программа работы

1. Составить реферат по теме лабораторной работы №3. Выполнить следующие требования:
   1.1. Реферат должен иметь объем не менее 7-10 страниц. Шрифт *Times New Roman*, размер 14 пт, параметры страницы: слева 3см, сверху и снизу 2 см, справа 1,5 см.
   1.2. Обязательными элементами реферата должны быть:
   а) рисунок;
   б) схема;
   в) список маркированный;
   г) список нумерованный;
   д) таблица.
   1.3. Текст реферата разбить на логические части с уровнем вложенности не менее 2 (заголовки 1 и 2 уровней).
   1.4. Обязательны ссылки на список литературы по тексту реферата.
   1.5. Составить 10 контрольных вопросов по материалам реферата.

2. Изучить состав вкладок «Главная», «Вставка» и «Разметка страницы» для MS Office 2007 или MS Office 2013 (рис. 4.1 - 4.3). Возможен выбор другого текстового редактора.

Рис. 1. Вкладка «Главная» MS Word 2013

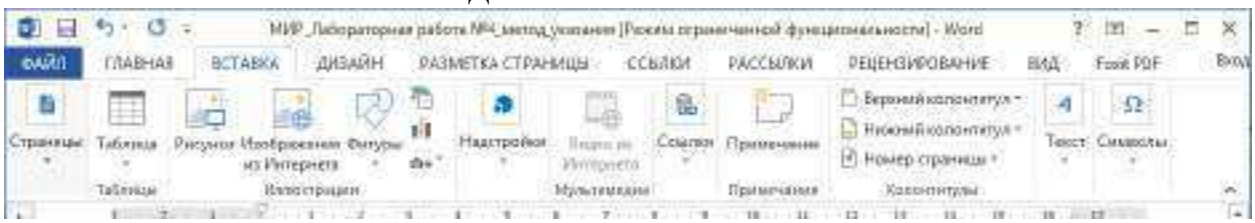


Рис.2. Вкладка «Вставка» MS Word 2013

Рис. 3. Вкладка «Разметка страницы» MS Word 2013

3. Освоить навыки набора и форматирования текста своего реферата.
4. Освоить замену символов в тексте по варианту (например, заменить непечатный символ «Разрыв строки» (↵) – символом «Знак абзаца» (¶); комбинацию из двух пробелов – одним; слово «Интернет» словом «Internet» и т.п.).
5. Выполнить вставки в текст различных элементов и объектов:
   5.1. Маркированный список.
   5.2. Нумерованный список.
   5.3. Таблица.
   5.4. Символы, например $\sum$, ∞.
   5.5. Формулы (по варианту).
   5.6. Колонтитулы (сверху – Ф.И.О. и тема реферата, снизу – номер страницы).
   5.7. Рисунок с подрисуночной подписью.
   5.8. Схему, вычерченную средствами векторного редактора MS Word (оформить в виде схемы любую классификацию).

5.9. Ссылки – сноски[1]

5.10. Объекты WordArt и SmartArt (в случае работы в MS Word).

6. Освоить установку стилей в тексте. Разбить текст на пункты и подпункты, пронумеровать их, установить стили: Обычный, Заголовок1, Заголовок2, Заголовок3. Сформировать оглавление документа. Продемонстрировать работу преподавателю.
7. Ответить на контрольные вопросы. Оформить отчет.

Варианты формул

1) $\dfrac{b+\sqrt{b^2+4ac}}{2a}-a^3c-b^{-2};\ \dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}+x.$

2) $\dfrac{\sin x+\cos y}{\cos x-\sin y}\cdot \operatorname{tg} xy;\ \dfrac{\cos^2 x}{\sin x}-xyz+\dfrac{ax^2+bx+c}{dx^3-f}.$

3) $\dfrac{3+\mathrm{e}^{y-1}}{1+x^2\left|y-\operatorname{tg} x\right|}\cdot \operatorname{tg} xy;\ F(x)=\begin{cases}4x^2+2x-19, \text{ если } x\ge -3{,}5;\\ -\dfrac{2x}{-4x+1}, \quad \text{если } x<3{,}5.\end{cases}$

4) $\ln\left|\left(y-\sqrt{|x|}\right)\left(x-\dfrac{y}{x+\dfrac{x^2}{4}}\right)\right|;\ 2\operatorname{ctg}(3x)-\dfrac{1}{12x^2+7x-5};$

5) $\dfrac{\cos x}{\pi-2x}+16x\cdot\cos(xy)-2;\ F(x)=\begin{cases}45x^2+5, \text{ если } x>3{,}6;\\ \dfrac{5x}{10x^2+1}, \text{ если } x\le 3{,}6.\end{cases}$

[1] Сноской называется дополнительный текст (пояснение, ссылка на источник, примечание редактора и т. п), помещаемый внизу полосы (колонки) и отделяемый от основного текста прямой линией.

6) $\left|x^2-x^3\right|-\dfrac{7x}{x^3-15x}$; $F(x)=\begin{cases}-3x+9, \text{ если } x>3;\\ \dfrac{x^3}{x^2+8}, \text{ если } x\le 3.\end{cases}$

7) $\dfrac{x+y}{y+1}-\dfrac{xy-12}{34+x}$; $\dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}$.

8) $e^x-x-2+(1-x)^x$; $F(x)=\begin{cases}1{,}2x^2-3x-9, \text{ если } x>3;\\ \dfrac{12{,}1}{2x^2+1}, \text{ если } x\le 3.\end{cases}$

9) $\left(1+\dfrac{1}{x^2}\right)^x-12x^2y$; $F(x)=\begin{cases}-x^2+3x+9, \text{ если } x\le 3;\\ \dfrac{x}{x^2+1}, \text{ если } x>3.\end{cases}$

10) $\left(\dfrac{x+1}{x-1}\right)^x+18xy^2$; $F(x)=\begin{cases}x^2+3x+9, \text{ если } x\le 3;\\ \dfrac{\sin x}{x^2-9}, \text{ если } x>3.\end{cases}$

11) $\dfrac{1+\sin\sqrt{x+1}}{\cos(12y-4)}$; $\dfrac{\cos^2 x}{\sin x}-xyz+\dfrac{ax^2+bx+c}{dx^3-f}$.

12) $(1-\operatorname{tg}x)^{\operatorname{ctg}x}+\cos(x-y)$; $\dfrac{1+\sin^4(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}+x$.

13) $e^x-\dfrac{y^2+12xy-3x^2}{18y-1}$; $\dfrac{\cos^2 x}{\sin x}-xyz+\dfrac{ax^2+bx+c}{dx^3-f}$.

14) $\sin\sqrt{x+1}-\sin\sqrt{x-1}$; $F(x)=\begin{cases}1{,}2x^2-3x-9, \text{ если } x>3;\\ \dfrac{12{,}1}{2x^2+1}, \text{ если } x\le 3.\end{cases}$

15) $2\mathrm{ctg}(3x)-\dfrac{\ln\cos x}{\ln\left(1+x^2\right)}$; $F(x)=\begin{cases} x^4+9, \text{ если } x<3{,}2; \\ \dfrac{54x^4}{-5x^2+7}, \text{ если } x\geq 3{,}2. \end{cases}$

16) $x\cdot\ln x+\dfrac{y}{\cos x-\dfrac{x}{3}}$; $F(x)=\begin{cases} -x^3+9, \text{ если } x\leq 13; \\ -\dfrac{3}{x+1}, \text{ если } x>13. \end{cases}$

17) $x\cdot\ln x+\dfrac{y}{\cos x-\dfrac{x}{3}}$; $F(x)=\begin{cases} 4x^2+2x-19, \text{ если } x\geq -3{,}5; \\ -\dfrac{2x}{-4x+1}, \text{ если } x<3{,}5. \end{cases}$

18) $x-10\sin x+\left|x^4-x^5\right|$; $F(x)=\begin{cases} 45x^2+5, \text{ если } x>3{,}6; \\ \dfrac{5x}{10x^2+1}, \text{ если } x\leq 3{,}6. \end{cases}$

19) $\mathrm{e}^x-\dfrac{y^2+12xy-3x^2}{18y-1}$; $\dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}+x.$

20) $\dfrac{1+\sin\sqrt{x+1}}{\cos(12y-4)}$; $\dfrac{\cos^2 x}{\sin x}-xyz+\dfrac{ax^2+bx+c}{dx^3-f}.$

Контрольные вопросы и упражнения

1. Какие возможности работы с документами предоставляет закладка «Вид»? Какие режимы отображения документов вы знаете?
2. Как вставить в текст разрыв страницы и разрыв раздела? Чем отличаются эти вставки?
3. Как установить параметры страницы?
4. Как в документе сделать несколько страниц альбомной ориентации? Сделать рисунок на отдельном листе и установить для него альбомную ориентацию страницы.
5. Что такое шаблон, как он составляется, для чего предназначен?

Лабораторная работа №5

Цель работы: знакомство с языком гипертекстовой разметки документов HTML, приобретение навыков реализации фрагментов web-страниц.

Программа работы

1. Ознакомиться с теоретическим материалом.
2. Освоить на основе фрагментов своего реферата элементы (конструкции) языка HTML, включающие **теги**:
   - верхнего уровня:
     HTML HEAD BODY;
   - заголовка документа:
     TITLE META COMMENT;
   - форматирования текста:
     P(с атрибутами) BR PRE
   - форматирования шрифта:
     B I U STRIKE SUB SUP
   - заголовков документов:
     H1 … H6
   - гиперссылок:
     A (с атрибутами);
   - работы с изображениями:
     IMG;
   - для списков:
     UL LI OL
   - для таблиц:
     TABLE CAPTIONTH TR TD
   - для фреймов:
     FRAMESET FRAME
3. Изучить атрибуты тегов, описать их действие.
4. Включить в отчет html-код и вид страничек в браузере.
5. Ответить на контрольные вопросы. Оформить отчет.

Контрольные вопросы и упражнения

1. Дайте определение следующих понятий: гипертекст, гиперссылка Web-сайт, URL-адрес в Интернете, фрейм, апплет, скрипт, браузер, элемент, тег, атрибут.
2. Как организовать элемент, являющийся гиперссылкой на другой файл? На фрагмент текущего файла?
3. Какие атрибуты элемента BODY вы изучили?
4. Самостоятельно изучить функции и правила оформления любых трех тегов из Приложения, отличных от приведенных в программе работы.

Лабораторная работа №6

Цель работы: знакомство с реализацией форм в *HTML*, приобретение навыков реализации сценариев на языке *JavaScript*.

Программа работы

1. Ознакомиться с теоретическим материалом.
2. Освоить создание форм на языке HTML. Отладить готовые примеры.
3. Изучить использование сценариев (скриптов) на языке *JavaScript*. Модифицировать под свою тему скрипт тестирования (5-7 вопросов).
4. Создать многостраничный информационный ресурс на основе своего реферата. Информация заголовков первого уровня – в отдельные файлы, заголовки второго уровня – гиперссылки внутри документа. Включить в состав ресурса контрольные вопросы, страницу с тестом, библиографический список.
5. Ответить на контрольные вопросы. Оформить отчет.

Контрольные вопросы и упражнения

1. Что такое интерактивность web-страниц? Как она реализуется?

2. Найти в сети Интернет и отладить готовый скрипт на языке JavaScript. Написать пояснения. Включить в отчет html-код и вид страничек в браузере.
3. Назовите программные продукты для автоматизации создания web-сайтов и форматирования html-кода.

2. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 71,1 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 1. Введение.** Сообщения. Виды сообщений.	5	[1,3]
2	**Тема 2. Основные понятия и сущность информационных ресурсов.** Роль и значение информационных ресурсов в развитии информационных технологий и в информатизации общества.	10	[1]
3	**Тема 4. Документированные информационные ресурсы.** Характеристика вторичных документов (информационных изданий). Правила оформления.	11,1	[1]
4	**Тема 5. Электронные информационные ресурсы.** Аппаратное и программное обеспечение для создания электронных документов.	15	[3]
5	**Тема 7. Система адресов Интернет** Семиуровневая модель сетевого обмена ISO/OSI. Задачи, функции, протоколы уровней модели. Регулирование и стандартизация в сети Интернет.	10	[3]
6	**Тема 8. Основы языка HTML.** Гиперссылки. Фреймы. Каскадные таблицы стилей CSS. Использование языка JavaScript в создании динамических элементов web-страниц.	20	[2]

Задания для самостоятельного изучения и конспектирования

Тема 1. Рассмотреть следующие вопросы:

- понятие сообщения;
- источники непрерывных и дискретных сообщений;
- характеристики источника дискретных сообщений;
- характеристики канала передачи дискретных сообщений;
- понятие сигнала. Виды сигналов.

Тема 2. Рассмотреть следующие вопросы:

- трактовка понятия информатизация общества в соответствии с Законом «Об информации, информационных технологиях и о защите информации»;
- этапы процесса информатизации общества;
- этапы развития вычислительных средств обработки информации;
- причины создания информационного продукта;
- основные черты информационного общества;
- понятие информационной культуры и ее основные аспекты.

Тема 4. Рассмотреть следующие вопросы:

- основные отличия первичных документов от вторичных;
- функции вторичных документов;
- документы, относящиеся к классу вторичных документов;
- классификация вторичных документ по степени аналитико-синтетической переработки содержащейся в них информации;
- формы распространения сигнальной информации;
- трактовка понятия реферат в международном стандарте ISO 214-1976 Documentation Abstracts for Publications and Documentation;
- виды реферативных изданий;
- правила оформления вторичных документов.

Тема 5. Рассмотреть следующие вопросы:

- электронные тексты. Системы оптического считывания и распознавания текстов и таблиц (Spot, FineReader и пр.). Оптические сканеры;
- технические и программные средства создания и обработки изображений (PhotoShop, CorelDraw и др.);
- понятие гипертекста и гипермедиа. Принципы функционирования;
- информационные и функциональные возможности мультимедиа-технологии;
- основные принципы функционирования мультимедийных программ;
- программное обеспечение для разработки мультимедийных продуктов.

Тема 7. Рассмотреть следующие вопросы:

- структурная схема модели OSI;
- схема взаимодействия компьютеров в базовой эталонной модели OSI;
- формирование пакета каждого уровня семиуровневой модели;
- функции уровней модели OSI;
- протоколы уровней модели OSI;
- характеристика деятельности Internet Society (ISOC);
- характеристика деятельности Internet Architecture Board (IAB);
- характеристика деятельности Internet Engineering Steering Group (IESG);
- характеристика деятельности Internet Engineering Task Force (IETF);
- правовое регулирование сети Интернет.

Тема 8. Рассмотреть следующие вопросы:

- понятие гиперссылки. Тег <a>. Создание текстовой гиперссылки и ссылки-изображения;
- виды интернет-адресов, применяемых в атрибуте href;
- создание якоря;
- понятие и структура фрейма, его достоинства и недостатки;

- стандарты Интернета и поддержка их браузерами;
- стилевые таблицы CSS;
- создание стилевых классов. Способы подключения стилевых таблиц;
- принципы каскадирования и группировки;
- использование псевдоклассов и псевдоэлементов;
- позиционирование элемента на странице при помощи CSS.
- кроссбраузерность CSS и CSS хаки;
- загрузка соответствующих CSS для разных устройств;
- обзор клиентских веб-технологий;
- инструментальные средства разработки клиентских веб-приложений;
- понятие языка сценариев. Основные сценарные языки;
- язык Javascript, история развития и версии;
- встраивание сценариев Javascript в HTML-документы;
- синтаксис языка Javascript;
- строки, переменные, массивы, выражения и операции. Массивы;
- управляющие конструкции. Условные операторы;
- операторы организации цикла;
- определение и вызов функций. Аргументы. Возврат значений;
- события в JavaScript. Обработчики событий. Объект event;
- создание сценариев с помощью функций и событий;
- объекты Date, String, Math;
- регулярные выражения. Проверка достоверности вводимых данных;
- использование методов и свойств объектов при создании сценариев;
- создание объектов. Конструктор;
- выявление и исправление ошибок.

Зачетные вопросы по курсу

1. Аспекты понятия информация.
2. Законодательная база в сфере информационных технологий.
3. Что такое сообщение?
4. Какие виды сообщений вы знаете?
5. Что такое данные?
6. Дайте определение понятия «информация».
7. Дайте определение понятия «информационные процессы».
8. Дайте определение понятия «информационные технологии».
9. Дайте определение понятия «данные» (применительно к IT).
10. Перечислите основные информационные процессы.
11. Что такое ресурсы?
12. Какие виды ресурсов вы знаете?
13. В чём отличие информационных ресурсов от других видов?
14. Чем представлены информационные ресурсы?
15. Какую информацию объединяют информационные ресурсы?
16. Дайте определение информационных ресурсов?
17. Дайте понятие мировых ИР.
18. Возникновение и развитие информационных ресурсов (6 фаз).
19. Проблемы роста ИР.
20. Какие классификационные признаки для ИР вы знаете?
21. Классификация ИР по признаку формы представления или фиксации информации.
22. Классификация ИР по признаку подлинности.
23. Классификация ИР по признаку тематической принадлежности.
24. Классификация ИР по признаку ограничения доступа.
25. Классификация ИР по признаку коммерциализации.
26. Какие виды недокументированных ресурсов Вы знаете?
27. Дайте определение понятия «индивидуальные знания специалистов».
28. Какие категории специалистов-носителей знаний Вы знаете?
29. Дать определение понятия «управление».
30. Что такое менеджмент?
31. Методы формирования и использования коллективных знаний.
32. Перечислите основные правила «мозгового штурма».

33. Специфика метода Delphi.
34. Изучить классификацию документированных текстовых ресурсов: рукописных и печатных.
35. Что такое тип данных? Назовите основные типы данных, использующиеся для хранения электронных ресурсов.
36. Какие смысловые единицы текстовых данных Вы знаете?
37. Что такое кодировочная таблица?
38. Какие кодировки, поддерживающие кириллицу, Вы знаете?
39. Понятие дискретизации.
40. Понятие квантования.
41. С какой частотой надо дискретизировать звуковой сигнал, чтобы он был слышим для человека.
42. Определение компьютерной графики.
43. Виды компьютерной графики.
44. Особенности растровых изображений. Форматы. Редакторы.
45. Особенности векторных изображений. Форматы. Редакторы.
46. Особенности фрактальных изображений. Форматы. Редакторы.
47. Цветовые модели. RGB. CMYK.
48. Кодировки цвета в RGB.
49. Определение размера графического файла.
50. Форматы представления видеоданных.
51. Определение кодека.
52. Программы для работы с видео.
53. Как рассчитать объем видеоинформации?
54. Основные сервисы Internet. Характеристики.
55. Определения: каталог, поисковая машина, метапоисковая система, сайт, портал, релевантность.
56. Информационно-поисковая система: определение, примеры.
57. Основные источники индексирования для документов WWW.
58. Режимы поиска. Приемы повышения эффективности поиска в сети Internet.
59. Операторы для составления запросов в поисковых системах.
60. Определение компьютерной сети. Типы сетей.
61. Обязанности системного администратора.
62. Сервер, клиент, клиент-серверная архитектура.

63. Понятие топологии сети. Основные топологии ВС. Достоинства и недостатки.
64. Понятие сетевых технологий. Примеры.
65. Сетевое оборудование.
66. Протоколы служб Internet.
67. Структура IP-адреса.
68. Домен, доменные адреса, пример.
69. Структура URL. Пример.
70. Определения: гипертекст, гиперссылка фрейм, апплет, скрипт, браузер, элемент, тег, атрибут.
71. Структура HTML-документа.

Контактная внеаудиторная работа

СРС – групповые консультации с преподавателем в течение семестра – 0,9 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Блюмин А.М. Мировые информационные ресурсы [Электронный ресурс]: учебное пособие / А.М. Блюмин, Н.А. Феоктистов. – Электрон. дан. – М.: Дашков и К, 2015. – 680 с. – Режим доступа: http://www.knigafund.ru/books/55384 – Загл. с экрана.
2. Савельев А. О. HTML5. Основы клиентской разработки [Электронный ресурс]: / Савельев А. О., Алексеев А. А.– Электрон. дан. – Национальный Открытый Университет «ИНТУИТ», 2016. – 272 с. – Режим доступа: http://www.knigafund.ru/books/177968 – Загл. с экрана.
3. Воробьев С. П. Информатика. Сетевые технологии : учеб. пособие / С.П. Воробьев. – Новочеркасск : изд-во НВВКУС, 2006. –103 с.
4. Георгица И.В. Информатика: учебно-методическое пособие к лабораторным работам для студентов направления «Информатика и вычислительная техника» / Юж.-Рос. гос. техн. ун-т (НПИ). – Новочеркасск: ЮРГТУ (НПИ), 2012. – 108 с.

Учебно-методическое издание

Кузнецова Алла Витальевна
Кущий Дарья Николаевна

Мировые информационные ресурсы

**Методические указания
к лабораторным работам
и самостоятельной работе по курсу**

Редактор *Н.А.Юшко*
Подписано в печать 9.12.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,39. Уч.-изд.л. 1,25. Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации

Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова

А. Н. Иванченко, А. А. Масленников, П. А. Иванченко

ОСНОВЫ ПРОГРАММИРОВАНИЯ (ЯЗЫК С)

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 004.438 (075.8)
ББК 32.973-18.1 Я 73
И 231

Рецензенты:

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Панфилов Александр Николаевич

кандидат технических наук, доцент, доцент кафедры
«Программное обеспечение вычислительной техники»
Кузнецова Алла Витальевна

Иванченко А.Н., Масленников А.А., Иванченко П.А.

И231 Основы программирования (язык С): учебное пособие / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова – Новочеркасск: ЮРГПУ (НПИ), 2016. – 88 с.

Приводится достаточно полное описание языка **C** с учетом последних изменений, введенных стандартом **C11**. Приемы программирования на языке **C** демонстрируются на примерах. Большое внимание уделяется форматам представления данных встроенных типов, операциям и технике работы с указателями. Подробно описываются вопросы создания и использования функций, в том числе и рекурсивных.

УДК 004.438 (075.8)

ОГЛАВЛЕНИЕ

1. Введение 5
1.1 Задачи, алгоритмы и программы 5
1.2 Языки программирования 11
1.3 Трансляция 14
2. Основы языка С 16
2.1 Общая характеристика и история языка С 16
2.2 Представление чисел. Системы счисления 17
2.3 Биты, байты и машинные слова 18
2.4 Беззнаковые числа и числа со знаком 19
2.5 Основные типы данных 19
2.6 Константы 22
2.7 Операции 24
2.7.1 Арифметические операции 25
2.7.2 Операции инкремента и декремента 25
2.7.3 Операции отношения 26
2.7.4 Логические операции 26
2.7.5 Поразрядные (битовые) операции 26
2.7.6 Операция присваивания и операции, совмещенные с присваиванием 27
2.7.7 Тернарная условная операция 28
2.7.8 Приоритет и порядок выполнения операций 29
2.8 Приведение типов данных 31
3. Операторы языка С 33
3.1 Простые операторы и блоки 33
3.2 Операторы простого выбора 34
3.3 Вложенные операторы выбора 34
3.4 Множественный выбор 36
3.5 Операторы повторения (цикла) 39
3.6 Оператор break 41
3.7 Оператор continue 42
3.8 Оператор goto и метки 42
4. Функции и структура программы 44
4.1 Функции и понятие модульности программы 44

4.2 Определение функции и структура программы 45
4.3 Функция main 47
4.4 Пример программы с модульной структурой 48
4.5 Рекурсия 50
5. Указатели и массивы 54
5.1 Адреса объектов программы 54
5.2 Адресная арифметика 55
5.3 Указатели 59
5.4 Указатели и аргументы функций 61
5.5 Массивы 62
5.6 Инициализация массивов 64
5.7 Динамические массивы 65
5.8 Символьные указатели, символьные массивы и строки символов 67
5.9 Массивы указателей 70
5.10 Многомерные массивы 71
5.11 Многомерные массивы нерегулярной структуры 72
5.12 Аргументы командной строки 74
5.13 Указатели на функции 75
6. Структуры и объединения 78
6.1 Структуры 78
6.2 Указатели на структуры и массивы структур 79
6.3 Инициализация структур 80
6.4 Анонимные структуры 80
6.5 Структуры и функции 81
6.6 Объединения 83
Список литературы 87

1. Введение

1.1 Задачи, алгоритмы и программы

Компьютеры (электронные вычислительные машины, ЭВМ) предназначены для решение задач, а результаты их решения используются человеком. Для того, чтобы компьютер мог решить задачу, ему нужна программа, которую человек (программист) составляет для него и записывает на *языке программирования* (ЯП). Ниже на рис. 1.1 схематично представлены роли человека и компьютера при решении задачи, а на рис. 1.2 – основные этапы решения задачи с помощью компьютера.

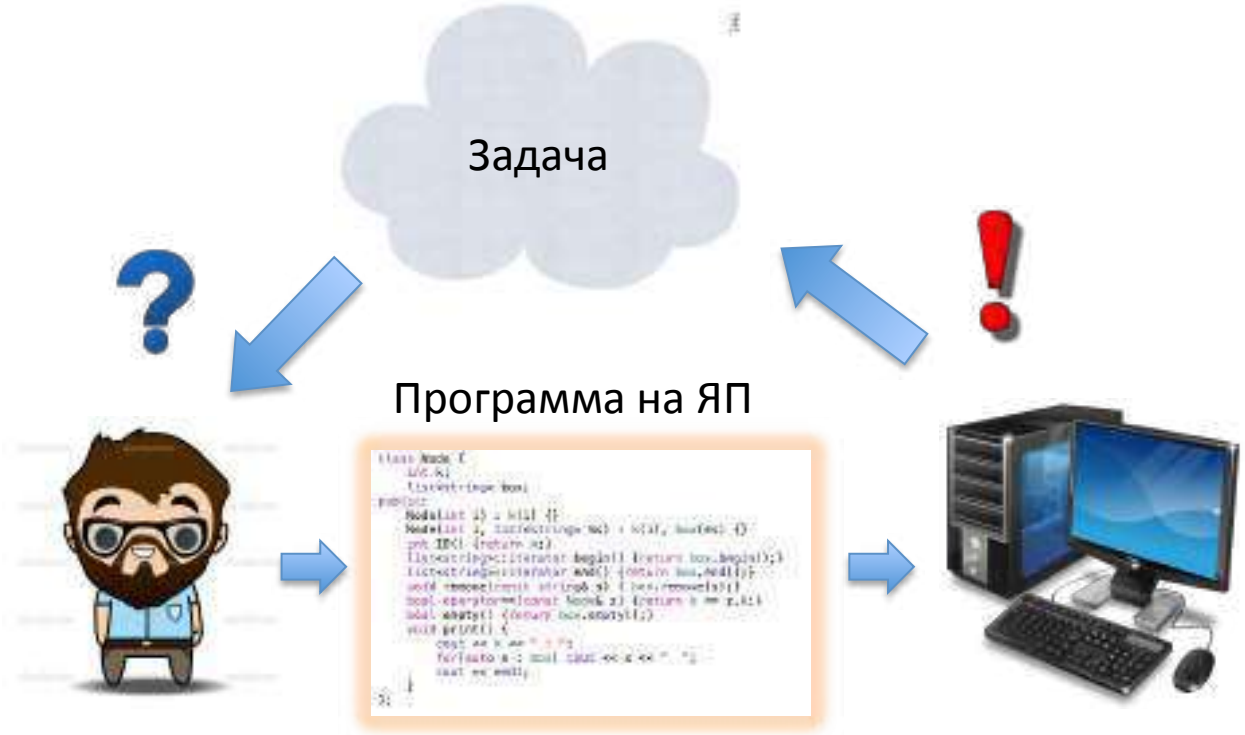


Рис. 1.1. Роли человека и компьютера при решении задачи

Первые компьютеры создавались исключительно для вычислений (computing), что нашло отражение в названиях «компьютер» и «ЭВМ». Вторым крупным применением компьютеров стали базы данных. Третье важнейшее применение компьютеров - управление всевозможными устройствами.

Сегодня компьютеры стали главным информационным инструментом на производстве, в офисе, дома и практически везде. Любая работа с информацией выполняется с помощью компьютера. Современные суперкомпьютеры используются для моделирования сложных физических и биологических процессов. Наиболее сложным применением компьютеров является искусственный интеллект.

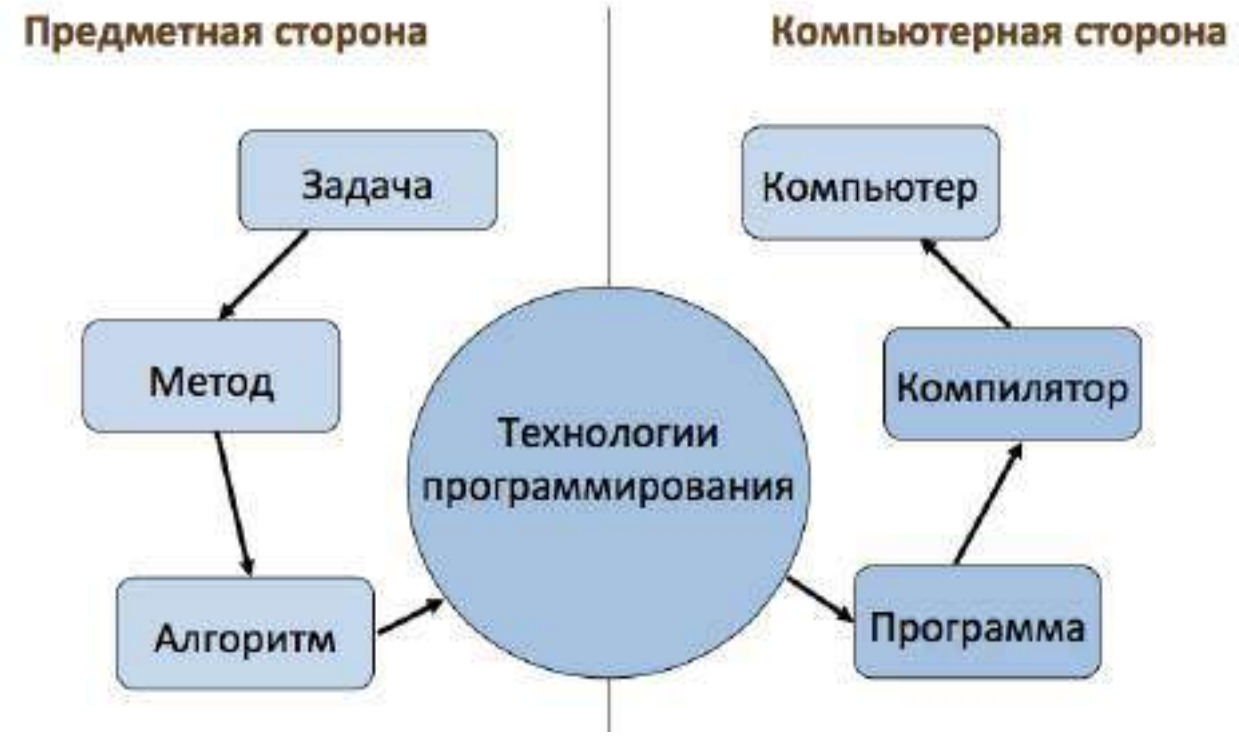


Рис. 1.2. Основные этапы решения задачи с помощью компьютера

Любая задача решается компьютером в виде последовательности элементарных шагов (вычислений), а эта последовательность определяется *алгоритмом*. Понятие алгоритма является основным для всей области компьютерных наук (Computer Science).

Слово «алгоритм» (algorithm) происходит от имени великого среднеазиатского ученого Абу Джофара Мухаммеда бен Муса аль Хорезмий (род. в 783 году в окрестностях Хивы). Из его математических работ до нашего времени дошли только две – алгебраическая и арифметическая. Термин алгоритм употреблялся в них для обозначения четырех арифметических операций, именно в таком значении он и вошел в некоторые европейские языки.

Постепенно значение слова алгоритм расширялось. К 1950 г. слово алгоритм чаще всего ассоциировалось с алгоритмом Евклида, который представляет собой процесс нахождения наибольшего общего делителя двух чисел. Этот алгоритм приведен в книге Евклида «Начала».

Существует несколько способов записи алгоритмов:

- на *естественном языке* (например, на русском языке) с использованием интуитивно понятных общепринятых обозначений (например, знаков арифметических операций),
- на *псевдокоде* – частично формализованном естественном языке и использованием некоторых конструкций языка программирования,
- в виде графической схемы (*блок-схемы*) с использованием стандартного набора блоков;
- на *формальном языке* (языке программирования), допускающем автоматический перевод записи алгоритма (программы) в машинные команды (такой перевод называется *трансляцией*, *компиляцией* или *интерпретацией*).

Ниже приведена запись алгоритма Евклида в трех формах: в виде блок-схемы (рис. 1.3), на частично формализованном естественном языке (в стиле, принятом Д. Кнутом в его знаменитой «библии программистов» [3]) и в форме программы на языке С.

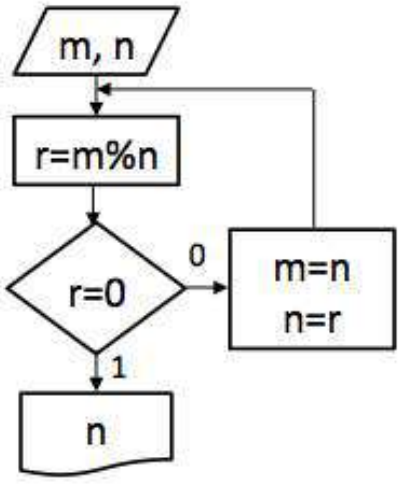


Рис. 1.3. Блок-схема алгоритма Евклида

Алгоритм Е (Алгоритм Евклида). Даны два целых положительных числа *m* и *n*. Требуется найти их наибольший общий делитель, т. е. наибольшее целое положительное число, которое нацело делит оба числа *m* и *n*.

Е1 [Нахождение остатка] Разделим *m* на *n*, и пусть остаток от деления будет равен *r* (очевидно, что $0 <= r < n$).

Е2 [Сравнение с нулем] Если $r = 0$, то выполнение алгоритма прекращается; *n* — искомое значение.

Е3 [Замещение] Присвоить $m \leftarrow n$, $n \leftarrow r$ и вернуться к шагу Е1. ♦

```
// Алгоритм Евклида
#include <stdio.h>
int main() {
  int m, n, r;
  printf("Введите m: "); scanf("%d", &m);
  printf("Введите n: "); scanf("%d", &n);
  if (m<n) {r=m; m=n; n=r;}
  while (1) {
    r=m%n;
    if (r==0) break;
    m=n; n=r;
  }
  printf("НОД = %d \n", n);
  return 0;
}
```

Результат работы:

```
Введите m: 23562
Введите n: 1105
НОД = 17
Program ended with exit code: 0
```

Пояснения к тексту программы на языке С:

1. Первая строка программы содержит комментарий – произвольный текст, предназначенный не компьютеру, а человеку, который читает программу. Комментарий начинается с пары символов “//” и продолжается до конца текущей строки.
2. Вторая строка также непосредственно не относится к алгоритму решения задачи. Это – команда подключения (`#include`) к программе функций ввода - вывода из

стандартной библиотеки, описание которой находится в файле `stdio.h` (файлы с расширением `h` называются заголовочными).

3. В третьей строке находится заголовок «главной» (`main`) функции, с которой начинается выполнение любой программы на языке С. Границы функции (ее «тело») обозначаются парой фигурных скобок: открывающей “`{`“ (после заголовка функции) и закрывающей “`}`“ в последней строке программы.
4. Перед именем функции `main` указано слово `int` (integer – целое число), которое означает, что после завершения работы функции `main` в операционную систему компьютера будет возвращено целое значение (код завершения программы). В примере функция `main` завершает свою работу командой (*оператором*) `return 0`, которая передает операционной системе значение `0` (успешное завершение). На скриншоте с результатом можно видеть сообщение операционной системы “Program ended with exit code: 0” (программа завершена с кодом 0).
5. Строки программы с 4 по 13 реализуют алгоритм Евклида. Подробное описание конструкций языка С, использованных для записи этого алгоритма, будет приведено далее в учебном пособии. Здесь отметим лишь библиотечные функции, позволяющие организовать диалог с человеком:
   - `printf` – вывод текстового сообщения на стандартное устройство вывода (экран монитора); эта функция позволяет не только выводить заданные ей символьные строки, но и преобразовывать числовые значения в текстовую форму по указанному формату и также выводить их на экран;
   - `scanf` – ввод текстовых строк со стандартного устройства ввода (клавиатура компьютера) и преобразование их к значению заданного формата; например, в операторе ввода `scanf("%d", &m)` вводимая с клавиатуры строка символов будет

преобразована к формату целого значения ("%d"), которое будет присвоено переменной m.

Современное значение слова алгоритм во многом аналогично таким понятиям, как рецепт, процесс, метод, способ, процедура, программа, но все-таки слово "algorithm" имеет дополнительный смысловой оттенок. Алгоритм — это не просто набор конечного числа правил, задающих последовательность выполнения операций для решения задачи определенного типа. Помимо этого, он имеет пять важных особенностей, или свойств:

1. **Конечность**. Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов. Алгоритм E удовлетворяет этому условию, потому что после шага E1 значение r меньше, чем n. Поэтому если $r \neq 0$, то в следующем цикле на шаге E1 значение n уменьшается. Убывающая последовательность положительных целых чисел имеет конечное число членов, поэтому шаг E1 может выполняться только конечное число раз для любого первоначально заданного значения n.
2. **Определенность**. Каждый шаг алгоритма должен быть точно определен. Действия, которые нужно выполнить, должны быть строго и недвусмысленно определены для каждого возможного случая.
3. **Ввод**. Алгоритм имеет некоторое (возможно, равное нулю) число входных данных, т. е. величин, которые задаются до начала его работы или определяются динамически во время его работы. Эти входные данные берутся из определенного набора объектов. Например, в алгоритме Евклида есть два входных значения: m и n, которые принадлежат множеству целых положительных чисел.
4. **Вывод**. У алгоритма есть одно или несколько выходных данных, т. е. величин, имеющих вполне определенную связь с входными данными. У алгоритма Евклида имеется только одно выходное значение — n, получаемое на шаге E2. Это наибольший общий делитель двух входных значений.
5. **Эффективность**. Алгоритм обычно считается эффективным, если все предписываемые им действия

достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени.

На практике нужны не просто алгоритмы, а хорошие алгоритмы в широком смысле этого слова. Одним из критериев качества алгоритма является *время*, необходимое для его выполнения; данную характеристику можно оценить по тому, сколько раз выполняется каждый шаг. Другими критериями являются *адаптируемость* алгоритма к различным компьютерам, его *простота*, *изящество* и т. д.

Часто решить одну и ту же проблему можно с помощью нескольких алгоритмов и нужно выбрать наилучший из них. Этими вопросами занимается важная область теоретического программирования – *анализ алгоритмов*. Предмет этой области состоит в том, чтобы для заданного алгоритма определить его рабочие характеристики.

Теория алгоритмов – это более широкая область, которая включает не только вопросы анализа алгоритмов, но и рассматривает формальные модели алгоритмов, вопросы существования или не существования эффективных алгоритмов решения определенных задач (алгоритмически неразрешимые проблемы), эквивалентность алгоритмов и др.

1.2 Языки программирования

На *языке программирования* записывают компьютерные программы, содержащие набор инструкций, позволяющих компьютеру выполнить некоторый вычислительный процесс, результатом которого будет решение определенной задачи, отыскание нужной информации, управление различными объектами и т. п.

Язык программирования – это формальная знаковая система, содержащая набор *лексических*, *синтаксических* и *семантических* правил, определяющих внешний вид программы и действия, которые выполнит компьютер под её управлением.

Язык программирования – это «посредник» между человеком и компьютером.

Языки программирования – это всегда некоторые модели (виртуальные вычислительные машины), позволяющие наиболее эффективно использовать возможности вычислительных средств, существенные для конкретных областей применения. Фактически, при написании программ ориентируются не на вычислительную машину как техническое устройство («железо»), а на некоторую абстрактную модель вычислительного устройства. Качество абстрактной модели, ее соответствие сути решаемых задач во многом определяет качество и эффективность проектирования с использованием этой модели.

Сложность задач, которые возможно решить с применением тех или иных концепций проектирования и выражающих их языков, непосредственно связана с уровнем абстракции как при постановке задачи, так и в ходе ее решения.

На нижнем уровне абстракции находятся *языки ассемблера*, позволяющие избежать необходимости программирования в машинном коде («родном» языке компьютера). Ассемблеры являются абстракцией компьютера (точнее говоря, компьютерной архитектуры), для которого были спроектированы. Это – машинно-ориентированные языки низкого уровня с командами, обычно соответствующими командам компьютера.

Языки *процедурного программирования* являются абстракциями ассемблеров и относятся к типу *языков высокого уровня* (ЯВУ). Это: Фортран, Алгол, КОБОЛ, PL/1, Паскаль, С и многие другие. Процедурный язык программирования дает возможность программисту определять каждый шаг процесса решения задачи. Используя процедурный язык, программист записывает языковые конструкции для выполнения последовательности алгоритмических шагов и представляет программу в виде множества процедур и функций.

Основной недостаток процедурных языков заключается в том, что достигаемый ими уровень абстракции все еще требует от программиста мышления в большей мере в терминах вычислительной машины, чем в терминах задачи, которую ему приходится решать.

Качество проектирования определяется в итоге тем, насколько удачно программисту удалось установить

соответствие между множеством понятий, характерных для решаемой задачи, и набором изобразительных средств языка, не всегда позволяющих адекватно отобразить эти понятия в рамках машинной модели.

Развитие абстрактных моделей не всегда означало переход на более высокие уровни абстракции представления компьютера как исполнительного устройства. Специфические особенности некоторых классов задач способствовали появлению альтернативных моделей, реализованных, например, в языках *функционального* и *логического* программирования.

Функциональное программирование – это программирование, основанное на функциях (в математическом понимании этого термина), без использования переменных (т.е. изменяемых объектов). Примеры языков функционального программирования: Lisp, Clojure, Haskell, F#, Scala, Erlang.

Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций. Самым известным языком логического программирования является Prolog. Другие языки логического программирования: Planner, Mercury, Oz, Fril.

В настоящее время развитие языков программирования идет в основном в направлении реализации *объектно-ориентированной (ОО) парадигмы* (концепции). Согласно этой парадигме разработчику предоставляются средства, позволяющие описать задачу и основную часть проекта в терминах предметной области, а не компьютерной модели. ОО-анализ начинается с исследования предметов реального мира, являющихся частью решаемой задачи.

Применение ОО-парадигмы не гарантирует автоматически повышения качества проектирования, но способствует ясному представлению данных и присущих им свойств и действий (поведения) на уровне программного кода. При этом программные структуры на формальном ОО-языке соответствуют модели описания задачи и способов ее решения на языке, естественном для данной предметной области. Эта особенность отличает ОО-языки от языков процедурного

программирования, которые моделируют действия компьютера. Примерами ОО-языков являются: C++, Java, C# и многие другие.

ОО-проектирование предоставляет разработчикам гибкий мощный универсальный инструмент, не связанный с каким-то определенным классом задач. Объектный подход является фундаментальной идеей всей компьютерной науки, применимой не только в программировании, но также в проектировании интерфейса пользователя, баз данных и архитектуры компьютеров.

1.3 Трансляция

Программа, написанная на языке высокого уровня, перед исполнением должна быть преобразована в программу на языке команд компьютера («машинном языке»). Такой процесс называется *трансляцией*, или *компиляцией*. По типу выходных данных различают два основных вида трансляторов:

- компилирующие окончательный выполнимый код;
- компилирующие интерпретируемый код, для выполнения которого требуется дополнительное программное обеспечение.

Окончательным выполнимым кодом являются приложения, реализованные как exe-файлы, dll-библиотеки, com-компоненты. К интерпретируемому коду можно отнести байт-код Java-программ, выполняемый посредством виртуальной машины JVM (Java Virtual Machine).

Языки, предполагающие формирование компилятором окончательного выполнимого кода, называются *компилируемыми языками*. К ним относятся языки C, C++, FORTRAN, Pascal. Языки, предполагающие формирование компилятором интерпретируемого кода, называются *интерпретируемыми языками*. К таким языкам относятся языки Java, C#, LISP, Perl, Prolog.

В большинстве случаев код, получаемый в результате процесса трансляции, формируется из нескольких программных модулей (программный модуль – это определенным образом

оформленный код на ЯВУ). Процесс трансляции в этом случае может выполняться как единое целое – компиляция и редактирование связей, или как два отдельных этапа – сначала компиляция в объектные модули, а затем вызов редактора связей, создающего окончательный код. Последний подход более удобен для разработки программ. Он реализован в трансляторах языков С и С++.

Объектный код, создаваемый компилятором, представляет собой область данных и область машинных команд, имеющих адреса, которые в дальнейшем "согласуются" *редактором связей* (его еще называют компоновщиком или линковщиком). Редактор связи размещает в едином адресном пространстве все по отдельности откомпилированные объектные модули и статически подключаемые библиотеки.

2. Основы языка C

2.1 Общая характеристика и история языка C

Язык C является компилируемым процедурным языком общего назначения, поддерживающим структурное программирование, области видимости переменных, статическую типизацию и рекурсию. Конструкции языка C обеспечивают эффективное отображение их на типичные машинные команды, поэтому C может использоваться в приложениях, которые ранее были реализованы на языке ассемблера, включая операционные системы, а также различное программное обеспечение от суперкомпьютеров до встраиваемых систем.

Язык C был разработан Деннисом Ритчи в 1969-73 гг. в AT&T Bell Laboratories, и впоследствии на нем была переписана операционная система Unix. С тех пор он стал одним из наиболее широко используемых языков программирования всех времен, с компиляторами от различных поставщиков, доступным для большинства существующих компьютерных архитектур и операционных систем.

На язык C оказал значительное влияние его предшественник, язык B, созданный Кэном Томпсоном, а этот язык, в свою очередь, является последователем языка BCPL (Basic Combined Programming Language), который был создан в 1969 г. Мартином Ричардсом в рамках проекта "Комбинированный язык программирования" в Кэмбриджском университете в Лондоне.

Язык C был стандартизирован Американским национальным институтом стандартов (ANSI) в 1989 г. и впоследствии Международной организацией по стандартизации (ISO). В настоящее время актуальным является стандарт 2011 г., поэтому текущую версию языка часто называют C11 (рис. 2.1).

Многие более поздние языки имеют прямые или косвенные заимствования из C, в том числе C++, D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Verilog (язык описания аппаратных средств), и командная оболочка

UNIX C shell (csh). Эти языки имеют общую синтаксическую схожесть с языком С и используют многие из его структур управления и другие синтаксические особенности. Язык С также используется в качестве промежуточного языка для других языков и для построения стандартных библиотек и исполнительных систем (runtime systems) для языков высокого уровня, таких как Cpython.

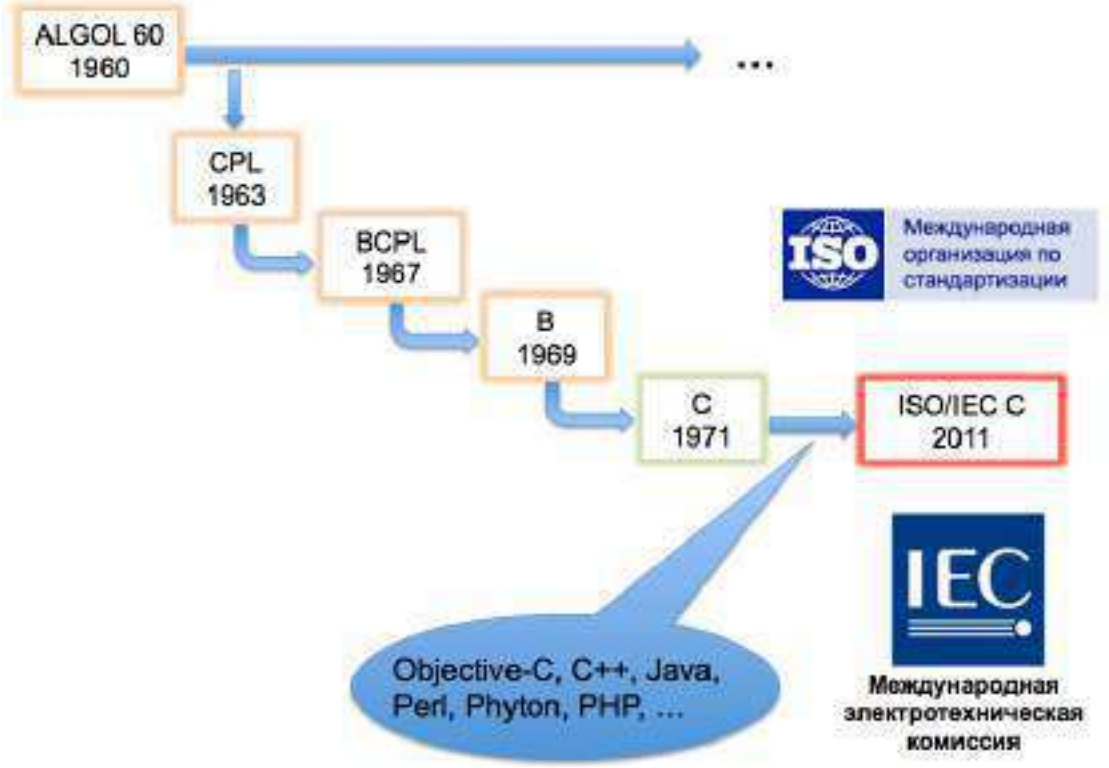


Рис. 2.1. История языка С

Несмотря на «солидный» возраст язык С и сегодня входит в TOP5 языков программирования (www.tiobe.com/tiobe-index). Полная справочная информация по языку С, включая его последние версии, содержится на сайте: http://en.cppreference.com/w/c .

2.2 Представление чисел. Системы счисления

Система счисления – это символический метод записи чисел, позволяющий представить числа с помощью письменных знаков (цифр). Наибольшее распространение получили позиционные системы счисления (ПСС), в которых положение

цифры в записи числа однозначно определяет то количество, которое представляет эта цифра, например:

$$274 = 2*10^2+7*10^1+4*10^0=200+70+4$$

Важнейшими характеристиками любой ПСС являются: *основание* системы счисления и *набор знаков*, используемых для записи чисел. Так, основанием десятичной СС является 10, а набором цифр – множество {0,1,2,3,4,5,6,7,8,9}.

Для компьютеров используется двоичная СС с основанием 2 и цифрами 0 и 1. Пример числа в двоичной СС:

$$1010_2 = 1*2^3+0*2^2+1*2^1+0*2^0=8+2=10$$

Также распространены восьмеричная СС (основание=8, цифры={0,1,2,3,4,5,6,7}) и шестнадцатеричная СС (основание=16, цифры={0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f}) .

Примеры:

$$375_8 = 3*8^2+7*8^1+5*8^0 = 3*64+7*8+5 = 192+56+5 = 25310$$

$$1f7a1_6 = 1*16^3+15*16^2+7*16^1+10*16^0 = 1*4096+15*256+7*16+10 = 4096+3840+112+16 = 805810$$

2.3 Биты, байты и машинные слова

Любая информация в компьютере представляется комбинацией 0 и 1 и наименьшей порцией данных является 1 бит, способный хранить либо 0, либо 1. Синонимом термина “бит” является “двоичный разряд”. На практике наряду с битами используют боле крупные единицы – байты и машинные слова. Размер 1 байта = 8 битов, а размер машинного слова зависит от архитектуры компьютера. Для 32-разрядной архитектуры 1 машинное слово = 32 бита, или 4 байта.

Изображать значения, хранящиеся в байтах и машинных словах, в двоичном коде громоздко, поэтому используют восьмеричную или шестнадцатеричную запись, например:

$$10101011_2 = 253_8 = ab1_6$$

В языке С допускается записывать целые числа в десятичной, восьмеричной и шестнадцатеричной системах счисления:

- десятичная запись – обычная математическая запись числа (но первой цифрой не может быть 0);

- восьмеричная запись – начинается с 0, например: 0253;
- шестнадцатеричная запись – начинается с 0x (или с 0X), например: 0xab.

2.4 Беззнаковые числа и числа со знаком

В языке С допускается использовать целые числа без знака (они всегда ≥ 0) или числа со знаком (положительные и отрицательные). Для беззнаковых чисел все разряды используются для представления величины числа. Например, если все разряды байта равны 1, то мы имеем: 0xff = 255.

Для чисел со знаком правила более сложные. У положительных чисел старший бит (крайний левый) всегда равен 0, а остальные биты выражают само число, например: 0x7f = 127 и это будет максимальное положительное число, которое можно записать в 1 байт. Отрицательные числа записываются в так называемом *дополнительном коде* и у них всегда старший бит будет равен 1. Чтобы получить число в дополнительном коде, нужно взять сначала соответствующее положительное число, а затем применить к нему две операции:

- инвертировать двоичную запись этого числа;
- добавить к полученному коду 1.

Пример: 116 = 0x74 = 01110100; -116 = 10001011 + 1 = 10001100 = 0x8c; 116 + (–116) = 0x74 + 0x8c = 0

Обратное преобразования отрицательного числа в соответсвующее положительное число выполняют по тому же алгоритму. Например, однобайтный код 0xd9 – это отрицательное число в дополнительном коде. Его двоичное представление: 11011001. После инвертирования и добавления 1 получим: 00100111 = 0x27 = 39. Следовательно, исходное число равно -39.

2.5 Основные типы данных

Фундаментальные объекты данных, с которыми работает программа – это *переменные* и *константы*. Переменная имеет

имя, тип и текущее значение. Имя для переменной выбирается программистом произвольно с учетом следующих ограничений:

- имя должно начинаться с буквы и может состоять из букв и цифр (знак «подчеркивание» считается буквой);
- буквы в верхнем и нижнем регистре различаются;
- в качестве имен не разрешается использовать зарезервированные слова языка (`for`, `while`, `int` и т.п.).

Замечание. Кроме имен переменных в программах могут использоваться символические имена (символические константы), которые определяются с помощью директивы препроцессора `#define` и задают правило подстановки:

```
#define имя текст_для_подстановки
```

Например: `#define STEP 20`

Каждая переменная относится к определенному типу данных. Такая принадлежность устанавливается при объявлении переменной, например, так: `int k;`

Все переменные одного типа данных имеют одинаковый размер, одинаковое «внутреннее устройство» и одинаковый набор операций, которые можно к ним применить.

В языке С (с учетом последних стандартов языка) определены следующие типы данных:

- `bool` - булевский (логический) тип, способный хранить значения 0 («ложь») или 1 («истина»); этот тип данных появился в последних стандартах языка С и для его использования к программе нужно подключить заголовочный файл `stdbool.h`,
- `char` - символьный тип (как правило, размером 1 байт), способный хранить один символ,
- пять стандартных знаковых целых типов:
  - `signed char` - однобайтовое целое число,
  - `short int` - «короткое» целое число,
  - `int` - целое число стандартной длины (для данной архитектуры),
  - `long int` - «длинное» целое число,
  - `long long int` - «сверхдлинное» целое число.

- пять соответствующих им беззнаковых целых типов: `unsigned char`, `unsigned short int`, `unsigned int`, `unsigned long int`, `unsigned long long int`,
- три стандартных типа вещественных чисел:
  - `float` - число одинарной точности,
  - `double` - число двойной точности,
  - `long double` - число максимальной точности.

Кроме того, в стандарте C99 была добавлена возможность указывать точный размер целочисленных переменных (в битах) с помощью новых типов, определенных в заголовочном файле `stdint.h`. Некоторые из новых типов:

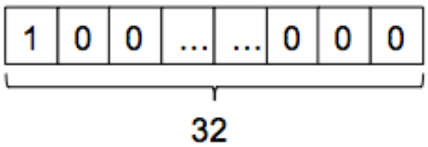
- четыре знаковых целых типа:
  - `int8_t` - однобайтовое целое число,
  - `int16_t` - двухбайтовое целое число,
  - `int32_t` - четырехбайтовое целое число,
  - `int64_t` - восьмибайтовое целое число,
- четыре беззнаковых целых типа: `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`.

Пример. Пусть для компьютера с 32-разрядной архитектурой текущее значение некоторой переменной таково:

или в формате шестнадцатеричной константы: `0xffffffff` .

Тогда, если считать, что это переменная типа `signed int`, то будем иметь число -1. Если же считать, что это переменная типа `unsigned int`, то получим 2^{32}-1 = 4 294 967 295.

Другой пример:



или в формате шестнадцатеричной константы: `0x80000000` .

Для типа `signed int` это будет -2 147 483 648, а для типа `unsigned int`: 2^{31} = 2 147 483 648.

Любая переменная в программе на языке С должна быть объявлена до первого обращения к ней. При объявлении переменной указывается её тип, имя и, возможно, начальное значение. Переменные одного типа можно объявлять списком:

```
int lower, upper, step;
char c, line[1000];
char esc = '\\';
int limit = MAXLINE + 1;
float eps = 1.0e-05;
```

К объявлению любой переменной можно добавить модификатор `const`, который придаст этой переменной свойство "read only" (только для чтения). Попытка присвоить такой переменной (или элементу массива) нового значения вызовет ошибку компиляции:

```
const double e = 2.71828182845905;
const char msg[] = "Warning: ";
```

Важно! Модификатор `const` нельзя использовать для объявления неинициализированных переменных.

2.6 Константы

Константы, в отличие от переменных, не имеют имён. Запись константы в виде цепочки символов одновременно является и значением этой константы. Для каждого типа данных существуют свои правила записи констант.

Целочисленные константы записывают, как правило, с помощью десятичных цифр, хотя возможна их запись в восьмеричной и шестнадцатеричной системах:

`123, 67543, 037, 07777, 0xabf7, 0XFFFF, …`

Допускается использование суффиксов `l` или `L` (long), а также `u` или `U` (unsigned):

`123456789L, 0XFUL` (это просто число 15).

Вещественные константы содержат десятичную точку, а также степенную часть (экспоненту) или и то, и другое:

`1234.5, 1e-2, 12.3E1, …`

Такие константы имеют тип `double`, однако имеется возможность записи констант `float` и `long double`:

`1.2f, 23.45F, 1e-1L, …`

В стандартной библиотеке языка С имеется заголовочный файл `limits.h`, в котором, среди прочего, содержатся определения множества «полезных» символических констант, характеризующих особенности представления данных различных типов в конкретной архитектуре. Например:

```
#define INT_MAX 2147483647
#define INT_MIN (-INT_MAX-1)
#define ULONG_MAX 0xFFFFFFFFUL
```

Символьная константа – это целое число, записываемое в форме одиночного символа в одинарных кавычках: `'x'`, `'0'` и т.п. Значением символьной константы является числовое значение кода символа и поэтому они могут участвовать в арифметических операциях. Например, значение константы `'0'` – это число 48.

Для записи «неграфических» символов используют управляющие последовательности:

`\a` – подача звукового сигнала,
`\b` – возврат назад и затирание,
`\f` – прогон страницы,
`\n` – конец строки,
`\r` – возврат каретки,
`\t` – горизонтальная табуляция,
`\v` – вертикальная табуляция,
`\\` - обратная косая черта,
`\?` – вопросительный знак,
`\'` – одинарная кавычка,
`\"` – двойная кавычка,
`\000` – восьмеричное число,
`\xhh` – шестнадцатеричное число,
`\0` – нулевой байт.

Строковая константа – это последовательность из нескольких символов, заключенная в кавычки:

`"Это строка"`, `""` (а это – пустая строка)

Фактически строковая константа является массивом символов (массивом типа `char`), причем в конце массива автоматически добавляется нулевой байт (признак конца строки)

и поэтому длина массива всегда на 1 больше размера строки. Пустая строка имеет нулевую длину и занимает в памяти 1 байт!
Внимание! Нельзя путать символьные константы и строковые константы из одного символа. Например: 'x' и "x" – это совершенно разные вещи!

Константы перечислимого типа. Язык С дает возможность программисту создавать собственные типы данных, представляющие фиксированные множества целочисленных значений, причем для каждого значения задается имя, которое будет играть роль константы данного типа. Такие типы называются *перечислениями*. Например, перечисления `boolean`, `months` и `escapes` можно задать так:

```
enum boolean {FALSE, TRUE};
enum months {JAN=1, FEB, MAR, APR, MAY, JUN,
JUL, AUG, SEP, OCT, NOV, DEC};
enum escapes {BELL='\a', BACKSPACE='\b',
TAB='\t', NEWLINE='\n', RETURN='\r'};
```

Допустимы *анонимные перечисления*, например:

```
enum {FALSE, TRUE};
```

Это перечисление просто определяет две целых константы.

Числовые значения константам – членам перечисления задаются либо явно путем их *инициализации*, либо автоматически по правилу: значение текущей константы на 1 больше значения предыдущей (первая константа в списке получает значение 0).

После определения перечисления можно описывать переменные перечислимого типа, например:

```
enum months x;
```

Такое объявление означает, что переменной `x` можно присваивать константы типа `months` или другие переменные этого же типа.

2.7 Операции

Операции определяют преобразования, которые можно производить с объектами программы. В зависимости от количества операндов, участвующих в одной операции, операции делятся на унарные (одноместные), бинарные (двуместные), и

тернарные (трехместные). В результате выполнения операции всегда получается некоторое новое значение, которое можно использовать для последующих вычислений и, возможно, изменяется значение одного из операндов. Из операций конструируются более сложные объекты программы – выражения.

2.7.1 Арифметические операции

В языке С определены 5 бинарных арифметических операций: сложение (+), вычитание (-), умножение (*), деление (/) и получение остатка от деления (%). Операндами арифметических операций могут быть данные любых типов, за исключением операции получение остатка от деления. Эта операция не допускает операнды вещественного типа.

2.7.2 Операции инкремента и декремента

Инкремент и декремент – это унарные операции, позволяющие увеличить или уменьшить на 1 текущее значение операнда, который может быть переменной любого типа. Каждая из этих операций существует в двух формах – постфиксной и префиксной.

Постфиксная форма

`x++` означает то же самое, что и `x=x+1;`

`x--` означает то же самое, что и `x=x-1;`

`y=x++;` означает то же самое, что и `y=x; x=x+1;` то есть значение `x` используется *до* его увеличения на 1.

`y=x--;` означает то же самое, что и `y=x; x=x-1;` то есть значение `x` используется *до* его уменьшения на 1.

Префиксная форма

`++x` означает то же самое, что и `x=x+1;`

`--x` означает то же самое, что и `x=x-1;`

`y=++x;` означает то же самое, что и `x=x+1; y=x;` то есть значение `x` используется *после* его увеличения на 1.

`y=--x;` означает то же самое, что и `x=x-1; y=x;` то есть значение `x` используется *после* его уменьшения на 1.

2.7.3 Операции отношения

В языке С определены 6 бинарных операций отношения: больше (>), больше или равно (>=), меньше (<), меньше или равно (<=), равно (=), не равно (!=). Результат их выполнения – это обычное целое число 0 или 1, которое может использоваться как логическое (булевское) значение: «ложь» – 0, «истина» – 1.

2.7.4 Логические операции

В языке С определены 3 логические операций: две бинарные – «логическое и» (&&) и «логическое или» (||) и одна унарная операция – «логическое отрицание» (!). Выражения, содержащие логические операции, вычисляются слева направо, и вычисление прекращается, как только установлена гарантированная истинность или ложность результата, например:
`(3==3) || ((c=getchar())=='y')` – второе выражение не вычисляется.
`(0) && ((x=x+1)>0)` - второе выражение не вычисляется.

2.7.5 Поразрядные (битовые) операции

Эти операции рассматривают свои операнды (или операнд) как битовую строку и операция выполняется с каждым битом (парой битов) независимо. Операндами не могут быть вещественные значения. Всего в языке С имеется 6 битовых операций, из которых 3 операции работают с двумя битовыми строками:

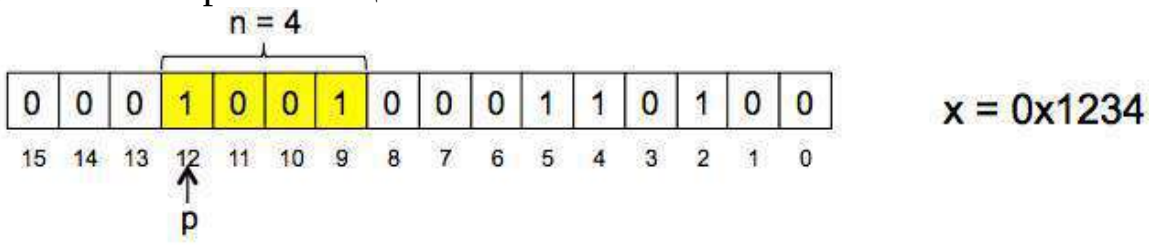
- «и» – поразрядное логическое умножение (&);
- «или» – поразрядное логическое сложение (|);
- «исключающее или» – поразрядное логическое сложение по модулю 2 (^);

и 3 операции работают с одной битовой строкой:

- унарная операция «инверсия битов» (~);
- бинарная операция «сдвиг влево» (<<);

– бинарная операция «сдвиг вправо» (>>).

Пример. Следующая программа извлекает n бит из переменной x, начиная с p-й позиции.




```
int main() {
  int p = 12, n = 4;
  unsigned short x = 0x1234;
  printf("before = %x \n", x);
  printf("after = %x \n", (x>>(p+1-n)) & ~(~0<<n));
  return 0;
}
```

Результат работы:

```
before = 1234
after = 9
Program ended with exit code: 0
```

2.7.6 Операция присваивания и операции, совмещенные с присваиванием

В языке С операция присваивания существенно отличается от аналогичной операции в других языках. Присваивание `a = b` действительно является операцией (а не оператором), т.е. после завершения этой операции получается некоторое значение (в данном примере – значение переменной `b`), которое может быть использовано для последующих вычислений. Следовательно, присваивание может применяться внутри выражений, наряду с другими операциями.

Пример. Иллюстрация операции присваивания:

```
int main() {
  int a = 12, b = 3, c;
  printf("result = %d \n", c=2*b+(b=a)-5);
  return 0;
}
```

Результат работы:

```
result = 13
Program ended with exit code: 0
```

Кроме обычного присваивания существует ещё 10 модификаций, в которых присваивание совмещается с какой-либо бинарной операцией, например, `k+=2` эквивалентно `k=k+2`, или ещё: `x>>=1` эквивалентно `x= x>>1` и т.д. Полный перечень бинарных операций, совмещаемых с присваиванием:

`+ - * / % << >> & ^ |`

Пример. Подсчет количества единичных битов в двух байтах

```
int main() {
  unsigned short x = 0x1234;
  int b;
  for (b=0; x!=0; x >>= 1) if (x & 01) b++;
  printf("the number of bits = %d \n", b);
  return 0;
}
```

Результат работы:

```
the number of bits = 5
Program ended with exit code: 0
```

2.7.7 Тернарная условная операция

Во многих алгоритмах часто встречается конструкция вида:

```
if (условие) x = <выраж1>;
else x = <выраж2>;
```

Иными словами, переменная `x` получает значение одного из двух выражений, в зависимости от истинности условия. В языке С для компактной записи такого фрагмента существует специальная тернарная условная операция:

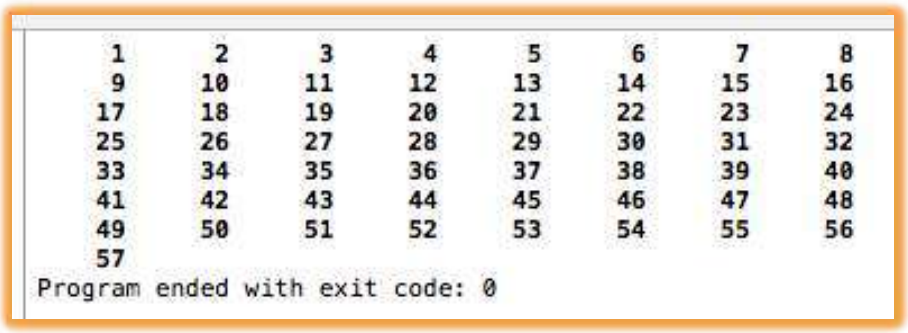
```
x = условие ? <выраж1> : <выраж2>
```

Пример. Иллюстрация условной операции:

```
#define N 57
int main() {
  int i, x[N];
  for (i=0; i<N; ++i) x[i]=i+1;
  for (i=0; i<N; ++i)
   printf("%6d%c",x[i],(i%8==7||i==N-1) ? '\n':' ');
```

```
  return 0;
}
```

Результат работы:



2.7.8 Приоритет и порядок выполнения операций

Последовательность выполнения операций в сложном выражении определяется их приоритетом, а в случае равенства проиритетов у подряд идущих операций – установленным порядком выполнения одноприоритетных операций (слева направо или справа налево), табл. 2.1.

Таблица 2.1

№	Операции	Порядок
1	`()  []  ->  .`	→
2	`!  ~  ++  --  +  -  *  &  (тип)  sizeof`	←
3	`*  /  %`	→
4	`+  -`	→
5	`<<  >>`	→
6	`<  <=  >  >=`	→
7	`==  !=`	→
8	`&`	→
9	`^`	→
10	`|`	→
11	`&&`	→
12	`||`	→
13	`?:`	←
14	`=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=`	←
15	`,`	→

Операции, находящиеся в одной строке таблицы, принадлежат одной группе, т.е. имеют одинаковый приоритет и одинаковый порядок выполнения (слева направо или наоборот). Все операции распределены по 15 группам, группа 1 имеет наивысший приоритет, а группа 15 – наиболее низкий приоритет.

В группу 1 входят операции: «`( )`» – вызов функции, «`[ ]`» – доступ к элементу массива, «`->`» – обращение к элементу структуры через указатель, «`.`» – обращение к элементу структуры через имя структуры. Описание этих операций будет приведено ниже.

В группу 2 входят унарные операции: «`+`», «`-`» – унарные «плюс» и «минус», «`*`» – «разыменование» указателя, т.е. обращение к объекту, на который он указывает, «&» – получение адреса объекта, «`(тип)`» – преобразование типа операнда к заданному типу, «`sizeof`» – определение размера объекта.

Операции из групп 3 – 14 были рассмотрены выше. В последнюю, 15-ю группу, входит операция «запятая» `(,)` - бинарная операция, имеющая вид: `выражение1,выражение2`. Действие операции заключается в последовательном вычислении выражений, а её результатом является значение второго выражения.

Пример. Иллюстрация операции «запятая» (подсчет суммы элементов массива)

```
int main() {
  int i, s, x[] = {1, 3, 5, 7, 9};
  for(i=0, s=0; i<sizeof(x)/sizeof(x[0]); s+=x[i++]);
  printf("s = %d\n", s);
  return 0;
}
```

Результат работы:

```
s = 25
Program ended with exit code: 0
```

Пример. Иллюстрация к приоритету операций: чему равно значение переменной `y` после вычисления выражения:

```
y*=x+=--k<<++i
  5   4   2  3  1
```

```
int main() {
  int x = 4, y = 6, k = 5, i = 1;
  y*=x+=--k<<++i;
  printf("y =%d, x =%d, k =%d, i =%d\n", y, x, k, i);
   return 0;
}
```

Результат работы:

```
y = 120, x = 20, k = 4, i = 2
Program ended with exit code: 0
```

2.8 Приведение типов данных

Приведение типа – это преобразование значения одного типа в значение другого типа. Различают явное и неявное приведения типов. При *явном приведении* с помощью унарной операции `(тип)` указывается тип, к которому необходимо преобразовать значение выражения, расположенного справа от этой операции. При *неявном приведении* преобразование происходит автоматически, по правилам, заложенным в языке.

Пример. Иллюстрация приведения типов

```
int main() {
  int x=32768, y=65535;
  short sx=32768, sy=65535; char c1=32768, c2=65535;
  unsigned z = 0xffffffff;
  float fz = 0xffffffff;
  printf("%d   %d   %d   %f   %d\n", x, (short)x, sx,
(float)x, c1);
  printf("%d   %d   %d   %f   %d\n", y, (short)y, sy,
(float)y, c2);
  printf("%u   %f   %f   %f\n", z, (float)z, (double)z,
fz);
  return 0;
}
```

Результат работы:

```
32768  -32768  -32768  32768.000000  0
65535  -1  -1  65535.000000  -1
4294967295  4294967296.000000  4294967295.000000
4294967296.000000
Program ended with exit code: 0
```

Корректными преобразования типов, не вызывающими искажения значений или потери точности, являются приведения более «узких» типов к более «широким».

Если же значение более «широкого» типа приводится к более «узкому» типу, возможно искажение значения или потеря точности, о чем выдает *предупреждение (warning)* компилятор.

Явные преобразования типов всегда происходят по «инициативе» программиста, а инициатором неявных преобразований выступает компилятор в тех случаях, когда при выполнении операции ожидается один тип, а фактически присутствует другой тип.

Замечание. В некоторых случаях неявное преобразование бывает просто неосуществимым «по определению». Например, в выражении `3.0%2.0` значения `3.0` и `2.0` не преобразуются автоматически к целому типу, т.к. операция `%` применима только к целочисленным значениям и поэтому компилятор выдаст *сообщение об ошибке (error)*.

3. Операторы языка С

3.1 Простые операторы и блоки

Операторы – это элементарные конструкции языка, порождающие некоторый фрагмент вычислительного процесса. Простейший оператор получается, если в конце любого выражения поставить символ ; (точка с запятой). Такая конструкция называется *простым оператором*:

```
z = foo(x+y); x += y; 4*5;
```

Простые операторы в программе могут располагаться последовательно, друг за другом:

```
temp = x+y; z = foo(temp);
```

Последовательность операторов любого вида можно заключить в фигурные скобки { и }. В этом случае получится *составной оператор*, или, иначе – *блок*, который синтаксически эквивалентен простому оператору и компилируется как самостоятельная конструкция. После закрывающей скобки точка с запятой не ставится. Блок может быть пустым:

```
{} // это пустой блок
```

Внутри блока можно размещать объявления переменных:

```
{ int temp = x+y; z = foo(temp); }
```

Такие переменные будут локальными; они создаются при входе в блок и исчезают при выходе из блока. За пределами блока локальные переменные невидимы (точнее, они просто не существуют). Локальные переменные размещаются по ходу выполнения программы в общей области памяти, которая называется *стек*.

Блоки могут быть вложенными, при этом глубина вложенности не ограничивается:

```
{ int temp = x+y;
  z = foo(temp);
  { float temp2 = x*y;
    z += bar(temp2);
  }
}
```

3.2 Операторы простого выбора

В простейшем случае требуется выбрать для выполнения определенный оператор, если некоторое условие истинно (т.е. его значение отлично от 0). Это записывается так:

```
if (условие) оператор
```

Например: `if(x%2) y += x/2;`

То есть, если `x` – нечетно, то к текущему значению `y` прибавляется `x/2`, в противном случае ничего не делается.

Если при выполнении условия нужно выполнить последовательность операторов, необходимо использовать блок.

В другом случае требуется выбрать для выполнения один из двух операторов, в зависимости от некоторого условия:

```
if (условие) оператор1 else оператор2
```

Например: `if (a>b) c=a; else c=b;`

На рис. 3.1 приведены блок-схемы для этих двух форм оператора простого выбора

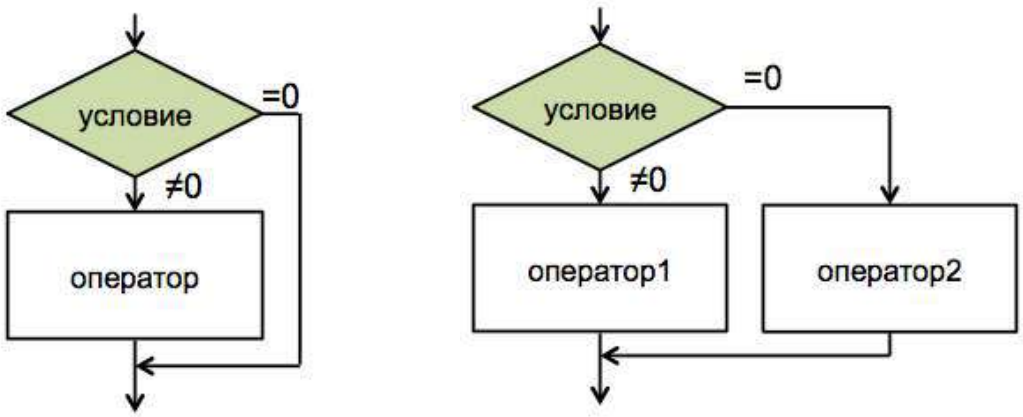


Рис. 3.1. Две формы оператора простого выбора

3.3 Вложенные операторы выбора

Если в качестве исполняемого оператора в операторе выбора используется другой оператор выбора, возникает ситуация вложенности операторов выбора. Пусть, например, требуется найти d = max(a, b, c). Блок-схема алгоритма для решения этой задачи приведена на рис. 3.2.

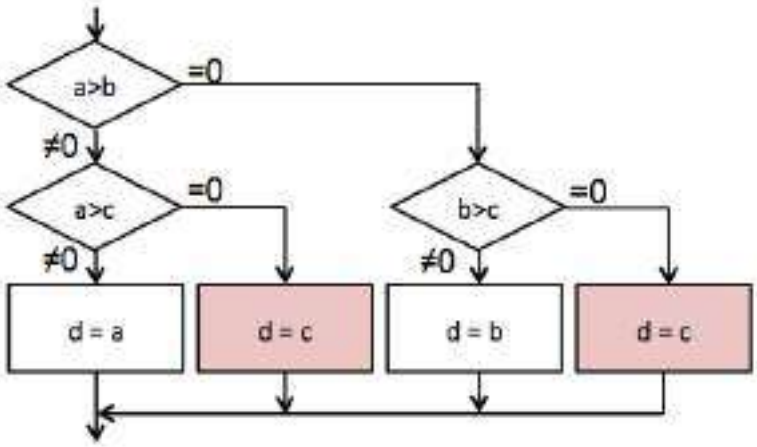


Рис. 3.2. Алгоритм выбора максимального из трех чисел

Соответствующая этой блок-схеме программа:

```
int main() {
  int a = 4, b = 15, c = 5, d;
  if (a>b)
    if (a>c) d = a;
    else d = c;
  else
    if (b>c) d = b;
    else d = c;
  printf("max = %d\n", d);
  return 0;
}
```

Результат работы: `max = 15`

Усовершенствуем алгоритм, убрав повтор оператора `d=c` (рис. 3.3) и составим по этому алгоритму программу.

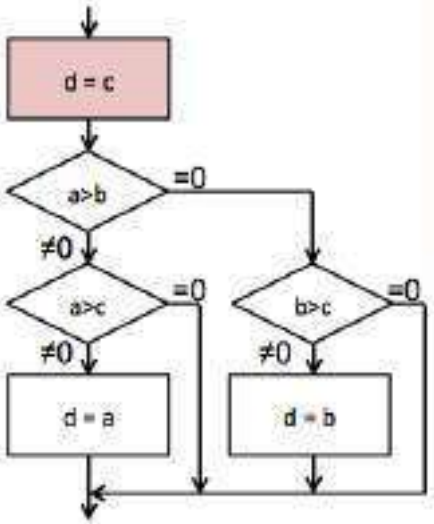


Рис. 3.3. Экономный алгоритм выбора максимального из трех чисел

```
int main() {
  int a = 4, b = 15, c = 5, d = c;
  if (a>b)
    if (a>c) d = a;
  else
    if (b>c) d = b;
  printf("max = %d\n", d);
  return 0;
}
```

Эта программа даст следующий результат: `max = 7`

Как видим, результат неправильный, так как нарушены правила записи вложенных операторов выбора. Точнее говоря, нарушено правило: «Каждый `else` относится к ближайшему к нему расположенному выше по тексту `if`».

Правильная программа будет иметь вид (здесь пришлось использовать конструкцию блока, чтобы указать, что оператор `if (a>c)` не имеет `else`-части):

```
int main() {
  int a = 4, b = 15, c = 5, d = c;
  if (a>b)
    {if (a>c) d = a;}
  else
    if (b>c) d = b;
  printf("max = %d\n", d);
  return 0;
}
```

Эта программа даст правильный результат: `max = 15`

3.4 Множественный выбор

В некоторых случаях требуется выбрать один из нескольких операторов, причем для каждого из них задано свое условие выбора. Например, выбор может быть задан следующей блок-схемой (рис. 3.4). Соответствующий этой блок-схеме фрагмент программы будет иметь вид:

```
if (c=='+') r = a + b;
else if (c=='-') r = a - b;
else if (c=='*') r = a * b;
else if (c=='/') r = a / b;
else printf("error!\n");
```

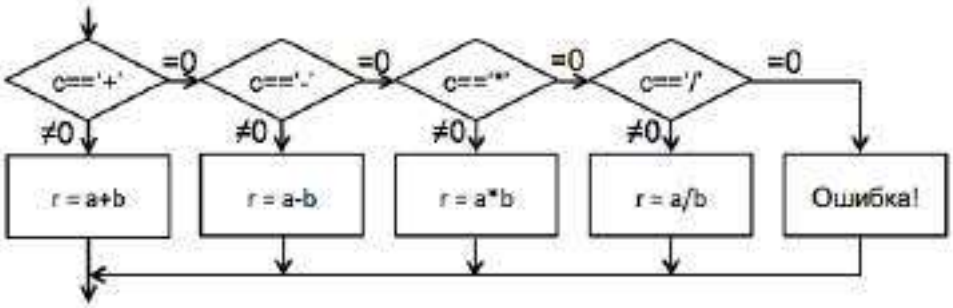


Рис. 3.4. Алгоритм выбора одного из нескольких вариантов

Однако для подобных алгоритмов удобнее использовать специальный оператор множественного выбора `switch`:

```
switch(c) {
  case '+': r = a + b; break;
  case '-': r = a - b; break;
  case '*': r = a * b; break;
  case '/': r = a / b; break;
  default: printf("error!\n");
}
```

Заметим, что если из приведенного фрагмента программы убрать операторы `break`, то будет реализован совершенно другой алгоритм (рис. 3.5).

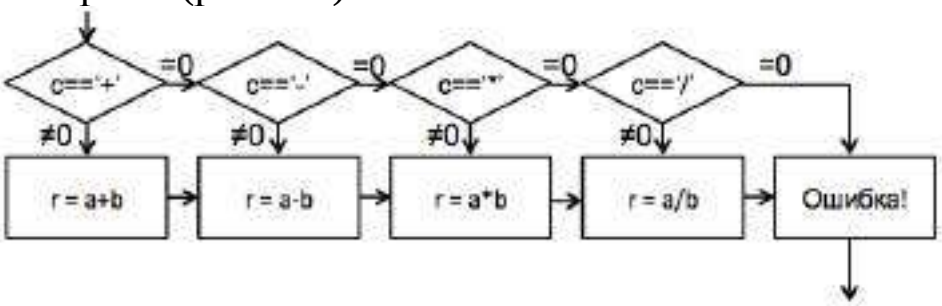


Рис. 3.5. Алгоритм работы оператора switch без break

Общий вид оператора `switch`:

```
switch (выражение) {
  case константа-выраж1: операторы
  case константа-выраж2: операторы
  …
  default: операторы
}
```

Действие оператора: вычисляется значение выражения в скобках после слова `switch` и затем это значение последовательно сравнивается со значениями константных выражений в блоках `case`. Если произошло совпадение

значений, выполняются операторы соответствующего блока `case`. Как правило, последним оператором в каждом блоке `case` является оператор `break` («прервать») и поэтому по завершению выбранного `case`-блока вычислительный процесс выходит за пределы оператора `switch`. Если совпадения значений не произошло ни для одного из `case`-блоков, то выполняются операторы блока `default` (если он есть) или вычислительный процесс сразу выходит за пределы оператора `switch`.

Пример. Следующая программа вводит с клавиатуры последовательность символов, пока не встретит символ '$'. После ввода программа подсчитывает частоту вхождения символов по группам: цифры, пробельные символы и прочие.

```
int main() {
  int c, i, nwhite=0, nother=0, ndigit[10];
  for (i = 0; i < 10; i++) ndigit[i] = 0;
  while ((c = getchar()) != '$') {
    switch (c) {
      case '0': case '1': case '2': case '3': case
'4':
      case '5': case '6': case '7': case '8': case
'9':
        ndigit[c-'0']++; break;
      case ' ': case '\n': case '\t':
        nwhite++; break;
      default: nother++; break;
    }
  }
  printf("digits =");
  for (i = 0; i < 10; i++) printf(" %d", ndigit[i]);
  printf(", white space = %d, other = %d\n",
    nwhite, nother);
  return 0;
}
```

Результат работы:

```
oikjnn   97yjfnvd;vln;l
ddfkafgjggj   333s
digits = 0 0 0 3 0 0 0 1 0 1, white space = 8, other = 29
```

3.5 Операторы повторения (цикла)

Во многих алгоритмах встречается элементарная конструкция, приведенная на следующей блок-схеме (рис. 3.6).

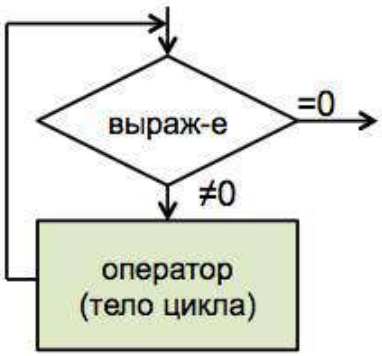


Рис. 3.6. Алгоритм работы оператора while

Она соответствует многократному (циклическому) повторению оператора (или нескольких операторов в блоке), пока некоторое выражение не станет равным нулю (получит значение «ложь»). Это записывается так:

```
while(выражение)
  оператор
```

Такая конструкция называется *циклом с предусловием*.

Пример. Подсчет количества символов в строке.

```
int main() {
  int i = 0;
  char s[] = "qwerty";
  while (s[i++]) ;
  printf("number of characters(%s) = %d\n", s, --i);
  return 0;
}
```

Результат выполнения:

```
number of characters(qwerty) = 6
```

В языке С существует еще одна форма цикла с предусловием, более сложная по сравнению с простейшей формой `while`. Эта форма называется циклом `for` и она реализует следующий алгоритм (рис. 3.7).

В цикле `for` дополнительно участвуют еще два выражения, одно из которых вычисляется однократно, до проверки условия, а

второе – многократно, после всех операторов, выполняемых в цикле. Цикл `for` записывается так:

```
for(выраж1; выраж2; выраж3)
  оператор
```

В круглых скобках любое из выражений можно опустить, но точки с запятой обязательно должны присутствовать. Если опущено `выраж2`, то оно считается равным 1. Поэтому конструкция: `for(;;);` является бесконечным циклом (как, впрочем, и конструкция `while(1);`).

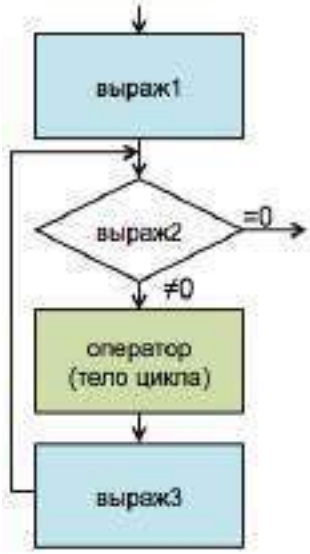


Рис. 3.7. Алгоритм работы оператора for

Пример. Следующая программа (парсер) преобразует символьную запись числа в строке `s` в само число:

```
int main() {
  int i, n, sign;
  char s[] = "  -347ab";
  for (i = 0; isspace(s[i]); i++) ;
  sign = (s[i] == '-') ? -1 : 1;
  if (s[i] == '+' || s[i] == '-') i++;
  for (n = 0; isdigit(s[i]); i++)
    n = 10 * n + (s[i] - '0');
  n*=sign;
  printf("s = %s\nn = %d\n", s, n);
  return 0;
}
```

Результат работы:

```
s =   -347ab
n = -347
```

В некоторых случаях тело цикла должно быть выполнено хотя бы один раз, вне зависимости от истинности выражения. Это может быть реализовано, если проверка значения выражения будет располагаться после тела цикла (рис. 3.8).

Рис. 3.8. Алгоритм работы оператора do-while

Это записывается так:

```
do оператор while(выражение);
```

Такая конструкция называется *циклом с постусловием*. Цикл с постусловием применяется на практике значительно реже, чем цикл с предусловием или цикл `for`.

3.6 Оператор break

Иногда возникает необходимость прервать выполнение тела цикла и «досрочно» выйти из цикла. Это можно сделать с помощью оператора `break`, который вызывает переход в точку программы, расположенную непосредственно за циклом. Оператор `break` может использоваться в любом операторе цикла, а также в операторе множественного выбора `switch`.

Пример. Следующая программа удаляет «незначащие» символы в конце строки `s`.

```
int main() {
  int n;
  char s[] = "qwerty  \t\t \n\n";
  printf("before:\n");
  printf("number of char's(%s) = %d\n",s,strlen(s));
  for (n = strlen(s)-1; n >= 0; n--)
    if (s[n]!=' ' && s[n]!='\t' && s[n]!='\n') break;
  s[n+1] = '\0';
  printf("after:\nnumber of char's(%s) = %d\n", s,
   strlen(s));
  return 0;
}
```

Результат работы:

```
before:
number of char's(qwerty

) = 13
after:
number of char's(qwerty) = 6
Program ended with exit code: 0
```

3.7 Оператор continue

Этот оператор похож на `break`, но, в отличие от него, досрочно прекращает выполнение текущей итерации цикла, а не всего оператора цикла. Для циклов `while` и `do` это означает переход к проверке условия, а для цикла `for` – вычисление `выражения3`, а уже затем переход к проверке условия.

Пример. Подсчитать количество положительных элементов массива и найти их среднее арифметическое.

```
int main() {
  int x[] = {-1, 4, 0, -3, -7, 5, 11, -2}, i, n;
  double s;
  for(i=0, n=0, s=0.0; i<sizeof(x)/sizeof(x[0]); i++)
{
    if(x[i] <= 0) continue;
    s += x[i]; n++;
  }
  if(n) s /= n;
  printf("number of positive elements = %d\n", n);
  printf("mean value = %f\n", s);
  return 0;
}
```

Результат работы:

```
number of positive elements = 3
mean value = 6.666667
```

3.8 Оператор goto и метки

Оператор `goto метка` вызывает безусловный переход к оператору (в той же самой функции), перед которым записана `метка` (обычное имя, заканчивающееся символом `:` - двоеточие).

Пример. Проверить, имеют ли два массива хотя бы один общий элемент.

```
int main() {
  int a[] = {-1, 4, 0, -3, -7, 5, 11, -2}, i;
  int b[] = {7, -12, 8, 9, 11, -2}, j;
  int n = sizeof(a) / sizeof(a[0]);
  int m = sizeof(b) / sizeof(b[0]);
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      if (a[i] == b[j]) goto found;
  printf("no common elements\n");
  goto final;
found:
  printf("common elements: a[%d] == b[%d]\n", i, j);
final:
  return 0;
}
```

Результат работы: `common elements: a[6] == b[4]`

Оператор `goto` является «нежелательным» оператором, т.к. «запутывает» логическую структуру программы, однако бывают ситуации, когда программа без `goto` получается более громоздкой, чем с `goto`.

Пример. Предыдущая программа без использования `goto`.

```
int main() {
  int a[] = {-1, 4, 0, -3, -7, 5, 11, -2}, i;
  int b[] = {7, -12, 8, 9, 11, -2}, j, found = 0;
  int n = sizeof(a) / sizeof(a[0]);
  int m = sizeof(b) / sizeof(b[0]);
  for (i = 0; i < n && !found; i++)
    for (j = 0; j < m && !found; j++)
      if (a[i] == b[j]) found = 1;
  if (found)
    printf("common elements: a[%d]==b[%d]\n",--i,--
j);
  else
    printf("no common elements\n");
  return 0;
}
```

Результат работы: `common elements: a[6] == b[4]`

4. Функции и структура программы

4.1 Функции и понятие модульности программы

Модульность в языках программирования – это принцип, согласно которому программное средство (ПС) – программа, библиотека, веб-приложение и др. разделяется на отдельные сущности, называемые модулями. Модульность позволяет упростить задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков, а также позволяет реализовать методологию *повторного использования кода*.

При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями. Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и другие программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

В языке С модульность поддерживается:

- функциями,
- препроцессоными командами,
- многофайловой структурой программы и
- заголовочными файлами.

Текст программа на языке С может весь размещаться в одном файле и такая программа будет иметь *однофайловую* структуру, либо текст программы может быть размещен в нескольких файлах и тогда программа будет *многофайловой*. Любая однофайловая программа является набором определений: типов, переменных и функций.

С помощью *функций* большие задачи разбивают на более мелкие. Это позволяет инкапсулировать («упрятать» в оболочку) детали *реализации* некоторой функциональности и предоставить пользователям («клиентам») формат обращения к этой

функциональности (*интерфейс*). В результате программа в целом становится более ясной и в нее проще вносить изменения.

Пример. Функция-парсер `atoi_()`, преобразующая символьное изображение числа, записанное в строке, в само число, которая затем вызывается в основной программе. В этом примере программа распределена по двум файлам: `main.c`, в котором находится основная программа и описывается интерфейс функции `atoi_()` и `atoi_.c`, который содержит реализацию этой функции.

```
// файл main.c
#include <stdio.h>
int atoi_(char[]); // описание интерфейса функции
int main() {
  int n = atoi_("  -347ab");
  printf("n = %d\n", n);
  return 0;
}
// файл atoi_.c - описание реализации функции
#include <ctype.h>
int atoi_(char s[]) {
  int i, n, sign;
  for (i = 0; isspace(s[i]); i++) ;
  sign = (s[i] == '-') ? -1 : 1;
  if (s[i] == '+' || s[i] == '-') i++;
  for (n = 0; isdigit(s[i]); i++)
    n = 10 * n + (s[i] - '0');
  return sign * n;
}
```

4.2 Определение функции и структура программы

Для того, чтобы использовать функцию, ее необходимо определить, т.е. описать её интерфейс (объяснить, как функцией можно воспользоваться) и привести программный код (тело функции), раскрывающий, как функция работает (записать реализацию функции на языке программирования).

Определение функции имеет следующую форму:

```
тип_возвращ_знач имя_функции(список_объявлений_арг) {
  объявления и операторы
}
```

Различные части этого определения могут отсутствовать, но обязательными являются: `имя_функции`, пара круглых скобок и пара фигурных скобок, т.е. «минимальная» функция определяется так: `fun(){}`. Это – «пустышка», которая не принимает никаких аргументов и ничего не делает. Подобные функции могут использоваться в качестве «заглушек» при разработке программ. Если при объявлении функции не указан тип возвращаемого значения, то по умолчанию подразумевается тип `int`. Отсутствие списка объявлений аргументов означает, что функция не использует аргументы.

Функции обмениваются данными посредством передачи аргументов и возвращения значений, а также через внешние переменные. Функции могут следовать друг за другом в файле исходного кода в любом порядке, и текст программы можно разбивать на любое количество файлов, но при этом запрещается разбивать текст функции между файлами. Функция не может быть также определена внутри другой функции.

В результате своей работы функция может возвратить в вызывающую ее функцию результат – некоторое значение, тип которого объявлен перед именем функции. Для этого в теле функции должен присутствовать хотя бы один оператор возврата вида: `return выражение;` Вызывающая функция может игнорировать (т.е. не использовать) возвращаемое значение.

Существует еще одна форма оператора возврата: `return;` В этом случае в вызывающую функцию ничего не передается, а при определении функции в качестве типа возвращаемого значения указывается `void`. В теле функции может отсутствовать оператор возврата `return`. В этом случае возврат из функции происходит при достижении конца тела функции (закрывающей скобки `}`).

Аргументы функции передаются в нее *по значению*, т.е. при вызове функции значения аргументов присваиваются временным (внутренним) переменным и поэтому все действия функции со своими аргументами никак не отражаются на переменных, переданных функции при вызове. Если же требуется, чтобы функция оперировала с переменными, которые ей были переданы

в качестве аргументов, то нужно использовать указатели (адреса переменных-аргументов).

Пример. Функция обмена значениями между двумя переменными:

```
void swap(int a, int b) {
  int c = a; a = b; b = c;
}
int main() {
  int x = 5, y = 7;
  printf("x = %d, y = %d\n",x,y);
  swap(x, y);
  printf("x = %d, y = %d\n",x,y);
  return 0;
}
```

Обмена не присходит, т.к. действия производились с локальными переменными:

```
x = 5, y = 7
x = 5, y = 7
```

Если же использовать указатели и вместо самих аргументов передавать их адреса:

```
void swap(int *a, int *b) {
  int c = *a; *a = *b; *b = c;
}
int main() {
  int x = 5, y = 7;
  printf("x = %d, y = %d\n",x,y);
  swap(&x, &y);
  printf("x = %d, y = %d\n",x,y);
  return 0;
}
```

то обмен будет произведен:

```
x = 5, y = 7
x = 7, y = 5
```

4.3 Функция main

Программа может использовать любое количество функций при одном условии: в программе обязательно должна присутствовать в точности одна функция с именем `main`

(«главная» функция), так как запуск программы на выполнение операционной системой производится всегда через эту функцию (т.е. `main` начинает выполняться первой).

Функция `main` всегда имеет тип возвращаемого значения `int`, через это значение ОС уведомляется об успешности или не успешности завершения программы (функция main передает в ОС «код завершения»). А т.к. `int` подразумевается для любой функции «по умолчанию», то тип возвращаемого значения для `main` часто не указывают.

Существуют две формы `main`:

- без аргументов – `main()` и
- с аргументами – `main(int, char**)`.

Вторая форма `main` позволяет при вызове программы из командной строки передать в нее произвольное количество строковых аргументов (этот вариант будет описан ниже).

4.4 Пример программы с модульной структурой

Разработаем программу, позволяющую находить с заданной точностью корень нелинейного уравнения $f(x)=0$ методом деления отрезка пополам. Пусть заданы: функция $f(x)$, отрезок $[a, b]$, на котором предположительно находится корень x_0; точность определения нуля функции – *epsy* и точность определения корня – *epsx*. Для наглядности ниже приведен эскиз, демонстрирующий ход решения этой задачи (рис. 4.1).

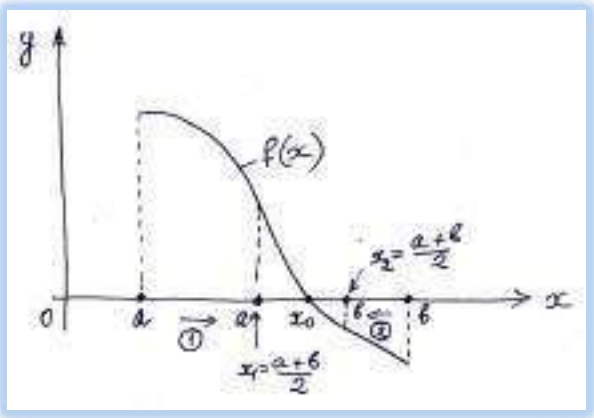


Рис. 4.1. Демонстрация работы метода деления отрезка пополам

Суть метода состоит в вычислении значения функции посередине текущего отрезка, сравнении полученного значения с

epsy и переносе левой или правой границы в середину, если не достигнута требуемая точность.

Далее приведен текст однофайловой программы, которая отыскивает корень функции cos(*x*) на отрезке [0, 3]:

```
double fun(double x) {
  return cos(x);
}
double bisec(double a, double b, double epsx,
             double epsy) {
  double y1, y2, x;
  if(fabs(y1=fun(a)) < epsy) return a;
  if(fabs(y2=fun(b)) < epsy) return b;
  if(y1*y2 > 0.0) {
    printf("bisec: no root...\n");
    return 1.e10;
  }
  while(1) {
    x = (a+b)/2.0;
    if(fabs(y1=fun(x)) < epsy) return x;
    if(y1*y2 > 0.0) b = x; else a = x;
    if(fabs(a-b) < epsx) return x;
  }
}
int main() {
  printf("result=%f\n",bisec(0., 3., 1e-10, 1e-10));
  return 0;
}
```

Результат работы: `result=1.570796`

При разработке больших по объему кода программ правильнее применять модульный подход и разбивать программу на отдельные модули (файлы). При этом рекомендуется создавать по два модуля на каждую функцию (кроме `main`): модуль реализации (файл `*.c`) и модуль интерфейса (файл `*.h`). С использованием этих рекомендаций представим программу поиска корня, как состоящую из 5 модулей (рис. 4.2).

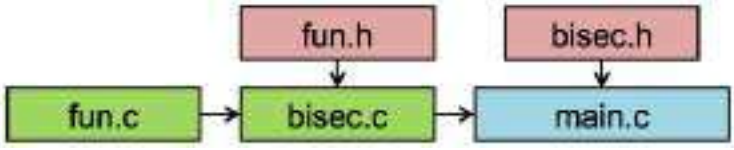


Рис. 4.2. Модульная структура программы поиска корня

Далее приведен текст многофайловой программы поиска корня:

```
// файл main.c
#include <stdio.h>
#include "bisec.h"
int main() {
  printf("result=%f\n",bisec(0., 3., 1e-10, 1e-10));
  return 0;
}
// файл bisec.h
double bisec(double, double, double, double);
// файл bisec.c
#include <stdio.h>
#include <math.h>
#include "fun.h"
double bisec(double a, double b, double epsx,
             double epsy) {
  double y1, y2, x;
  if(fabs(y1=fun(a)) < epsy) return a;
  if(fabs(y2=fun(b)) < epsy) return b;
  if(y1*y2 > 0.0) {
    printf("bisec: no root...\n");
    return 1.e10;
  }
  while(1) {
    x = (a+b)/2.0;
    if(fabs(y1=fun(x)) < epsy) return x;
    if(y1*y2 > 0.0) b = x; else a = x;
    if(fabs(a-b) < epsx) return x;
  }
}
// файл fun.h
double fun(double);
// файл fun.c
#include <math.h>
double fun(double x) { return cos(x); }
```

4.5 Рекурсия

Рекурсия – это процесс повторения чего-либо самоподобным способом. Например, вложенные отражения, производимые двумя точно параллельными друг другу зеркалами, являются одной из форм бесконечной рекурсии.

Наиболее общее применение рекурсия находит в математике и информатике. Здесь она является методом определения функций, при котором определяемая функция применена (вызвана) в теле своего же собственного определения.

При этом бесконечный набор случаев (значений функции) описывается с помощью конечного выражения, которое для некоторых случаев может ссылаться на другие случаи, если при этом не возникает циклов или бесконечной цепи ссылок. Фактически это способ определения множества объектов через самого себя с использованием ранее заданных частных определений.

Примеры рекурсии в математике:

- факториал целого неотрицательного числа n определяется так: $n! = n * (n\text{-}1)!$, $0! = 1$;
- числа Фибоначчи: $F_n = F_{n\text{-}1} + F_{n\text{-}2}$, $F_1 = F_2 = 1$;
- практически все геометрические фракталы задаются в форме бесконечной рекурсии, например, треугольник Серпинского (рис. 4.3).

Рис. 4.3. Последовательность треугольников Серпинского

В программировании рекурсия – это вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B – функцию A. Количество вложенных вызовов функции или процедуры называется *глубиной рекурсии*.

Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое число объектов. С помощью рекурсивной программы возможно описать бесконечное вычисление, причём без явных повторений частей программы.

Пример. Программа – генератор чисел Фибоначчи:

```
int fib(int i) {
```

```
  if (i < 3) return 1;
  return fib(i-1) + fib(i-2);
}
int main() {
  for(int i = 1; i < 11; i++)
    printf("F(%d) = %d\n", i, fib(i));
  return 0;
}
```

Результат работы:

```
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55
```

Пример. Программа, демонстрирующая взаимную рекурсию. Эта программа определяет, четно ли количество единиц в двоичном представлении заданного числа `n` (1 – четно, 0 – нечетно). Рекурсивная функция `g0()` обрабатывает состояние «четное количество» и завершается с результатом 1, а рекурсивная функция `g1()` обрабатывает состояние «нечетное количество» и завершается с результатом 0:

```
int g0 (unsigned n) {
  if( n == 0) return 1;
  if( (n & 01) == 0) return g0(n >> 1);
  return g1(n >> 1);
}
int g1 (unsigned n) {
    if( n == 0) return 0;
    if( (n & 01) == 0) return g1(n >> 1);
    return g0(n >> 1);
}
int main() {
  unsigned n;
  while(1) {
```

```
    printf("enter the number (>0): "); scanf("%d",
&n);
    if( n == 0) break;
    printf("result=%d\n", g0(n));
  }
  return 0;
}
```

Результат работы:

```
enter the number (>0): 10
result=1
enter the number (>0): 7
result=0
enter the number (>0): 5
result=1
enter the number (>0): 4
result=0
enter the number (>0): 0
```

5. Указатели и массивы

5.1 Адреса объектов программы

Оперативную память компьютера можно рассматривать как линейную последовательность пронумерованных «ячеек» (байтов). Номер ячейки памяти называется ее *адресом*. Если считать архитектуру компьютера 32-разрядной, то адреса будут иметь вид 32-битных кодов. Наименьшим адресом будет `0x00000000`, а наибольшим – `0xFFFFFFFF`. При этом адресуемый объем памяти составит 2^{32} байт, или, в более крупных единицах – 4 гигабайт (Гбайт).

Каждый объект программы занимает определенный участок оперативной памяти компьютера и, следовательно, может быть охарактеризован адресом начала этого участка, который и считается *адресом* данного *объекта*. В языке С имеется специальная унарная операция &, которая выдает адрес объекта (своего операнда) в оперативной памяти.

Пример. Программа, позволяющая «увидеть» адреса объектов программы с использованием специального формата вывода %p:

```
int x;
short y;
char a[256], b[512];
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
int main() {
  printf("addr(x) = %p\n", &x);
  printf("addr(y) = %p\n", &y);
  printf("addr(a) = %p\n", &a);
  printf("addr(b) = %p\n", &b);
  printf("addr(fact) = %p\n", &fact);
  return 0;
}
```

Результат работы:

```
addr(x) = 0x100001120
addr(y) = 0x100001124
addr(a) = 0x100001020
addr(f) = 0x100000e60
Program ended with exit code: 0
```

Заметим, что некоторые объекты программы не требуют применения к ним операции взятия адреса &, т.к. они сами и являются адресами. К их числу относятся функции и массивы. Для иллюстрации этого немного изменим приведенную выше программу, убрав операцию & для объектов `a`, `b` и `fact`:

```
int x;
short y;
char a[256], b[512];
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
int main() {
  printf("addr(x) = %p\n", &x);
  printf("addr(y) = %p\n", &y);
  printf("addr(a) = %p\n", a);
  printf("addr(b) = %p\n", b);
  printf("addr(fact) = %p\n", fact);
  return 0;
}
```

Результат работы будет тем же:

```
addr(x) = 0x100001120
addr(y) = 0x100001124
addr(a) = 0x100001020
addr(f) = 0x100000e60
Program ended with exit code: 0
```

5.2 Адресная арифметика

Можно предположить, что адреса являются обычными числами и могут участвовать в арифметических операциях. Для проверки этого составим программу:

```
int x;
short y;
```

```
char a[256], b[512];
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
int main() {
  printf("addr(x) = %p\n", &x);
  printf("addr(x)+2 = %p\n", &x+2);
  printf("addr(y) = %p\n", &y);
  printf("addr(y)-128 = %p\n", &y-128);
  printf("addr(a) = %p\n", a);
  printf("addr(a)+2 = %p\n", a+2);
  printf("addr(b) = %p\n", b);
  printf("addr(b)+528 = %p\n", b+528);
  printf("addr(fact) = %p\n", fact);
  printf("addr(fact)+2 = %p\n", fact+2);
  return 0;
}
```

Результат работы:

```
addr(x) = 0x100001320
addr(x)+2 = 0x100001328
addr(y) = 0x100001324
addr(y)-128 = 0x100001224
addr(a) = 0x100001020
addr(a)+2 = 0x100001022
addr(b) = 0x100001120
addr(b)+528 = 0x100001330
addr(fact) = 0x100000d80
addr(fact)+2 = 0x100000d82
Program ended with exit code: 0
```

Проанализируем полученные результаты:

- при добавлении константы 2 к адресу переменной `x` типа `int` получилось значение, на 8 больше адреса `x`. Это объясняется тем, что объект типа `int` имеет размер 4 байта;
- при вычитании константы 128 из адреса переменной `y` типа `short` мы получили значение, на 256 меньше адреса `y` (действительно, 0x100001324-0x100001224=0x100=256). Это объясняется тем, что объект типа `int` имеет размер 2 байта;

- при добавлении константы 2 к адресу массива `a` типа `char` мы получили значение, на 2 больше адреса `a`. Это объясняется тем, что объект типа `char` имеет размер 1 байт;
- при добавлении константы 528 к адресу массива `b` типа `char` мы получили значение, на 528 больше адреса `b` (действительно, 0x100001330-0x100001120=0x210=528). Это объясняется тем, что объект типа `char` имеет размер 1 байт;
- при добавлении константы 2 к адресу функции `fact` мы получили значение, на 2 больше адреса `fact`. Это объясняется тем, что условно принято считать, что объект типа функция имеет размер 1 байт.

На основании сделанных выводов можно сформулировать следующие *правила адресной арифметики*:

1. К адресу объекта можно прибавлять или из адреса объекта можно вычитать целые значения. При этом результат операции зависит от типа объекта, адрес которого участвует в операции. Если для данного типа размер объекта равен `n` байт и к адресу прибавляется `i`, то значение адреса изменится на `i*n`.
2. Никакие другие операции к адресам не применимы, т.е. адреса нельзя умножать, делить, складывать между собой и пр. Исключением является операция вычитания адресов однотипных объектов, результатом которой является целое число, равное количеству объектов данного типа, которые могут разместиться на участке памяти между адресами-операндами.

Пример. Программа, демонстрирующая операцию вычитания адресов:

```
int x;
short y;
char a[256], b[512];
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
```

```
int main() {
  printf("addr(b)-addr(a) = %d\n", b-a);
  printf("addr(a)-addr(b) = %d\n", a-b);
  printf("addr(y)-addr(a) = %d\n", (char*)&y-a);
  printf("addr(y)-addr(a) = %d\n", &y-(short*)a);
  printf("addr(x)-addr(a) = %d\n", (void*)&x-
(void*)a);
  printf("addr(x)-addr(a) = %d\n",
    (double*)&x-(double*)a);
  return 0;
}
```

Результат работы:

```
addr(b)-addr(a) = 256
addr(a)-addr(b) = -256
addr(y)-addr(a) = 772
addr(y)-addr(a) = 386
addr(x)-addr(a) = 768
addr(x)-addr(a) = 96
Program ended with exit code: 0
```

Проанализируем полученные результаты:

- при вычитании адресов двух байтовых массивов `b` и `a` получили 256, что равно размеру массива `a` и означает, что массив `b` расположен в памяти сразу за массивом `a`;
- при вычитании адресов этих же массивов в обратном порядке получен ожидаемый результат -256;
- при вычитании адресов объектов `y` и `a`, приведенных к одному типу `char*` (указатель на `char`), получено число 772, а при таком же вычитании, но с приведением адресов к типу `short*`, получено вдвое меньшее число потому, что размер объекта типа `short` равен 2 байта (т.е. в блоке памяти размером 772 байта размещается 386 объектов типа `short`);
- при вычитании адресов объектов `x` и `a`, приведенных к одному типу `void*` (указатель на `void`), получено число 768, а при таком же вычитании, но с приведением адресов к типу `double*`, получено в 8 раз меньшее число 96 потому, что размер объекта типа `double` равен 8 байт

(при этом размер объекта типа `void` условно считается равен 1 байту).

5.3 Указатели

Адреса объектов программы можно сохранять в переменных специального вида – *указателях*. Указатели образуют не один конкретный тип данных (хотя все указатели имеют один размер – 4 байта); типов указателей существует столько, сколько существует типов данных, т.е. для любого типа данных `T` существует тип указателя `T*`.

Примеры типов указателей: `char*`, `int*`, `short*`, `long*`, `float*`, `double*`, `void*`.

Указатели типа `void*` называются не типизированными (универсальными) указателями. При выполнении арифметических операций над такими указателями считается, что объекты, на которые они указывают, имеют размер 1 байт.

Пример. Демонстрация использования указателей.

```
int x;
short y;
char a[256];
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
int main() {
  int* px = &x;
  short* py = &y;
  char* pa = a;
  void* pf = fact;
  printf("px = %p\n", px);
  printf("py = %p\n", py);
  printf("pa = %p\n", pa);
  printf("pf = %p\n", pf);
  return 0;
}
```

Результат работы:

```
px = 0x100001120
py = 0x100001124
pa = 0x100001020
pf = 0x100000e80
Program ended with exit code: 0
```

Через указатель можно получить доступ к участку памяти (объекту), адрес которого является текущим значением указателя. Для этого в языке С существует унарная операция *, применяемая к указателями. Эта операция называется операцией *ссылки по указателю*, или операцией *разыменования*.

Операции & и * являются взаимно-дополняющими и, если они расположены подряд, взаимно компенсируют друг друга. Пусть, например, дано определение:

```
int x;
```

Тогда `&x` – адрес `x`; `*&x` – сама переменная `x`; `&*&x` – адрес `x` и т.д.

Переменная типа указатель может в разное время принимать различные значения. Следовательно, через одну и туже переменную-указатель можно обращаться к разным переменным (в разное время).

Пример. Демонстрация использования указателей.

```
int main() {
  int a=2, b=3, c=4;
  printf("a=%d, b=%d, c=%d\n", a, b, c);
  int* p=&a;
  *p=7;
  p=&b; *p=8;
  p=&c; *p=9;
  printf("a=%d, b=%d, c=%d\n", a, b, c);
}
```

Результат работы:

```
a=2, b=3, c=4
a=7, b=8, c=9
Program ended with exit code: 0
```

5.4 Указатели и аргументы функций

В теории программирования известны два способа передачи аргументов в функцию: *по значению* и *по имени* (ссылке). В обоих случаях аргументы функции являются её локальными переменными и «не видны» извне. Различие способов состоит в том, что при вызове функции её аргумент-локальная переменная получает:

- в первом случае - копию значения фактического аргумента;
- во втором случае - ссылку на фактический аргумент (т.е. его адрес) и, следовательно, может им распоряжаться.

Каждый из этих способов имеет свои преимущества и недостатки и программист должен обоснованно подходить к выбору наиболее подходящего для конкретной ситуации способа.

Можно указать, по крайней мере, два случая, когда необходимо использовать передачу аргумента по имени:

- функция должна изменить значение передаваемого ей аргумента;
- размер аргумента велик и передача его по значению приведет к нерациональным затратам памяти на создание копии аргумента.

В языке С «штатно» реализован первый способ (т.е. передача аргументов по значению), однако и второй способ легко реализуется, если в качестве аргумента передавать не сам объект, а его адрес (указатель).

Классическим примером можно считать передачу массива в функцию: в силу того, что имя массива является его адресом, функция не создает копию обрабатываемого массива, а работает с ним по месту его размещения в вызывающей программе.

Пример. Функция, подсчитывающая сумму элементов массива:

```
int sum(int m[], int n) {
  int i, s;
  for(i=0, s=0; i<n; s+=m[i++]);
  return s;
}
```

```
#define N 100
int main() {
  int z[N], i;
  for(i=0; i<N; i++)
    z[i] = 2*i+1;
  printf("summa = %d\n", sum(z, N));
  return 0;
}
```

Результат работы:

```
summa = 10000
Program ended with exit code: 0
```

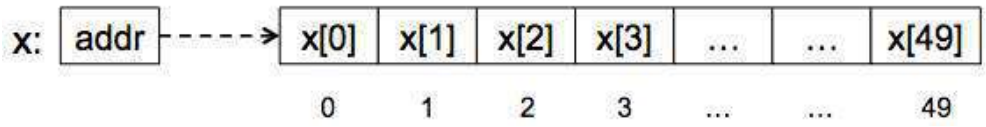
5.5 Массивы

Массив - это одна из наиболее простых структур данных, представляющих собой линейную последовательность фиксированного количества однотипных элементов, расположенных в памяти друг за другом непрерывно и допускающих прямой доступ к любому элементу по его номеру (индексу). Нумерация элементов массива начинается с 0, поэтому номер (индекс) последнего элемента *n*-элементного массива будет равен *n*-1.

Массив характеризуется именем, типом элементов и их количеством. Пример определения массива:

```
int x[50];
```

Следующий рисунок иллюстрирует расположение элементов массива в памяти, их нумерацию, а также тот факт, что имя массива является его адресом:



В языке С существует очень тесная связь между массивами и указателями, которая характеризуется простым утверждением: «имя массива – это константный указатель на его начало». Из этого утверждения следуют важные соотношения (здесь `T` – имя типа):

`x[i]` эквивалентно `*(x+i)`

`T x[];` эквивалентно `T *x;`

При определении массива в программе необходимо указать размер массива (количество элементов), причем эта величина должна быть известна в момент выделения памяти массиву компилятором. С учетом этого требования возможны следующие способы указания размера массива:

- размер массива задается константным выражением;
- размер массива задается выражением, в которое входят константы и внешние переменные по отношению к блоку, в котором производится определение массива, и эти переменные получили значения;
- размер массива не задается, а определяется списком инициализации;
- размер массива задается выражением, в которое могут входить переменные, которые определены в этом же блоке выше по тексту программы и получили значения к моменту определения массива. Это – так называемые массивы переменной длины (variable-length array , VLA), которые появились в стандарте C11.

Пример. Функция, подсчитывающая сумму элементов массива:

```
#define N 100
int n1 = 10;
int main() {
  int a[N], b[30], c[n1];
  int n = 20, m = 3;
  int d[n-4*m];
  int e[] = {1, 2, 3, 4};
  printf("Размер d = %d\n", sizeof(d)/sizeof(d[0]));
  printf("Размер e = %d\n", sizeof(e)/sizeof(e[0]));
}
```

Результат работы:

```
Размер d = 8
Размер e = 4
Program ended with exit code: 0
```

5.6 Инициализация массивов

Сразу после определения массива его элементы имеют неопределенные (произвольные) значения. Начальные значения элементам массива можно задать непосредственно при его определении с помощью *списка инициализации*. Количество элементов в списке не может быть больше количества элементов массива. Если в списке меньше элементов, чем размер массива, остальные элементы получают нулевые значения.

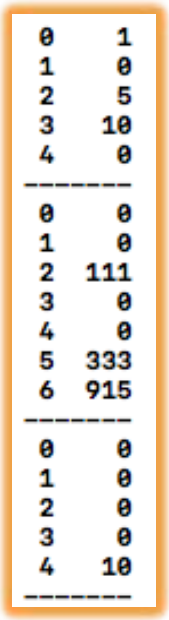
При использовании списка инициализации размер массива можно не указывать; он будет установлен по количеству элементов в списке инициализации (см. предыдущий пример).

В последней версии языка С, определенной в стандарте С11, появился новый вид инициализаторов массивов – *назначенные инициализаторы* (designated initializers), которые позволяют задать значение конкретному элементу массива с помощью индекса.

Пример. Использование назначенных инициализаторов:

```
int main() {
  int a[5] = {1, [2]=5, 10};
  int b[] = {[2]=111, [5]=333, 915};
  int c[] = {[4]=10};
  for (int i=0; i<sizeof(a)/sizeof(a[0]); i++)
    printf("%2d%5d\n", i, a[i]);
  printf("-------\n");
  for (int i=0; i<sizeof(b)/sizeof(b[0]); i++)
    printf("%2d%5d\n", i, b[i]);
  printf("-------\n");
  for (int i=0; i<sizeof(c)/sizeof(c[0]); i++)
    printf("%2d%5d\n", i, c[i]);
  printf("-------\n");
}
```

Результат работы:



Еще одно новшество стандарта C11 – это составные литералы (безымянные массивы). Они позволяют, например, передавать в функцию в качестве фактического аргумента сконструированный «на лету» массив:

```
int sum(int m[], int n) {
  int i, s;
  for(i=0, s=0; i<n; s+=m[i++]);
  return s;
}
int main() {
  printf("summa = %d\n", sum((int[]){2, [3]=4, 6},
5));
  return 0;
}
```

В этом примере «на лету» был сконструирован и передан в функцию `sum` массив типа `int` из 5 элементов {2, 0, 0, 4, 6} с помощью списка инициализации, в котором использован назначенный инициализатор.

5.7 Динамические массивы

В тех случаях, когда размер массива невозможно знать заранее (например, эта величина вводится человеком), можно

использовать динамические массивы, память для которых выделяется по ходу вычислительного процесса библиотечной функцией `calloc`:

```
void* calloc(size_t n, size_t r)
```

где `n` – количество элементов массива, `r` – размер элемента в байтах.

Функция `calloc` возвращает нетипизированный указатель на начало массива.

Пример. Динамическое выделение памяти массиву.

```
int sum(int m[], int n) {
    int i, s;
    for(i=0, s=0; i<n; s+=m[i++]);
    return s;
}
int main() {
  int i, n, *z;
  printf("Введите n : "); scanf("%d", &n);
   z = (int*)calloc(n, sizeof(*z));
  for(i=0; i<n; i++) z[i] = 2*i+1;
  printf("summa = %d\n", sum(z, n));
  return 0;
}
```

Результат работы:

```
Введите n : 10
summa = 100
Program ended with exit code: 0
```

В программировании встречается неприятная ситуация, которая называется «утечка памяти». Эта ситуация может возникнуть при использовании динамического выделения памяти функцией `calloc`. Для устранения этой ситуации рекомендуется освобождать неиспользуемую память с помощью функции

```
void free(void *p)
```

где `p` – нетипизированный указатель, получивший значение в результате вызова `calloc`.

Пример. Динамическое выделение памяти с последующим ее освобождением.

```
int main() {
  int i, n, *z;
  void* p;
  printf("Введите n : "); scanf("%d", &n);
  z = (int*)(p = calloc(n, sizeof(*z)));
  for(i=0; i<n; i++) z[i] = 2*i+1;
  printf("summa = %d\n", sum(z, n));
  free (p);
  return 0;
}
```

5.8 Символьные указатели, символьные массивы и строки символов

Язык С не имеет отдельного типа данных для работы со строками символов и они представляются массивами символов `char[]`. Например, строковая константа `"I am a string"` будет представлена в памяти символьным массивом из 14 элементов (13 символов+`'\0'`).

Рассмотрим два способа определения объектов программы, с помощью которых можно работать со строками:

```
char amessage[] = "I am a string";
char* pmessage = "I am a string";
```

В первом случае `amessage` – это массив, занимающий 14 байтов памяти, заполненных символами из строковой константы и каждый символ в этом массиве можно изменять. При этом переменная `amessage` всегда будет указывать на один и тот же участок памяти и её значение изменить нельзя.

Во втором случае `pmessage` – указатель размером 4 байта. С его помощью можно также получить доступ к любому символу строковой константы, но изменять символы нельзя. Кроме того, переменная `pmessage` в дальнейшем может получить другое значение, например, в результате присваивания:

```
pmessage = "I am another string";
```

При выполнении различных преобразований над строками необходимо очень внимательно подходить к учёту разницы между представлением строк массивами и указателями.

Рассмотрим известную задачу «обмен значениями» для двух строк.

Пример. В следующей программе строки представлены массивами, копирование строки-источника в строку-приемник выполняется посимвольно через указатели с инкрементированием указателей, условием останова является достижение конца строки-источника (копирование символа `'\0'`):

```
#define MAXLEN 200
void strcpy_(char* s, char* t) {
  while(*s++ = *t++);
}
void swaps(char* a, char *b) {
  char c[MAXLEN];
  strcpy_(c, a); strcpy_(a, b); strcpy_(b, c);
}
int main() {
  char s1[MAXLEN] = "first";
  char s2[MAXLEN] = "second";
  printf("%s, %s\n", s1, s2);
  swaps(s1, s2);
  printf("%s, %s\n", s1, s2);
  return 0;
}
```

Результат работы:

```
first, second
second, first
Program ended with exit code: 0
```

Пример. В следующей программе строки представлены указателями:

```
#define MAXLEN 200
void strcpy_(char* s, char* t) {
```

```
  while(*s++ = *t++);
}
void swaps(char* a, char *b) {
  char c[MAXLEN];
  strcpy_(c, a); strcpy_(a, b); strcpy_(b, c);
}
int main() {
  char* s1 = "first";
  char* s2 = "second";
  printf("%s, %s\n", s1, s2);
  swaps(s1, s2);
  printf("%s, %s\n", s1, s2);
  return 0;
}
```

В результате запуска этой программы будет получена ошибка времени исполнения:

Основная причина этой ошибки – попытка изменить символы в строковой константе, которая имеет свойство «только для чтения». Однако в этом случае есть более лёгкое решение – строки оставить на месте, а поменять значения указателей:

```
void swaps(char** a, char** b) {
  char *c = *a;
  *a = *b; *b = c;
}
int main() {
  char* s1 = "first";
  char* s2 = "second";
  printf("%s, %s\n", s1, s2);
  swaps(&s1, &s2);
  printf("%s, %s\n", s1, s2);
  return 0;
}
```

Результат работы:

```
first, second
second, first
Program ended with exit code: 0
```

В этой программе функция работает не с указателями на строки, а с адресами этих указателей, что позволило изменять значения указателей на строки и получить эффект обмена значениями между строками (хотя «физически» строки остались на своих местах).

5.9 Массивы указателей

Так как указатели являются обычными переменными, их можно хранить в массиве. Создадим объект `arrs` типа «массив строк», проинициализируем его и выполним перестановку некоторых элементов этого массива:

```
void swaps(char** a, char** b) {
  char *c = *a;
  *a = *b; *b = c;
}
int main() {
  char* arrs[] = {"first", "second", "third",
      "fourth", "fifth"};
  int i, n = sizeof(arrs)/sizeof(*arrs);
  for(i=0; i<n; i++) printf("%d: %s\n", i+1,
arrs[i]);
  swaps(&arrs[0], &arrs[2]);
  swaps(&arrs[1], &arrs[4]);
  printf("--------\n");
  for(i=0; i<n; i++) printf("%d: %s\n", i+1,
arrs[i]);
  return 0;
}
```

Результат работы:

```
1: first
2: second
3: third
4: fourth
5: fifth
--------
1: third
2: fifth
3: first
4: fourth
5: second
```

5.10 Многомерные массивы

Синтаксис языка С допускает, чтобы при определении массива его элементами были массивы (одинакового размера!). В этом случае приходим к понятию многомерного прямоугольного массива. Например, известный математический объект «матрица» может быть представлен двумерным массивом. Размер массива по первому измерению можно не задавать; он будет определен из списка инициализации:

```
int main() {
  int matr[][3] = {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9},
                   {10, 11, 12}};
  int i, j, n = sizeof(matr)/sizeof(*matr), m = 3;
  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++)
      printf("%4d", matr[i][j]);
    printf("\n");
  }
  return 0;
}
```

Результат работы:

```
   1   2   3
   4   5   6
   7   8   9
  10  11  12
Program ended with exit code: 0
```

Выясним роль указателей при работе с многомерными массивами. По аналогии с одномерными массивами можно предположить, что справедливо соотношение:

`x[i][j]` эквивалентно `*(*(x+i)+j)`.

Проверим это:

```
int main() {
  int matr[][3] = {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9},
                   {10, 11, 12}};
  int i, n = sizeof(matr)/sizeof(*matr);
  int j, m = 3;
  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++)
      printf("%4d", *(*(matr+i)+j));
    printf("\n");
  }
  return 0;
}
```

Результат работы тот же:

```
   1   2   3
   4   5   6
   7   8   9
  10  11  12
Program ended with exit code: 0
```

5.11 Многомерные массивы нерегулярной структуры

С помощью указателей можно сконструировать необычные, «зубчатые» массивы, в которых каждая строка может иметь свой размер, например так:

Пример. В следующей программе конструируется зубчатый массив и выводятся на печать неколько его элементов с

использованием указателей и с использованием операции доступа к элементу массива по индексам (как к обычному двумерному массиву):

```
int main() {
  int m1[] = {1, 2};
  int m2[] = {5, 3, 4, 7, 9, 2};
  int m3[] = {3, 8, 5};
  int *matr[] = {m1, m2, m3};
  printf("[1,1]->%4d\n",*(*(matr+1)+1));
  printf("[1,5]->%4d\n",*(*(matr+1)+5));
  printf("[2,2]->%4d\n",*(*(matr+2)+2));
  printf("[1,1]->%4d\n", matr[1][1]);
  printf("[1,5]->%4d\n", matr[1][5]);
  printf("[2,2]->%4d\n", matr[2][2]);
  return 0;
}
```

Результат работы:

```
[1,1]->   3
[1,5]->   2
[2,2]->   5
[1,1]->   3
[1,5]->   2
[2,2]->   5
```

Используя указатели и динамическое выделение памяти сконструируем зубчатый массив специального вида - нижнюю треугольную матрицу размера *n* (*n* вводится с клавиатуры), заполненную числовыми значениями по следующему правилу:

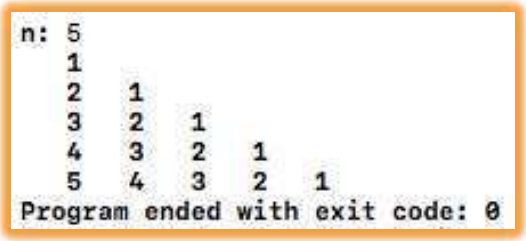
```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

Пример. Программа динамического формирования нижней треугольной матрицы.

```
int main() {
  int i, j, n;
```

```
  int **t;
  printf("n: "); scanf("%d", &n);
  t = (int **)calloc(n, sizeof(int*));
  for(i = 0; i < n; i++) {
    t[i] = (int *)calloc(i+1, sizeof(int));
    for(j = 0; j <= i; j++) t[i][j] = i-j+1;
  }
  for(i = 0; i < n; i++) {
    for(j = 0; j <= i; j++)
      printf("%4d",t[i][j]);
    printf("\n");
  }
  return 0;
}
```

Результат работы:



5.12 Аргументы командной строки

В операционной среде, обеспечивающей поддержку C, имеется возможность передать аргументы (параметры) запускаемой программе с помощью командной строки. В момент вызова функции `main` она получает два аргумента. В первом, обычно называемом `argc` (от argument count – счетчик аргументов), содержится число, равное количеству аргументов, заданных в командной строке. Второй, `argv` (от argument vector – вектор аргументов), является указателем на массив символьных строк, содержащих сами аргументы, - по одному в строке.

Следующая программа, которая названа `factorial`, позволяет вычислить факториалы нескольких чисел, которые задаются как параметры вызова данной программы:

```
int fact(int n) {
```

```
  return n ? n * fact(n-1) : 1;
}
int main(int argc, char **argv) {
  int n;
  while(--argc > 0) {
    n = atoi(*++argv);
    printf("%2d! = %10d\n", n, fact(n));
  }
  return 0;
}
```

Результат работы:

```
iMac-Aleksandr:~ aleksandrivancenko$
/Users/aleksandrivancenko/factorial 4 6 8 10
 4! =         24
 6! =        720
 8! =      40320
10! =    3628800
```

5.13 Указатели на функции

В программировании широко применяется техника *функций обратного вызова* (callback functions). Функции обратного вызова – это указатели на функции, через которые могут быть вызваны функции.

Пример. Программа, в которой используется объект `pf` типа «указатель на функцию» и через него происходит вызов разных функций:

```
int f_one(int k) {
  printf("Call f_one : k = %6d\n", k);
  return 100*k;
}
int f_two(int k) {
  printf("Call f_two : k = %6d\n", k);
  return 1000*k;
}
int main() {
  int n = 2;
```

```
  int (*pf)(int) = f_one;
  printf("1: %6d\n", pf(n++));
  pf = f_two;
  printf("2: %6d\n", pf(n++));
  return 0;
}
```

Результат работы:

```
Call f_one : k =      2
1:    200
Call f_two : k =      3
2:   3000
Program ended with exit code: 0
```

В предыдущем примере был использован объект `pf` типа «указатель на функцию» как самостоятельная переменная. Синтаксис языка С допускает объявить указатель на функцию как тип данных, что позволяет использовать динамические переменные и массивы указателей.

Пример. Программа, в которой объявляется тип «указатель на функцию» и он используется в дальнейшем для объявления массива:

```
int f_one(int k) {
  printf("Call f_one : k = %d\n", k);
  return 100*k;
}
int f_two(int k) {
  printf("Call f_two : k = %d\n", k);
  return 1000*k;
}
typedef int (*PF)(int);
int main() {
  int n = 2;
  PF p[] = {f_one, f_two, f_two, f_one};
  for (int i = 0; i < sizeof(p)/sizeof(p[0]); i++)
    printf("%d \n", p[i](n++));
  return 0;
}
```

Результат работы:

```
Call f_one : k = 2
200
Call f_two : k = 3
3000
Call f_two : k = 4
4000
Call f_one : k = 5
500
Program ended with exit code: 0
```

6. Структуры и объединения

6.1 Структуры

Структура – это набор нескольких переменных, объединённых в единое целое под общим именем. Входящие в структуру переменные называют элементами, полями или членами структуры. Членами структуры могут быть объекты произвольных типов, а их имена должны быть уникальны в пределах структуры. Доступ к членам структуры производится по составному имени, включающему имя структуры и имя члена структуры.

Пример. Объявление структуры и доступ к членам структуры:

```
struct {int x; int y;} pt1, pt2;
pt1.x = 3; pt1.y = 6;
pt2 = pt1;
```

Синтаксис языка С позволяет отделить описание «внутреннего устройства» структуры от определения конкретных переменных, имеющих это внутреннее устройство:

```
struct point {int x; int y;};
struct point pt1, pt2;
```

Такой подход позволяет рассматривать первое объявление как новый, структурный тип данных, а второе – как определение переменных этого типа. Допускается совмещать объявление структурного типа и объявление переменных этого типа:

```
struct point {int x; int y;} pt1, pt2;
```

Членами структуры могут быть структуры. Например, можно объявить структурный тип `rect` (прямоугольник) с двумя полями структурного типа `point` (точка):

```
struct rect { struct point p1, p2; };
```

6.2 Указатели на структуры и массивы структур

Структурные типы данных так же, как и обычные типы данных, могут использоваться для конструирования указателей и массивов.

Пусть `struct T` – структурный тип данных, тогда:

- `struct T x` – описывает переменную типа `struct T`, или просто структуру `x`,
- `struct T *p` – описывает указатель на структуру типа `struct T`,
- `struct T m[5]` – описывает массив из пяти элементов структурного типа `struct T`,
- `struct T *pm[]` – описывает массив неопределенного размера из указателей на объекты структурного типа `struct T`,
- `struct T (*pm)[]` – описывает указатель на массив неопределенного размера из элементов структурного типа `struct T` и т. д.

Указатели на структуры особенно удобны при передаче аргументов-структур в функции, т. к. приводят к экономии памяти: вместо копирования структуры из аргумента в функцию передается адрес структуры.

При использовании указателей на структуры язык С имеет специальный синтаксис для обращения к членам структуры через указатель на структуру. Пусть объявлен структурный тип `point`, конкретная структура `pt` этого типа и указатель `pp`:

```
struct point { int x, y;};
struct point pt, *pp = &pt;
```

Обычный синтаксис доступа к членам структуры `pt` через указатель `pp` выглядит так (скобки нужны потому, что приоритет операции `.` выше, чем у операции `*`):

```
(*pp).x = 3; (*pp).y = 5;
```

С использованием специального синтаксиса (операции «стрелка») эти же операторы можно записать более наглядно:

```
pp->x = 3; pp->y = 5;
```

6.3 Инициализация структур

При объявлении структуры ее поля могут быть инициализированы с использованием списка инициализации. Порядок размещения значений в таком списке и их типы должны соответствовать порядку расположения и типам полей в структуре.

Пример. Программа проверяет принадлежность точки прямоугольнику с использованием функции `pt_in_rect`:

```
struct point { int x, y; };
struct rect {
  struct point pt1, pt2;
};
int pt_in_rect(struct point p, struct rect r) {
  if(p.x < r.pt1.x || p.x > r.pt2.x) return 0;
  if(p.y < r.pt1.y || p.y > r.pt2.y) return 0;
  return 1;
}
int main() {
  struct point z = {2, 2};
  struct rect rc = {{1, 1}, {4, 5}};
  printf("%s\n",
    pt_in_rect(z, rc) ? "inside" : "outside");
  return 0;
}
```

Допускается инициализация членов структуры по их именам (а не по порядку):

```
struct rect rc = {.pt2={4, 5}, .pt1={1, 1}};
```

6.4 Анонимные структуры

Последние версии языка С позволяют создавать анонимные (безымянные) структуры. Эта возможность используется для случаев, когда членом структуры является структура и создавать для нее специальный структурный тип нецелесообразно.

Пример. Программа демонстрирует использование анонимной структуры и инициализацию членов структуры по именам:

```
struct person {
  int id;
  struct {char first[20]; char last[20];};
};
int main() {
  struct person man = {.last = "Никитин", .id = 3452,
      .first = "Афанасий"};
  puts(man.first);
  puts(man.last);
  return 0;
}
```

Результат работы:

```
Афанасий
Никитин
Program ended with exit code: 0
```

6.5 Структуры и функции

В приведенной выше программе, проверяющей принадлежность точки прямоугольнику, структуры использовались в качестве аргументов функции. Следующая программа демонстрирует, что структура может быть и возвращаемым из функции значением.

Пример. Программа находит минимальный прямоугольник, покрывающий заданное множество прямоугольников (рис. 6.1).

```
struct point { int x, y; };
struct rect { struct point pt1, pt2; };
struct rect max_rect(struct rect *r, int n) {
  struct rect rm = *r++;
  for(int i = 1; i < n; i++, r++) {
    if(r->pt1.x < rm.pt1.x) rm.pt1.x = r->pt1.x;
    if(r->pt1.y < rm.pt1.y) rm.pt1.y = r->pt1.y;
    if(r->pt2.x > rm.pt2.x) rm.pt2.x = r->pt2.x;
    if(r->pt2.y > rm.pt2.y) rm.pt2.y = r->pt2.y;
```

```
  }
  return rm;
}
int main() {
  struct rect rc[] = {{{1, 1}, {4, 5}},
                      {{2, 0}, {6, 6}},
                      {{3, 2}, {7, 4}}};
  int n = sizeof(rc) / sizeof (*rc);
  struct rect rcm = max_rect(rc, n);
  printf("x1=%d, y1=%d, x2=%d, y2=%d\n",
     rcm.pt1.x, rcm.pt1.y, rcm.pt2.x, rcm.pt2.y);
  return 0;
}
```

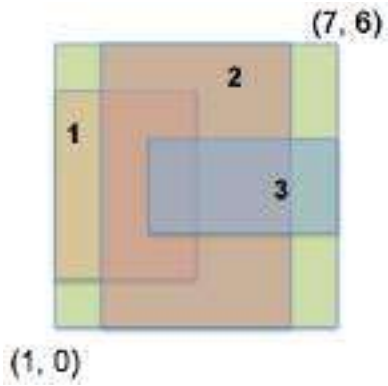


Рис. 6.1. Иллюстрация к поиску минимального покрывающего прямоугольника

Результат работы:

```
x1=1, y1=0, x2=7, y2=6
Program ended with exit code: 0
```

Текст приведенной выше программы можно немного сократить и сделать более «прозрачным», если воспользоваться возможностью создания синонимов для «громоздких» типов с помощью оператора `typedef` (рис. 6.2).

Пример. Программа с синонимом типа:

```
typedef struct {
  struct {int x, y;} pt1, pt2;
} Rect;
Rect max_rect(Rect* r, int n) {
  Rect rm = *r++;
```

```
  for(int i = 1; i < n; i++, r++) {
    if(r->pt1.x < rm.pt1.x) rm.pt1.x = r->pt1.x;
    if(r->pt1.y < rm.pt1.y) rm.pt1.y = r->pt1.y;
    if(r->pt2.x > rm.pt2.x) rm.pt2.x = r->pt2.x;
    if(r->pt2.y > rm.pt2.y) rm.pt2.y = r->pt2.y;
  }
  return rm;
}
int main() {
  Rect rc[] = {{{1, 1}, {4, 5}},
               {{2, 0}, {6, 6}},
               {{3, 2}, {7, 4}}};
  int n = sizeof(rc) / sizeof (*rc);
  Rect rcm = max_rect(rc, n);
  printf("x1=%d, y1=%d, x2=%d, y2=%d\n",
     rcm.pt1.x, rcm.pt1.y, rcm.pt2.x, rcm.pt2.y);
  return 0;
}
```

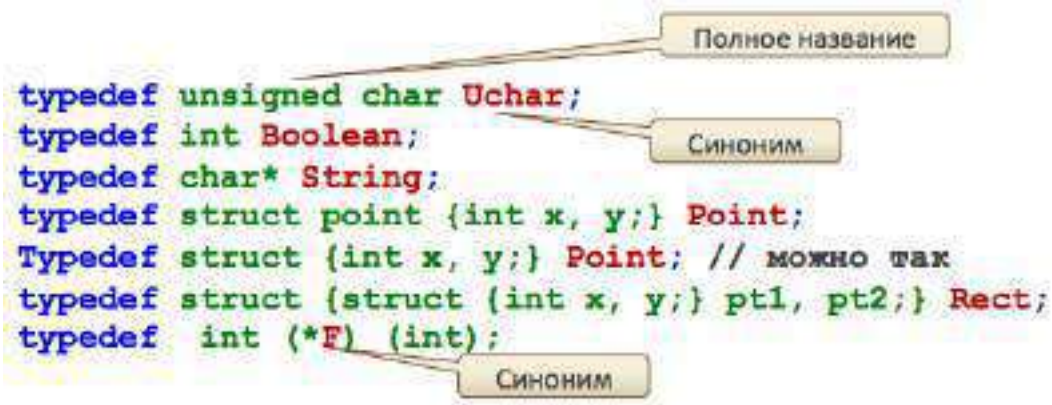


Рис. 6.2. Примеры оператора `typedef`

6.6 Объединения

Помимо структур (struct) в языке С имеется еще один вид конструкций, позволяющих собрать под одним именем разнотипные элементы – это объединения (union). Синтаксически объединения практически ничем не отличаются от структур, кроме использования ключевого слова `union` вместо ключевого слова `struct`.

Принципиальное отличие объединений от структур зключается в том, что все элементы объединения размещаются в памяти с одного адреса и перекрывают друг друга. Вследствие этого размер объединения в байтах определяется размером самого большого поля (возможно, с учетом необходимого выравнивания) и в каждый момент времени объединение может хранить только одно поле.

Пример. Размер объединения `u` равен размеру его самого большого поля `ldval`.

```
int main() {
  union {
    int ival;
    long double ldval;
    char *sval;
  } u;
  printf("Size of u = %d \n", sizeof(u));
  return 0;
}
```

Результат работы:

```
Size of u = 16
Program ended with exit code: 0
```

Пример. Размер объединения `p` больше размера его самого большого поля `c[5]` из-за необходимости выравнивания на границу, кратную размеру поля `f` типа `float`:

```
int main() {
  union {
    char c[5]; // 5 байт
    float f; // 4 байта
  } p = {.f = 1.23};
  printf("Размер объединения p = %zu\n", sizeof p);
  return 0;
}
```

Результат работы:

```
Размер объединения p = 8
Program ended with exit code: 0
```

Пример. Демонстрация механизма наложения полей объединения в результате их размещения в памяти с одного адреса:

```
int main() {
  union {
    uint64_t u64;
    uint32_t u32[2];
    uint16_t u16[2];
    uint8_t  u8;
  } s = {0x123456789abcdef};
  printf("Размер объединения s равен %zu и в нем
        содержится %zx\n", sizeof s, s.u64);
  printf("При этом остальные поля содержат:\n");
  printf("u32[0] = %x, u32[1] = %x\n", s.u32[0],
      s.u32[1]);
  printf("u16[0] = %x, u16[1] = %x\n", s.u16[0],
      s.u16[1]);
  printf("u8 = %x\n-------\n", s.u8);
  s.u16[0] = 0x3377;
  printf("После изменения поля u16[0] получим:\n");
  printf("u32[0] = %x, u32[1] = %x\n", s.u32[0],
      s.u32[1]);
  printf("u16[0] = %x, u16[1] = %x\n", s.u16[0],
      s.u16[1]);
  printf("u8 = %x\n-------\n", s.u8);
  return 0;
}
```

Результат работы:

```
Размер объединения s равен 8 и в нем содержится 123456789abcdef
При этом остальные поля содержат:
u32[0] = 89abcdef, u32[1] = 1234567
u16[0] = cdef, u16[1] = 89ab
u8 = ef
-------
После изменения поля u16[0] получим:
u32[0] = 89ab3377, u32[1] = 1234567
u16[0] = 3377, u16[1] = 89ab
u8 = 77
-------
Program ended with exit code: 0
```

Анализ результатов последнего примера позволяет установить, что для архитектуры компьютера, но котором выполнялась программа, нумерация байтов памяти производится справа налево и поэтому поля объединения `s` «выровнены» по правому краю. На этот факт указывает значение 8-битного поля `u8`, которое совпадает с младшим (крайним правым) байтом 64-битного поля `u64`.

Список литературы

1. Кетков Ю. Л. Введение в языки программирования C и C++ [Электронный ресурс]: Ю. Л. Кетков – М.:Национальный Открытый Университет «ИНТУИТ», 2008. - 252 с. - Режим доступа: http://www.knigafund.ru/books/176103
2. Баженова И. Ю., Сухомлин В. А. Введение в программирование [Электронный ресурс]: И. Ю. Баженова, В. А. Сухомлин– М.:Национальный Открытый Университет «ИНТУИТ», 2007. - 327 с. - Режим доступа: http://www.knigafund.ru/books/178416
3. Кнут Д. Искусство программирования, том 1. Основные алгоритмы, 3 изд.: Пер. с англ. – М.: Издат. дом «Вильямс», 2000. – 736 с., ил.
4. Керниган Б. В., Ричи Д. М. Язык программирования C [Электронный ресурс]: Б. В. Керниган, Д. М. Ричи – М.:Национальный Открытый Университет «ИНТУИТ», 2006. - 272 с. - Режим доступа: http://www.knigafund.ru/books/176845
5. Дейтел П., Дейтел Х. C для программистов с введением в C11 // Пер. с англ. – М.: ДМК Пресс, 2014. – 544 с., ил.
6. Дейтел Х., Дейтел П. Как программировать на C: Седьмое издание// Пер. с англ. – М.: Изд-во Бином, 2014. – 1000 с., ил.
7. Prata S. C Primer Plus (6th Edition) (Developer's Library). – Addison-Wesley, 2014. – 1037 p.
8. C reference [Электронный ресурс]: Режим доступа: http://en.cppreference.com/w/c

Учебное издание

Иванченко Александр Николаевич
Масленников Алексей Александрович
Иванченко Павел Александрович

ОСНОВЫ ПРОГРАММИРОВАНИЯ
(ЯЗЫК С)
Учебное пособие

Редактор Н.А. Юшко.
Подписано в печать 23.08.2016 . Формат 60х84 1/16.
Бумага офсетная. Печать цифровая. Усл. печ. л. 5,11 .
Уч.-изд.л. 5,00 . Тираж 100 экз. Заказ _____.

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический
университет (НПИ) имени М. И. Платова

Селезнев С.А.

ОРГАНИЗАЦИЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Методические указания к практическим занятиям, лабораторным работам и самостоятельной работе по курсу

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 004.382.7(076.5)
ББК 32.973.23я7

Рецензент – **С. П. Воробьев,** кандидат технических наук, доцент, доцент кафедры «Информационные и измерительные системы и технологии»

Селезнев С.А.

Организация электронных вычислительных машин: методические указания к практическим занятиям, лабораторным работам и самостоятельной работе по курсу / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2017. – 24с.

В данном издании представлены методические указания к практическим занятиям, лабораторным работам и самостоятельной работе по курсу «Организация электронных вычислительных машин».

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 004.382.7(076.5)

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Создание звука. Создание последовательности звуковых тонов. (Алгоритмы программной реализации)	4	Защита отчета	15÷20.03	[1-6]
2	Управление клавиатурой. (Алгоритмы программной реализации)	4	Защита отчета	15÷20.04	[1-6]
3	Управление выводом на дисплей. (Алгоритмы программной реализации)	4	Защита отчета	15÷20.05	[1-6]
4	Управление курсором (Алгоритмы программной реализации)	6	Защита отчета	15÷20.05	[1-6]

Практическое занятие № 1
СОЗДАНИЕ ЗВУКА. СОЗДАНИЕ ПОСЛЕДОВАТЕЛЬНОСТИ ЗВУКОВЫХ ТОНОВ

Цель работы: изучение принципов работы таймеров и практическое освоение алгоритмической реализации программной генерации звуковых последовательностей с помощью таймера.

Основные сведения о микросхеме таймеров 8253/8254. Таймеры 8253 и 8254 состоят из трех независимых каналов, или счетчиков. Каждый канал содержит регистры:

- состояния канала RS (8 разрядов);
- управляющего слова RSW (8 разрядов);
- буферный регистр OL (16 разрядов);
- регистр счетчика CE (16 разрядов);
- регистр констант пересчета CR (16 разрядов).

Каналы таймера подключаются к внешним устройствам при помощи трех линий:

- GATE - управляющий вход;
- CLOCK - вход тактовой частоты;
- OUT - выход таймера.

Регистр счетчика CE работает в режиме вычитания. Его содержимое уменьшается по заднему фронту сигнала CLOCK при условии, что на вход GATE установлен уровень логической 1. В зависимости от режима работы таймера при достижении счетчиком CE нуля тем или иным образом изменяется выходной сигнал OUT. Буферный регистр OL предназначен для запоминания текущего содержимого регистра счетчика CE без остановки процесса счета. После запоминания буферный регистр доступен программе для чтения.

Регистр констант пересчета CR может загружаться в регистр счетчика, если это требуется в текущем режиме работы таймера. Регистры состояния канала и управляющего слова предназначены, соответственно, для определения текущего состояния канала и для задания режима работы таймера. Упрощенная схема взаимодействия регистров канала приведена на рис.1.1.

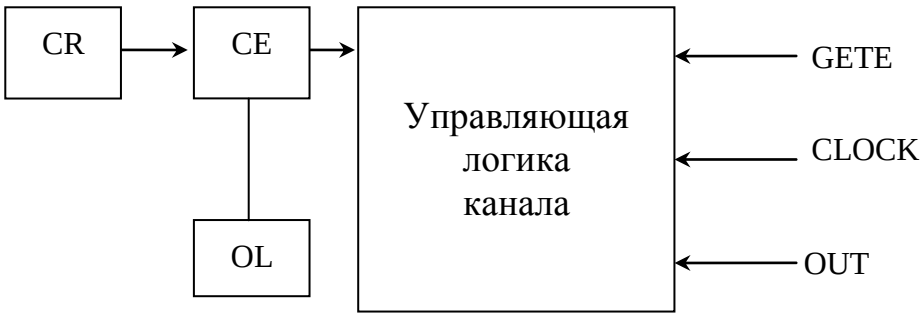


Рис. 1.1. Упрощенная схема взаимодействия регистров ка-

Возможны шесть режимов работы таймера. Они разделяются на три типа:

1. Режимы 0, 4 - однократное выполнение функций.
2. Режимы 1, 5 - работа с перезапуском.
3. Режимы 2, 3 - работа с автозагрузкой.

В режиме однократного выполнения функций перед началом счета содержимое регистра констант пересчета CR переписывается в регистр счетчика CE по сигналу CLOCK, если сигнал GATE установлен в 1. В дальнейшем содержимое регистра CE уменьшается по мере прихода импульсов CLOCK. Процесс счета можно приостановить, если подать на вход GATE уровень логического 0. Если затем на вход GATE подать 1, счет будет продолжен дальше. Для повторения выполнения функции необходима новая загрузка регистра CR, т.е. повторное программирование таймера.

При работе с перезапуском не требуется повторного программирования таймера для выполнения той же функции. По фронту сигнала GATE значение константы из регистра CR вновь переписывается в регистр CE, даже если текущая операция не была завершена.

В режиме автозагрузки регистр CR автоматически переписывается в регистр CE после завершения счета. Сигнал на выходе OUT появляется только при наличии на входе GATE уровня логической 1. Этот режим используется для создания программируемых импульсных генераторов и генераторов прямоугольных импульсов (меандра). В компьютере IBM PC/XT/AT/PS2 задействованы все три канала таймера.

Канал 0 используется в системных часах времени суток (не следует путать с часами реального времени, реализованными на другой микросхеме). Этот канал работает в режиме 3 и используется как генератор импульсов с частотой примерно 18.2 Гц. Именно эти импульсы вызывают аппаратное прерывание INT 8h.

Канал 1 используется для регенерации содержимого динамической памяти компьютера. Выход канала OUT используется для запроса к каналу прямого доступа DMA, который и выполняет обновление содержимого памяти. Вам не следует перепрограммировать этот канал, так как это может привести к нарушениям в работе основной оперативной памяти компьютера.

Канал 2 подключен к громкоговорителю компьютера и может быть использован для генерации различных звуков. Канал использует режим 3 таймера 8253/8254.

Канал 2 микросхемы 8254 связан с громкоговорителем компьютера. Однако громкоговоритель не просто соединен с выходом OUT канала 2. Порт вывода 61h также используется для управления громкоговорителем. Младший бит порта 61h подключен к входу GATE канала 2 таймера. Этот бит при установке в 1 разрешает работу канала, т.е. генерацию импульсов для громкоговорителя. Дополнительно для управления громкоговорителем используется бит 1 порта 61h. Если этот бит установлен в 1, импульсы от канала 2 таймера смогут проходить на громкоговоритель.

Таким образом, для включения звука надо выполнить следующие действия:

- запрограммировать канал 2 таймера на нужную частоту (т.е. загрузить регистр счетчика канала нужным значением);
- для включения звука установить в 1 два младших бита порта 61h.

Так как остальные 6 битов порта 61h используются для других целей, установка младших битов должна выполняться таким образом, чтобы значения остальных битов не были изменены. Для этого вначале надо считать байт из порта 61h в рабочую ячейку памяти, установить там нужные биты, затем вывести новое значение байта в порт 61h. Для выключения звука надо сбросить два младших бита порта 61h в 0 (при этом нельзя изменять значение остальных битов этого порта).

Функция, обеспечивающая генерацию на громкоговорителе тона заданной частоты и длительности, имеет вид:

```
/**
*.Name        tm_sound
*.Title      Формирование тона заданной длительности
*.Descr    Эта функция предназначена для генерации
*  на громкоговорителе тона заданной  длительности и частоты.
*.Proto       void tm_sound(int freq, int time);
*.Params       int freq - частота в герцах;
*   int time - длительность в тиках  таймера (за одну секунду таймер тикает 18.2 раза).
*.Return       Ничего
*.Sample       play.c
**/
#include <stdio.h>
#include <conio.h>
#include "sysp.h"
void tm_sound(int freq, int time) {
      int cnt, i;
// Задаем режим канала 2 таймера
      outp(0x43, 0xb6);
// Вычисляем задержку для загрузки в регистр счетчика таймера
      cnt = 1193180L / freq;
// Загружаем регистр счетчика таймера - сначала
// младший, затем старший байты
      outp(0x42, cnt & 0x00ff);
      outp(0x42, (cnt &0xff00) >> 8);
// Включаем громкоговоритель. Сигнал от
// канала 2 таймера теперь будет проходить
// на вход громкоговорителя.
      outp(0x61, inp(0x61) | 3);
```

```
// Выполняем задержку.
      tm_delay(time);
// Выключаем громкоговоритель.
 outp(0x61, inp(0x61) & 0xfc);
}
```

Задание. Разработать блок-схему алгоритма, реализующей на громкоговорителе персонального компьютера звуковое воспроизведение последовательности шести кодов Морзе символов, определенных номером варианта в табл. 1.

Таблица 1.

Варианты											
№ 1		№ 2		№ 3		№ 4		№ 5		№ 6	
A	•—	G	——•	M	——	S	•••	Y	—•——	5	•••••
B	—•••	H	••••	N	—•	T	—	Z	——••	6	—••••
C	—•—•	I	••	O	———	U	••—	1	•————	7	——•••
D	—••	J	•———	P	•——•	V	•••—	2	••———	8	———••
E	•	K	—•—	Q	——•—	W	•——	3	•••——	9	————•
F	••—•	L	•—••	R	•—•	X	—••—	4	••••—	0	—————

Отчет должен содержать:

1. Цель работы.
2. Описание работы таймеров. 8253/8254
3. Словесная постановка задачи звукового кодирования символов.
4. Блок схема алгоритма решения задачи звукового кодирования символов.
5. Описание алгоритм работы программы.

Практическое занятие № 2
УПРАВЛЕНИЕ КЛАВИАТУРОЙ

Цель работы: изучение принципов работы клавиатуры и практическое освоение алгоритмической реализации управления работой программы с помощью ввода символов от клавиатуры.

Принципы работы клавиатуры. Клавиатура выполнена, как правило, в виде отдельного устройства, подключаемого к компьютеру тонким кабелем. Малогабаритные компьютеры могут использовать встроенную клавиатуру. На рис.2.1. показана сильно упрощенная принципиальная схема клавиатуры. Внутри клавиатуры установлена микросхема процессора, выполняющего специализированные функции контроллера клавиатуры. Он отслеживает нажатия на клавиши и посылает номер нажатой кла-

виши в центральный компьютер. Все клавиши клавиатуры находятся в узлах матрицы

Все горизонтальные линии матрицы подключены через резисторы к источнику питания +5 В. Клавиатурный процессор имеет два порта - выходной и входной. Входной порт подключен к горизонтальным линиям матрицы (X_0-X_4), а выходной - к вертикальным (Y_0-Y_5).

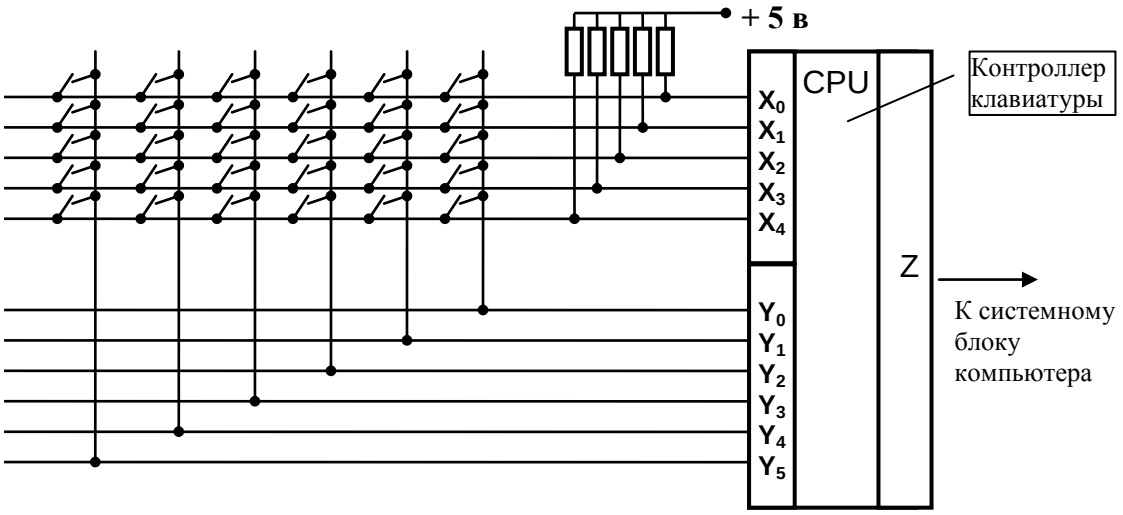


Рис. 2.1Упрощенная схема клавиатуры

Устанавливая по очереди на каждой из вертикальных линий уровень напряжения, соответствующий логическому 0, контроллер клавиатуры опрашивает состояние горизонтальных линий. Если ни одна клавиша не нажата, уровень напряжения на всех горизонтальных линиях соответствует логической 1 (т.к. все эти линии подключены к источнику питания +5 В через резисторы).

Если оператор нажмет на какую-либо клавишу, то соответствующая вертикальная и горизонтальная линии окажутся замкнутыми. Когда на этой вертикальной линии процессор установит значение логического 0, то уровень напряжения на горизонтальной линии также будет соответствовать логическому 0.

Как только на одной из горизонтальных линий появится уровень логического 0, клавиатурный процессор фиксирует нажатие на клавишу. Он посылает в центральный компьютер запрос на прерывание и номер клавиши в матрице. Аналогичные действия выполняются и тогда, когда оператор отпускает нажатую ранее клавишу.

Номер клавиши, посылаемый клавиатурным процессором, однозначно связан с распайкой клавиатурной матрицы и не зависит напрямую от обозначений, нанесенных на поверхность кла-

виш. Этот номер называется скан-кодом (Scan Code). Слово scan (сканирование), подчеркивает тот факт, что клавиатурный компьютер сканирует клавиатуру для поиска нажатой клавиши.

Но программе нужен не порядковый номер нажатой клавиши, а соответствующий обозначению на этой клавише ASCII-код. Этот код не зависит однозначно от скан-кода, т.к. одной и той же клавише могут соответствовать несколько значений ASCII-кода. Это зависит от состояния других клавиш. Например, клавиша с обозначением '1' используется еще и для ввода символа '!' (если она нажата вместе с клавишей SHIFT).

Поэтому все преобразования скан-кода в ASCII-код выполняются программным обеспечением. Как правило, эти преобразования выполняют модули BIOS центрального процессора. Для использования символов кириллицы эти модули расширяются клавиатурными драйверами.

Если нажать на клавишу и не отпускать ее, клавиатура перейдет в режим автоповтора. В этом режиме в центральный компьютер автоматически через некоторый период времени, называемый периодом автоповтора, посылается код нажатой клавиши. Режим автоповтора облегчает ввод с клавиатуры большого количества одинаковых символов.

Следует отметить, что клавиатура содержит внутренний 16-байтовый буфер, через который она осуществляет обмен данными с компьютером.

Используемые функции библиотеки Microsoft C. Стандартные библиотеки трансляторов Microsoft C 6.0 содержат набор функций, предназначенных для работы с клавиатурой. Самые простые из них - getch() и getche(). Они описаны в файле conio.h.

Функция getch() имеет следующий прототип: **int getch(void).** Эта функция возвращает ASCII-код прочитанного из клавиатурного буфера символа, причем прочитанный символ не отображается на экране. Если была нажата функциональная клавиша или клавиша перемещения курсора, функция возвращает 0. В этом случае функцию надо вызвать еще раз для получения расширенного ASCII-кода нажатой клавиши.

Функция обрабатывает клавиши Ctrl-C и Ctrl-Break - при вводе этих комбинаций клавиш работа программы завершается.

Если клавиатурный буфер пуст, программа переводится в состояние ожидания.

Функция getche() полностью аналогична функции getch(), за исключением того, что прочитанный символ отображается на экране. Прототип функции имеет вид: getche(): **int getche(void).**

Пример программы, отображающей на экране ASCII-коды и расширенные ASCII-коды нажимаемых клавиш, представлен ниже:

```
#include <conio.h>
#include <ctype.h>
#include <stdio.h>
void main() {
        int key;
// Читаем в цикле символы с клавиатуры и отображаем
// ASCII-коды нажатых клавиш.
// Выходим из цикла при нажатии не клавишу ESC
        for(;;) {
// Читаем символ
       key = getch();
// Если прочитанный символ равен 0, вызываем функцию getch()
// для получения расширенного ASCII-кода нажатой клавиши
                if( (key == 0) || (key == 0xe0) ) {
                        key = getch();
                        printf( "Расширенный ASCII-код:\t" );
                }
                else printf( "ASCII-код:\t");
                printf("%d\n",key);
// При нажатии на клавишу ESC выходим из цикла
                if( key == 27) break;
        }
}
```

Для проверки буфера клавиатуры на наличие символов можно использовать функцию kbhit(). Она также описана в файле conio.h. Прототип функции kbhit(): **int kbhit(void).** Если буфер клавиатуры не пуст, функция возвращает ненулевое значение. В этом случае программа может прочитать символы из буфера клавиатуры при помощи фукнкций getch() и getche(). Если буфер клавиатуры пуст, функция возвращает нулевое значение.

Ниже приведен пример программы, ожидающей нажатия на любую клавишу. Во время ожидания программа выводит на экран поочередно символы "<" и ">":

```
#include <conio.h>
void main() {
	int key;
// Ожидаем нажатия на любую клавишу.
// Во время ожидания выводим на экран поочередно
// символы "<" и ">"
	while(!kbhit()) printf("<\b>\b");
// Как только будет нажата какая-нибудь клавиша,
// выводим ее ASCII-код
		key = getch();
// Если прочитанный символ равен 0, вызываем функцию getch()
// для получения расширенного ASCII-кода нажатой клавиши
		if( (key == 0) || (key == 0xe0) ) {
			key = getch();
			printf( "Расширенный ASCII-код:\t" );
		}
		else printf( "ASCII-код:\t");
		printf("%d\n",key);
}
```

Задание. Разработайте блок-схему программы, обеспечивающей:

1. Ввод символа, из множества, заданного соответствующим номером варианта практического занятия №1;
2. Воспроизведение звуковой последовательности кода Морзе, соответствующей введённому символу.
3. Игнорирование попытки ввода любого символа, не принадлежащего подмножеству, заданному вариантом.

Отчет должен содержать:

1. Цель работы.
2. Описание работы клавиатуры.
3. Словесная постановка задачи звукового кодирования вводимых символов.
4. Блок схема алгоритма решения задачи звукового кодирования вводимых символов.
5. Описание алгоритм работы программы.

Практическое занятие № 3
УПРАВЛЕНИЕ ВЫВОДОМ НА ДИСПЛЕЙ

Цель работы: изучение принципов организации вывода данных на дисплей и практическое освоение алгоритмической реализации отображения данных в программе с помощью простейшего вывода символов на дисплей

Используемые функции библиотеки Microsoft C. Вывод на дисплей через стандартный выходной поток. Функции putchar() и putc(). Эти функции помещают один символ в текущую позицию выходного потока. Прототипы функций представлены ниже:

int putc(int ch, FILE *stream);

int putchar(int ch);

Функция putc() отличается от putchar() наличием второго аргумента, который определяет выходной поток, в который помещается символ. Она может использоваться и для записи символов в открытый файл и для записи в стандартные потоки ввода-вывода. В частности, если второй аргумент функции равен константе stdout, то она эквивалентна функции putchar() и выводит символ в стандартный выходной поток. Этот поток может идти на экран дисплея или в файл, если используется свойство переназначения потока. Фактически функция putchar() является макроопределением: #define putchar(_c) putc((_c),stdout).

Первые параметры обеих функций совпадают по смыслу и задают ASCII-код выводимого символа (не смотря на то, что переменная ch целого типа, используется только ее младший байт). При этом функции обрабатывают управляющие символы. Список управляющих символов приведен в таблице 3.1.

Функции возвращают ASCII-код выведенного символа, а в случае ошибки возвращают константу EOF. Чтобы конкретизировать ошибку можно воспользоваться функцией ferror().

```
// использование функции putchar()
#include <stdio.h>
void main(void) {
  char *ptr,
    out_str[] = "\aputchar\nputc\a";
  for(ptr = out_str; *ptr; putchar(*(ptr++)) );
}
```

Таблица 3.1

Символ	Код ASCII	Значение
\'	27h	символ кавычки
\"	22h	двойная кавычка
\\	5Ch	обратный слеш
\%	25h	знак процента
\a	07h	звуковой сигнал
\b	08h	перемещение назад
\f	0Ch	переход к новой странице
\n	0Ah	новая строка
\r	0Dh	возврат каретки

Включаемый файл stdio.h содержит спецификации функций putchar() и putc(). Следовательно, этот файл необходимо включать во все программы, использующие данные функции.

Функция **puts()** позволяет записать строку в стандартный выходной поток stdout. Строка должна оканчиваться нулем (символ '\0'). Этот символ не помещается в выходной поток. Вместо него записывается упрвляющий символ '\n', который переводит курсор в начало новой строки. Функция имеет следующий прототип:

int puts(char *string);

Единственный параметр функции содержит указатель на отображаемую строку. При успешном выполнении функция возвращает ноль. Если же произошла ошибка, то возвращается ненулевая величина. При использовании функции puts() необходимо включить директивой include файл stdio.h, содержащий спецификацию этой функции.

```
// использование функци puts
#include <stdio.h>
void main(void) {
  puts("Работает функция puts!");
}
```

Задание. Разработайте блок-схему программы, обеспечивающей:

1. Ввод символа, из множества, заданного соответствующим номером варианта практического занятия №1.
2. Воспроизведение звуковой последовательности кода Морзе, соответствующей введённому символу.

3. Игнорирование попытки ввода любого символа, не принадлежащего подмножеству, заданному вариантом.
4. Вывод на дисплей каждого введенного символа с новой строки с отображением рядом его кода Морзе в виде последовательности точек и тире.

Отчет должен содержать:

1. Цель работы.
2. Словесная постановка задачи звукового кодирования и отображения на дисплеи вводимых символов.
3. Блок схема алгоритма решения задачи звукового кодирования и отображения на дисплеи вводимых символов.
4. Описание алгоритм работы программы.

Практическое занятие № 4.
УПРАВЛЕНИЕ КУРСОРОМ

Цель работы: изучение принципов организации вывода форматированных данных на дисплей и практическое освоение алгоритмической реализации отображения данных в программе с помощью структурированного вывода символов на дисплей.

Используемые функции библиотеки Microsoft C. Функция printf(). Наиболее универсальной стандартной функцией, обеспечивающей запись в выходной поток stdout, является функция printf(). Она производит вывод на экран дисплея строк, отдельных символов и чисел в различных форматах.

Прототип функции printf():

int printf(char *format [,argument]...);

Первый аргумент функции содержит указатель на управляющую строку, которая может состоять из:

- непосредственно выводимых символов;
- управляющих символов;
- спецификаторов формата.

Непосредственно выводимые символы и управляющие символы помещаются в выходной поток без изменения. Если управляющая строка содержит спецификаторы формата, то каждому спецификатору должен соответствовать один аргумент, следующий за управляющей строкой. Аргументы представляют собой

переменные или константы, которые преобразуются согласно соответствующих им спецификаторов формата и затем также помещаются в выходной поток.

В более общем случае, в спецификаторе формата между символом '%' и символом, определяющим тип преобразования можно разместить флаги и префиксы типов:

% [флаг] [ширина] [точность] [префикс типа] символ преобразования

Различные спецификаторы формата перечислены в таблице 4.1.

Таблица 4.1

Символ	Значение
%d, %i	целое десятичное число
%s	текстовая строка
%c	один символ
%e, %E	число с плавающей точкой в экспоненциальной форме
%f	число с плавающей точкой в десятичной записи
%u	целое десятичное число без знака
%o	целое восьмеричное число без знака
%x, %X	целое шестнадцатеричное число без знака
%g, %G	либо %e, либо %f в зависимости от того, запись в каком формате короче
%p	значение указателя
%n	число символов

Поле [ширина] задает минимальную ширину поля, используемую при печати строки или числа. Если это число или строка не помещаются в поле целиком, то ширина поля увеличивается.

Поле [точность] определяет для чисел количество отображаемых после запятой цифр, а для строк - максимальное число символов в строке. Флаги приведены в таблице 4.2, а префиксы типов в таблице 4.3.

Функция printf() возвращает число напечатанных ею символов или в случае ошибки - отрицательную величину. Использование различных спецификаторов формата демонстрируется следующим примером:

```
// при использовании printf также необходим файл stdio.h
#include <stdio.h>
void main(void) {
  int  char_count;
  char ptr[] = "строка";
  printf("\n%23d  \n%e  \n%.3E  \n%f  \n%+g  \n%-23G \n\n",
```

```
        1111,   0.01, 0.01,   0.01, 0.01,  0.0000001);
    printf("%s%n", ptr,  &char_count);
    printf("\n\nПредыдущая функция вывела %d символов.\n\n",
          char_count);
    printf("Это %s, расположенная по адресу %Lp.\n\a", ptr, ptr);
}
```

Таблица 4.2

Префикс	Значение
F	дальний указатель
N	ближний указатель
h	короткое целое
l, L	длинное целое

Таблица 4.3

Флаг	Значение
-	выравнивание по левой границе
+	перед числом помещается его знак
Пробел ' '	перед данными помещается пробел

Задание. Разработайте блок-схему программы, которая должна обеспечивать вывод на экран последовательности вводимых символов из множества, заданного соответствующим номером варианта практического занятия №1. При этом должны выполнятся следующие условия:

1. Все символы разбиваются на группы по пять штук.
2. В каждой строке выводится одна группа символов.
3. Внутри группы символы отделяются друг от друга тремя пробелами.
4. Каждый введенный символ отображается соответствующим знаком алфавита и кодом Морзе в виде последовательности точек и тире.

Отчет. Отчет должен содержать:

1. Цель работы.
2. Описание работы клавиатуры.
3. Словесная постановка задачи звукового кодирования вводимых символов.
4. Блок схема алгоритма решения задачи звукового кодирования вводимых символов.
5. Описание алгоритм работы программы.

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Создание звука. Создание последовательности звуковых тонов.	4	Защита отчета	15.03÷20.03	[1-6]
2	Управление клавиатурой.	4	Защита отчета	15.04÷20.04	[1-6]
3	Управление выводом на дисплей	4	Защита отчета	15.05÷20.05	[1-6]
4	Управление курсором	6	Защита отчета	15.05÷20.05	[1-6]

Лабораторная работа №1
Создание звука.
Создание последовательности звуковых тонов

Цель работы: практическое применение принципов работы таймера и программная реализация генерации звуковых последовательностей с помощью таймера.

Задание.

1. Разработать техническое задание (ТЗ) на создание программы, обеспечивающей звуковое воспроизведение последовательности кодов Морзе символов, определенных номером варианта в табл.1.1.
2. Разработать блок-схему программы, реализующей требования технического задания.
3. Написать и отладить программу, удовлетворяющую требованиям согласованного технического задания.
4. Использовать материалы и результаты практического занятия № 1

Отчет по лабораторной работе включает в себя:

- цель работы;
- согласованное Техническое задание на создание программы, обеспечивающей генерацию звуковых последовательностей с помощью таймера;
- блок-схему обобщенного алгоритма создаваемой программы;
- листинг программы, реализующей разработанный алгоритм.

Лабораторная работа №2
Управление клавиатурой

Цель работы: практическое освоение принципов работы клавиатуры путем разработки программы ввода символов от клавиатуры.

Задание.

1. Разработать техническое задание (ТЗ) на создание программы, обеспечивающей:
   - ввод символа, из множества, заданного соответствующим номером варианта практического занятия №1;
   - воспроизведение звуковой последовательности кода Морзе, соответствующей введённому символу;
   - игнорирование попытки ввода любого символа, не принадлежащего подмножеству, заданному вариантом.
2. Использовать материалы и результаты практического занятия № 2.
3. Разработать блок-схему программы, реализующей ввод символов и их обработку в соответствии с согласованным техническим заданием.
4. Написать и отладить программу, удовлетворяющую требованиям согласованного технического задания.

Отчет по лабораторной работе включает в себя:

- цель работы;
- согласованное техническое задание на создание программы;
- блок-схему обобщенного алгоритма создаваемой программы;
- листинг программы, реализующей разработанный алгоритм.

Лабораторная работа №3
Управление выводом на дисплей

Цель работы: практическое освоение принципов организации вывода данных на дисплей путем разработки программы, обеспечивающей простейший вывод символов на дисплей

Задание.

1. Разработать техническое задание (ТЗ) на создание программы, обеспечивающей:
   - ввод символа, из множества, заданного соответствующим номером варианта практического занятия №1;

– воспроизведение звуковой последовательности кода Морзе, соответствующей введённому символу;
– игнорирование попытки ввода любого символа, не принадлежащего подмножеству, заданному вариантом;
– вывод на дисплей каждого введенного символа с новой строки с отображением рядом его кода Морзе в виде последовательности точек и тире.

2. Использовать материалы и результаты практического занятия № 2.
3. Разработать блок-схему программы, реализующей ввод символов и их обработку в соответствии с согласованным техническим заданием.
4. Написать и отладить программу, удовлетворяющую требованиям согласованного технического задания.

Отчет по лабораторной работе включает в себя:

– цель работы;
– согласованное техническое задание на создание программы;
– блок-схему обобщенного алгоритма создаваемой программы;
– листинг программы, реализующей разработанный алгоритм.

Лабораторная работа №4
Управление курсором

Цель работы: практическое освоение принципов организации вывода форматированных данных на дисплей и путем разработки программы, обеспечивающей структурированный вывод символов на дисплей.

Задание.

1. Разработать техническое задание (ТЗ) на создание программы, обеспечивающей:
   – вывод на экран последовательности вводимых символов из множества, заданного соответствующим номером варианта практического занятия № 1.
   – все символы разбиваются на группы по пять штук;
   – в каждой строке выводится одна группа символов;
   – внутри группы символы отделяются друг от друга тремя пробелами;
   – каждый введенный символ отображается соответст-

вующим знаком алфавита и кодом Морзе в виде последовательности точек и тире.

2. Использовать материалы и результаты практического занятия № 4.
3. Разработать блок-схему программы, реализующей ввод символов и их обработку в соответствии с согласованным техническим заданием.
4. Написать и отладить программу, удовлетворяющую требованиям согласованного технического задания.

Отчет. Отчет по лабораторной работе включает в себя:

- цель работы;
- согласованное техническое задание на создание программы;
- блок-схема обобщенного алгоритма создаваемой программы;
- листинг программы, реализующей разработанный алгоритм.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 87,1 ч.

№	Наименование тем	Кол-во часов	Номер компетенции	Литература
1	**Тема № 1.** Многоядерная архитектура (Multicore). Достоинства и недостатки многоядерной архитектуры Проблемы Multicore	20	ОПК-2, ПК-19, ПК-20	[1-7]
2	**Тема № 2.** Топологии связей систем с массовым параллелизмом (ММР). Общие достоинства и недостатки	20	ОПК-2, ПК-19, ПК-20	[1-7]
3	**Тема № 3** Организация системных шин в компьютере. Типы шин по назначению. Основные характеристи983ыфы	20	ОПК-2, ПК-19, ПК-20	[1-7]
4	**Тема № 4.** Иерархическая организация памяти. Принцип локальности ссылок. Латентность памяти и тайминги. Виды таймингов памяти	27,1	ОПК-2, ПК-19, ПК-20	[1-7]

Задания для самостоятельного изучения и конспектирования

Тема 1. Рассмотреть следующие вопросы:
- понятие «Многоядерная архитектура» (Multicore);
- обобщенная структура многоядерного процессора;
- физическая реализация многоядерного процессора;
- достоинства многоядерной архитектуры;
- проблемы, возникающие при использовании многоядерной архитектуры.

Тема 2. Рассмотреть следующие вопросы:
- типы графов, соответствующие топологии сетей ММР;
- особенности различных топологий сетей ММР;
- достоинства сетей ММР;
- недостатки сетей ММР.

Тема 3. Рассмотреть следующие вопросы:
- классификация шин по функциональному назначению;
- конфигурация «Общей шины»;
- достоинства и недостатки общей шины;
- системные шины FSB, QPI и HyperTransport**;**
- достоинства и недостатки шин FSB, QPI и HyperTransport;
- последовательные шины Serial ATA и USB;
- достоинства и недостатки. шин Serial ATA и USB.

Тема 4. Рассмотреть следующие вопросы:
- иерархическая организация памяти;
- принцип локальности ссылок;
- латентность памяти и тайминги;
- виды таймингов памяти.

СРС экз. – самостоятельная работа по подготовке к экзамену в период экзаменационной сессии – 35.65 ч.

Экзаменационные вопросы по курсу

1. Классы ЭВМ по принципу действия.
2. Классы систем обработки данных (СОД).
3. Определение вычислительной системы (ВС).
4. Определение понятий: функция системы, структура системы, организация системы.

5. Поколения ЭВМ. Отличительные признаки поколений ЭВМ.
6. Типы компьютеров. Одноразовые компьютеры. Микроконтроллеры. Игровые компьютеры
7. Типы компьютеров. Персональные компьютеры, серверы, мэйнфреймы.
8. Структурная классификация современных компьютерных систем.
9. Система команд процессора. Обобщенный алгоритм выполнения команд процессором.
10. Понятие архитектуры команд процессора. RISC и CISC архитектура.
11. Организация системных шин в компьютере. Типы шин по назначению. Основные характеристики шин.
12. Иерархическая организация памяти. Принцип локальности ссылок.
13. Адресная память. Латентность памяти и тайминги. Виды таймингов памяти.
14. Ассоциативная память.
15. Организация кэш-памяти. Типы кэш-памяти. Кэш с прямым отображением.
16. Организация кэш-памяти. Типы кэш-памяти. Полностью ассоциативный кэш.
17. Организация кэш-памяти. Типы кэш-памяти. Множественно-ассоциативный кэш.
18. Целостность данных. Методы обеспечения целостности данных.
19. Конвейер операций. Принцип совмещения операций. Достоинства и недостатки.
20. Многофункциональная обработка. Суперскалярная обработка. VLIW-архитектура.
21. Векторная обработка. Матричная система. Векторно - конвейерный принцип обработки данных.
22. Многопроцессорные вычислительные системы. Сущность понятий: многомашинная ВС и многопроцессорная ВС.
23. SMP – архитектура. Основные варианты структуры. Достоинства и недостатки.
24. Мультипроцессорная ВС с равнодоступной модульной оперативной памятью и перекрестной коммутацией.

25. Топологии связей систем с массовым параллелизмом (ММР). Общие достоинства и недостатки.
26. Понятие Кластерной вычислительной системы. Основные свойства.
27. Многоядерная архитектура (Multicore). Проблемы Multicore.

Контактная внеаудиторная работа

СРС – групповые консультации с преподавателем в течение семестра – 0,9 ч.

Групповая консультация перед экзаменом – 2 ч.

СРС экз. – сдача экзамена – 0,35 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Цилькер Б.Я. Организация ЭВМ и систем : учебник / Б.Я. Цилькер, С.А. Орлов. - СПб. : Питер, 2004. - 668 с.
2. Гуров В.В., Чуканов В.О. Архитектура и организация ЭВМ. [Электронный ресурс] / Национальный открытый университет «Интуит», 2016.184с. - режим доступа http://www.knigafund.ru/

Дополнительная литература.

3. Пупков А.Н., Царев Р.Ю., Мыльникова Е.В. Информатика и программирование [Электронный ресурс] / Сибирский федеральный университет • 2014. 132 с. - режим доступа http://www.knigafund.ru/
4. Диков А.В. Компьютер изнутри [Электронный ресурс] / Директ-Медиа • 2015 г • 126 с. - режим доступа http://www.knigafund.ru/

Методические указания к лабораторным и практическим занятиям

5. Северов Д.С. Архитектура ЭВМ и язык ассемблера. Свободно доступный ресурс интернет-университета «Интуит». (www.intuit.ru/department/hardware/comparch/6).
6. Visual Studio. Среда разработки Visual Studio. Свободно доступный раздел MSDN Library. (http://msdn.microsoft.com/ru-ru/library/52f3sw5c(v=vs.90).aspx)

Учебно-методическое издание

Селезнев Сергей Анатольевич

ОРГАНИЗАЦИЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Методические указания
к практическим занятиям, лабораторным работам
и самостоятельной работе по курсу

Редактор *Я.В. Максименко*
Подписано в печать 20.04.2017.
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,39 ч.-изд.л. 1,5 Тираж _____ экз. Заказ_____ .

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ОСНОВЫ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНЫХ ПРОЕКТОВ

Методические указания к практическим занятиям, лабораторным работам и самостоятельной работе студентов

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 51:512(076.5)
ББК 22.1 я73

Рецензент

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

Дьяченко В.Б., Мохов В.А.

Основы разработки и реализации программных проектов: методические указания к практическим занятиям, лабораторным работам и самостоятельной работе студентов / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 29 с.

В данном издании представлены методические указания к практическим занятиям, лабораторным работам и самостоятельной работе по курсу «Основы разработки и реализации программных проектов».

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.04 Программная инженерия.

УДК 51:512(076.5)

1.ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

наименование и объем в часах

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Модель прецедентов	2	Тест	15-20.02	5-7
2	Модель классов	2	Тест	15-20.02	5-7
3	Модель сущность-связь	2	Тест	15-20.03	5-7
4	Построение проекта на базе каскадной модели	2	Тест	15-20.03	5-7
5	Построение проекта на базе спиральной модели	2	Тест	15-20.04	5-7
6	Построение проекта на базе MSF модели	2	Тест	15-20.04	5-7
7	Формирование команды на базе административной модели	2	Тест	15-20.05	5-7
8	Формирование команды на базе модели хаоса	2	Тест	15-20.05	5-7
9	Формирование команды на базе модели открытой архитектуры	2	Тест	15-20.05	5-7

Практическое занятие № 1

Диаграмма прецедентов (диаграмма вариантов использования) в UML — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью ***модели прецедентов***, позволяющей описать систему на концептуальном уровне (рис. 1.1).

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

При моделировании системы с помощью диаграммы прецедентов системный аналитик стремится:

- чётко отделить систему от её окружения;
- определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы;
- определить в глоссарии предметной области понятия, относящиеся к детальному описанию функциональности системы (то есть, прецедентов).

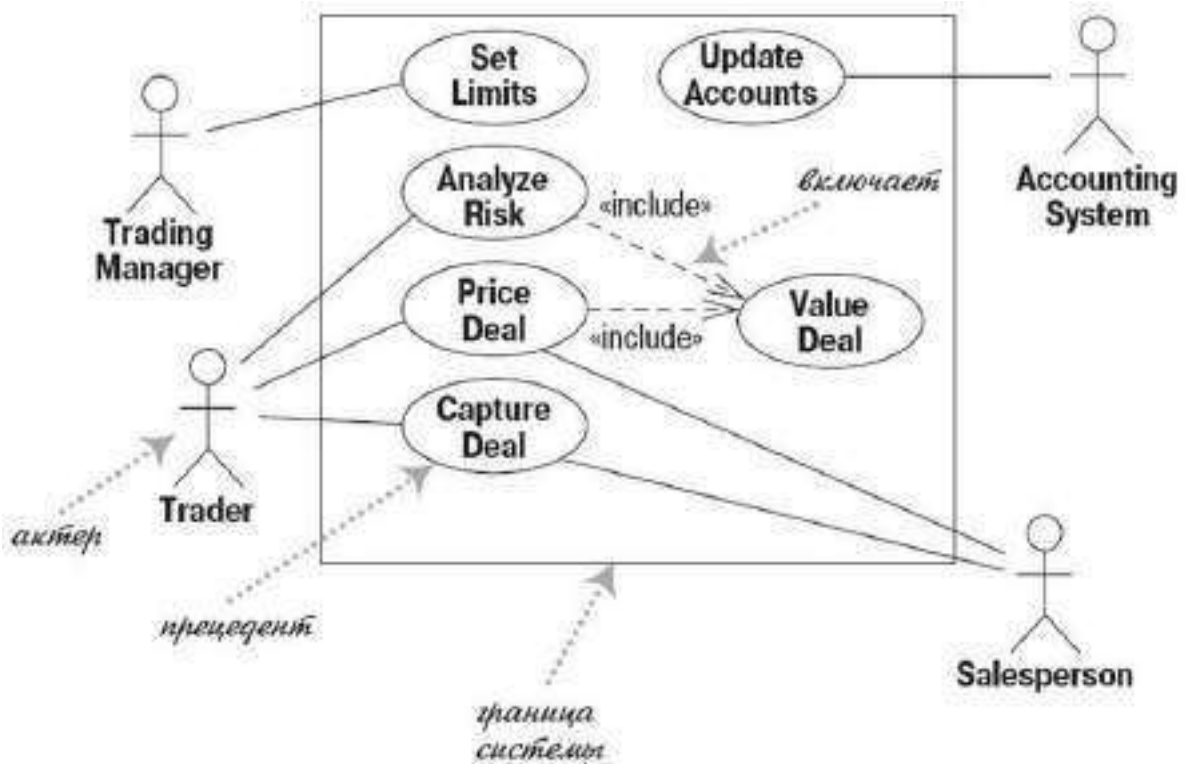


Рис. 1.1. Диаграмма прецедентов

Работа над диаграммой может начаться с текстового описания, полученного при работе с заказчиком. При этом нефункциональные требования (например, конкретный язык или система программирования) при составлении модели прецедентов опускаются (для них составляется другой документ).

Для отражения модели прецедентов на диаграмме используются:

•рамки системы (англ. *system boundary*)— прямоугольник с названием в верхней части и эллипсами (прецедентами) внутри. Часто может быть опущен без потери полезной информации;

•актёр (англ. *actor*) — стилизованный человечек, обозначающий набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), взаимодействующего с некоторой сущностью (системой, подсистемой, классом). Актёры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования);

•прецедент — эллипс с надписью, обозначающий выполняемые системой действия (могут включать возможные варианты), приводящие к наблюдаемым актёрами результатам. Надпись может быть именем или описанием (с точки зрения актёров) того, «что» делает система (а не «как»). Имя прецедента связано с непрерываемым (атомарным) *сценарием* — конкретной последовательностью действий, иллюстрирующей поведение. В ходе сценария актёры обмениваются с системой сообщениями. Сценарий может быть приведён на диаграмме прецедентов в виде UML-комментария. С одним прецедентом может быть связано несколько различных сценариев.

Отношения между прецедентами

Часть дублирующейся информации в модели прецедентов можно устранить указанием связей между прецедентами:

•***обобщение прецедента*** — стрелка с незакрашенным треугольником (треугольник ставится у более общего прецедента);

•***включение прецедента*** — пунктирная стрелка со стереотипом «include»;

•***расширение прецедента*** — пунктирная стрелка со стереотипом «extend» (стрелка входит в расширяемый прецедент, в дополнительном разделе которого может быть указана *точка расширения* и, возможно в виде комментария, условие расширения).

При работе с вариантами использования важно помнить несколько простых правил:

•каждый прецедент относится как минимум к одному действующему лицу;

•каждый прецедент имеет инициатора;

•каждый прецедент приводит к соответствующему результату.

Практическое занятие № 2

Моделирование простых зависимостей

Часто встречающийся вид зависимости – это связь класса, который использует другой класс в качестве параметра своей операции. Чтобы смоделировать эту связь использования, необходимо создать зависимость, направленную от класса с операцией к классу, используемому в качестве ее параметра. В примере на рис. 2 показан набор классов, составляющих систему, которая управляет назначением студентов и преподавателей на курсы в университете. Этот рисунок изображает зависимость от CourseSchedule (РасписаниеКурсов) к Course (Курс), поскольку Course используется в операциях add (добавить) и remove (удалить) класса CourseSchedule.

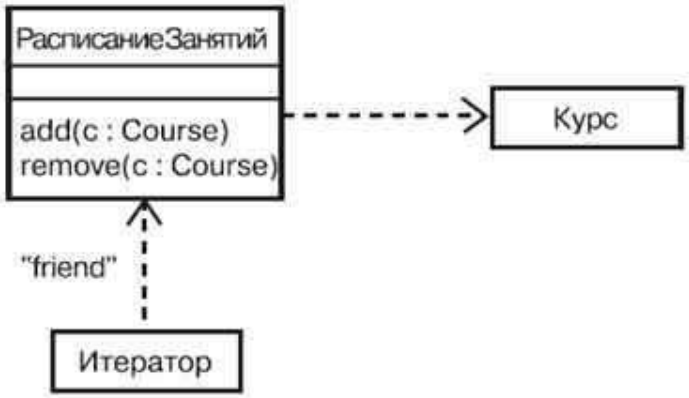


Рис. 2.1. Связи зависимости в UML

Рис. 2.1 демонстрирует зависимость, которая не связана с классами в операциях, а моделирует одну общую идиому C++. Зависимость от Iterator (Итератор) говорит о том, что Iterator использует CourseSchedule, то есть CourseSchedule ничего не знает об Iterator. Зависимость помечена стереотипом <permit> (<разрешить>), который подобен предложению friend (друг) в C++.

Моделирование одиночного наследования

В моделировании словаря системы сталкиваются с классами, которые структурно или по своему поведению похожи на другие классы. Каждый из них можно смоделировать как отдельную независимую абстракцию. Однако лучше извлечь из них общие структурные или поведенческие части и поместить их в более общие классы, от которых эти классы будут унаследованы.

Чтобы смоделировать связи наследования, необходимо:

1. Найти в наборе классов обязанности, атрибуты и операции, общие для нескольких классов.

2. Перенести эти обязанности, атрибуты и операции в более общий класс. Если необходимо, создать такой класс (но будьте осторожны – не создавайте слишком много уровней).

3. Назначить более специализированные классы потомками общих классов, проведя связь обобщения от каждого специализированного класса к его родителю.

На рис. 2.2 представлен ряд классов, формирующих приложение управления торговлей. Здесь имеется связь обобщения, проведенную от четырех классов – CashAccount (РасчетныйСчет), Stock (Акция), Bond (Облигация) и Property (Собственность), – к более общему классу Security (ЦенныеБумаги). Класс Security является родителем, а CashAccount, Stock, Bond и Property – потомками (дочерними классами). Каждый из этих четырех специализированных классов представляет собой некоторую разновидность Security. Последний включает две операции: presentValue (текущаяСтоимость) и history (История). Поскольку Security является родителем CashAccount, Stock, Bond и Property, все они наследуют эти две операции, равно как и любые другие атрибуты и операции Security, которые могут быть не указаны на этом рисунке.

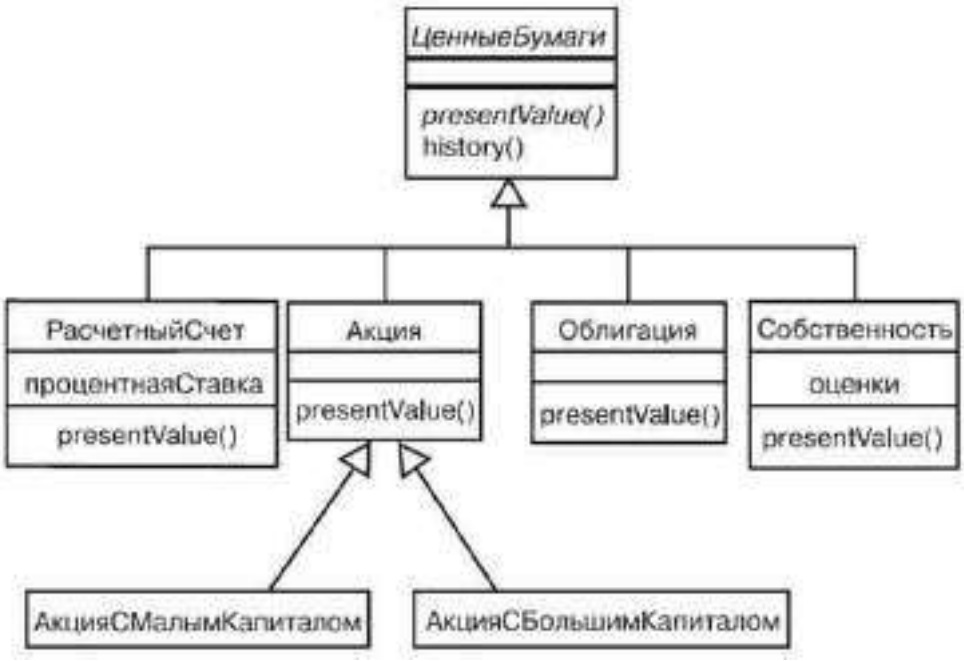


Рис. 2.2. Связи наследования в UML

Практическое занятие № 3

Информационная система (ИС) создаётся для решения задач некоторой организации (завода, банка, вуза, библиотеки и т.д.). Для создания и эксплуатации ИС требуется её описание. Полное, исчерпывающее, описание ИС должно включать в себя не только саму ИС, но и окружающую среду, то есть должно быть описанием предметной области.

Подробное описание предметной области можно дать в общем случае только в свободной форме. Для графического описания абстрактной модели проектируемой системы используется UML (Unified Modeling Language - унифицированый язык моделирования). Мы не будем изучать этот язык из-за его чрезмерной абстрактности и сложности.

Существенной, если не главной частью ИС являются хранящиеся в ней данные. При проектировании ИС данные нужно представить в виде простой модели, отображающей смысл данных, их взаимосвязь и не привязываться при этом к конкретному типу базы данных. Такие модели получили название инфологических.

Инфологическую модель можно построить, опираясь только на интуитивное представление о данных.

На рис. 3.1. приведён простой пример инфологической модели.

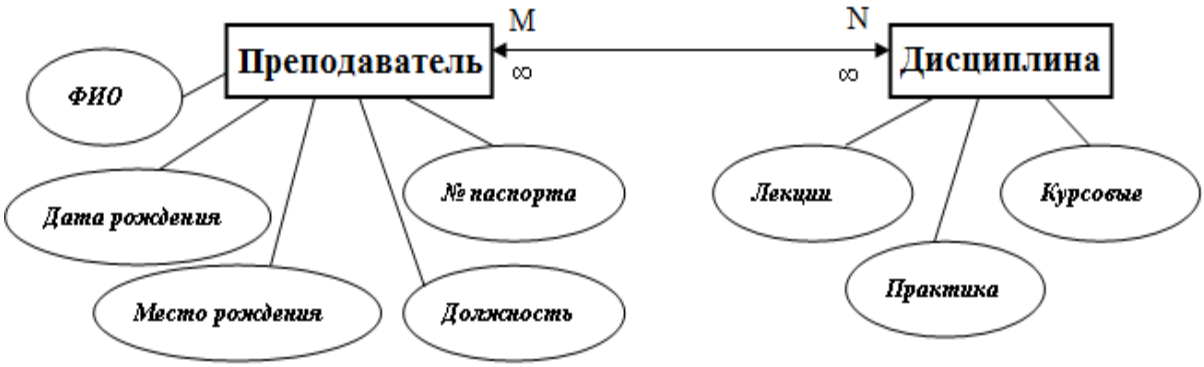


Рис. 3.1. Инфологическая модель преподаватель-дисциплина

В 1976 году Чен предложил термин *сущность*, а состоящая из связанных между собой сущностей модель получила название модель сущность-связь (entity-relation).

Сущность - это то, что может быть чётко идентифицировано, за чем хотелось бы или необходимо наблюдать в рамках поставленной задачи. Буквально, сущность - это то что существует.

Сущность состоит из множества экземпляров, обладающих одинаковым набором свойств.

Совокупность свойств, необходимая для отличия одного экземпляра от других, называется идентификатором сущности.

Практическое занятие № 4

Разработать гейм-дизайн. В разработке по каскадной или спиральной модели обучающийся или группа создают дизайн-документ, в котором подробно описываются многие игровые функции и механика. Затем дизайн-документ разбивается на части, которые другая группа использует для извлечения необходимых функциональностей и ресурсов. Требования, предъявляемые к ресурсам и элементам, касаются всех команд, вовлечённых в разработку.

Классическая каскадная модель улучшает оценочный анализ других, зачастую более эффективных моделей жизненного цикла, основанных на данной модели. Это напоминает процесс, изображенный на рис. 4.1.

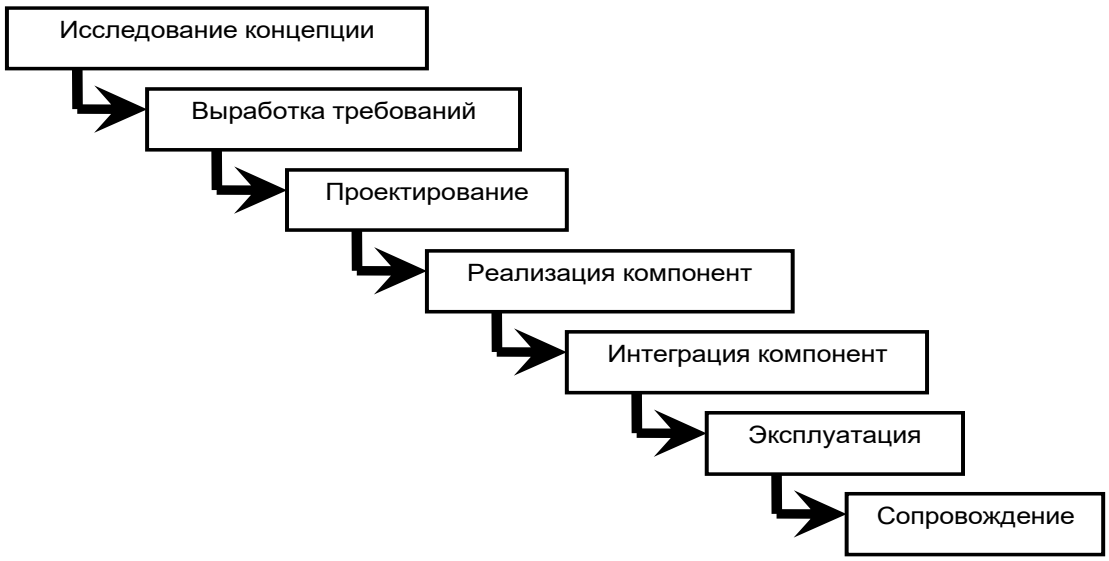


Рис. 4.1. Каскадная или водопадная модель

Основными принципами каскадной модели являются:

1. Строго последовательное выполнение фаз.
2. Каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы.
3. Каждая фаза имеет определенные критерии входа и выхода: входные и выходные данные.
4. Каждая фаза полностью документируется.
5. Переход от одной фазы к другой осуществляется посредством формального обзора с участием заказчика.
6. Основа модели – сформулированные требования – техническое задание (ТЗ), которые меняться не должны.
7. Критерий качества результата – соответствие продукта установленным требованиям.

Практическое занятие № 5

Зачастую на стадии формулировки требований заказчик не может точно определить все требования к программному продукту. Для преодоления данной проблемы был предложен «спиральный» процесс создания программ (рис. 5.1). Разработка системы по данной методологии происходит итерациями, и после прохождения каждого витка спирали пользователь получает очередную версию системы. После получения заказчиком каждой версии уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали.

Работа выполняется согласно алгоритму:

1. В процессе общения с заказчиком формируется общее видение проекта, а также описываются функциональные возможности, которые необходимо реализовать в определенные сроки с нужным качеством. Прототип 1, Прототип 2, Прототип 3. Определение требований, Анализ, Интеграция, Реализация и тестирование, Проектирование

2. Расставляются приоритеты, задающие порядок реализации основных функциональных возможностей.

3. Согласовываются временные рамки проекта. Часто для этого применяются методики стоимостного прогнозирования. Исполнитель решает, сколько функциональных возможностей в

соответствии с их приоритетами удастся реализовать в оговоренный срок.

4. На данном этапе определяются архитектура и ядро будущей системы. Это наиболее ответственный момент, так как здесь необходимо учесть пока еще не детализированные полностью требования к проекту – а они могут быть противоречивыми. Ядро должно представлять собой законченный работающий вариант системы с небольшим набором необходимых возможностей. Далее определяется способ реализации требований с более низкими приоритетами – это можно делать, например, с помощью встроенного языка сценариев или подключением динамических библиотек, для чего необходимо определить внутренние интерфейсы ядра. Этот шаг выполняется, как правило, в два и большее число итерационных циклов.

5. Готовится план работ. Он ориентирован на сроки, определенные на третьем этапе, и нацелен на скорейшую реализацию ядра системы. Взаимодействуя с работающим прототипом, заказчик быстрее и точнее формулирует дальнейшие требования.

6. Разработка системы в соответствии с планом. Для этого этапа характерны три типичных класса проблем:

– изменения в требованиях к проекту;

– изменения параметров самого проекта (сроков, бюджета, качества);

– временные задержки, связанные с текущими вопросами (техникой, персоналом).

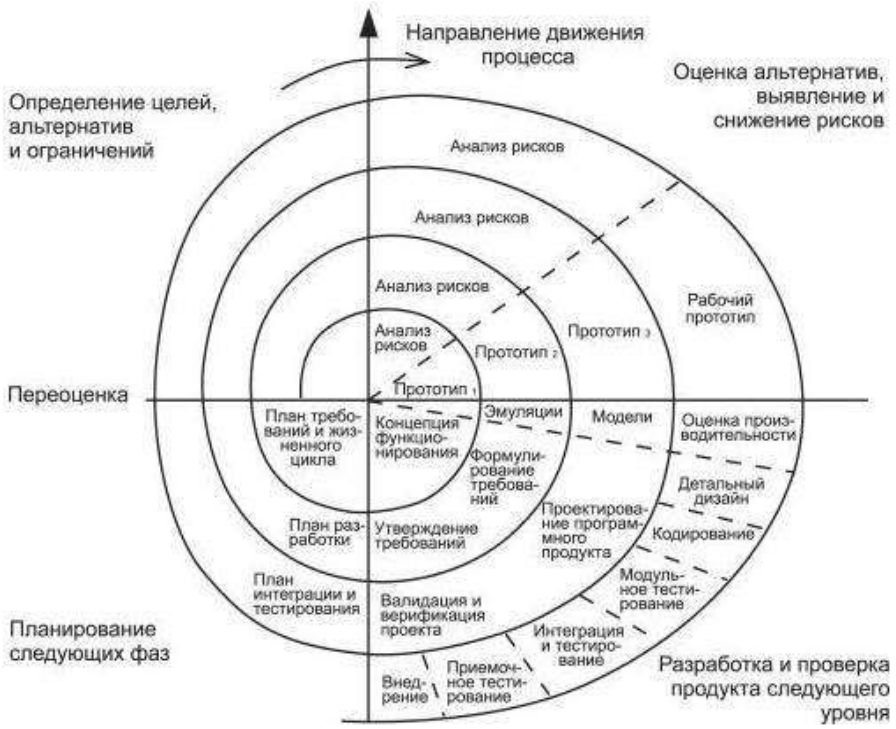


Рис. 5.1. Спиральная модель

Практическое занятие № 6

Построение команды в MSF соответствует ряду ключевых концепций. К очевидным можно отнести следующие принципы. Концентрация на нуждах заказчика. Это главный приоритет любой приличной проектной группы. Он подразумевает обязательное понимание разработчиками бизнес-задач заказчика, активное участие заказчика в проектировании решения и получение его отзывов в ходе процесса разработки. Нацеленность на конечный результат. Получению конечного результата проекта уделяется больше внимания, чем процессу его достижения. Каждый участник проектной группы должен рассматривать собственную работу как самостоятельный проект или вклад в "общее дело". Установка на отсутствие дефектов. Это стремление к высочайшему уровню качества. В успешной команде ответственность за качество продукта не может быть делегирована, каждый сотрудник чувствует свою ответственность. Концепции, считающиеся "ноу-хау" методологии MSF: "Проектная группа – команда равных". Концепция означает равноправное положение каждой из ролей в команде. Каждый из членов команды, независимо от роли, должен нести ответственность за качество продукта, понимать инте-

ресы заказчика и сущность решаемой бизнес-задачи. Решения принимаются методом консенсуса между ролями, но не методом консенсуса между сотрудниками (каждая ролевая группа требует определенной организационной иерархии). Это неустанное саморазвитие посредством накопления опыта и обмена знаниями. По завершению проекта и по окончанию основных его фаз предполагается проведение открытых обсуждений и объективный анализ. К концепции команды равных в MSF тесно примыкает идея о том, что каждая ролевая группа имеет зону ответственности и защищает интересы заинтересованных лиц из этой зоны. Модель проектной группы MSF выделяет 7 ролевых групп и 6 ролей (табл. 6.1).

Таблица 6.1. Модель проектной группы

Ролевая группа	Зона ответственности
Управление программой	управление проектом; верная трактовка ожидания заинтересованных сторон и их проведение через проект
Архитектура продукта	система в целом (вырабатывает архитектуру решения, включая сервисы, технологии и стандарты, которые будут использованы в ходе работы)
Разработка	проектирование и осуществление реализации
Тестирование	качество решения с точки зрения заказчика и будущих пользователей
Управление выпуском	гладкое внедрение решения в инфраструктуру заказчика
Удовлетворение потребителя	понимание потребностей пользователей и их надлежащую реализацию в решении
Управление продуктом	представляет бизнес-сторону проекта и обеспечивает его согласованность со стратегическими целями заказчика (успешное получение бизнес-отдачи от внедрения разрабатываемого решения

Практическое занятие № 7

Условия формирования команды (рис. 7.1):

1. Люди, выполняющие работу, должны быть специалистами в своей области.

2. Совокупный опыт и таланты людей, работающих в коллективе, должны превышать опыт и способности любого из тех, кто работает в одиночку.
3. Большинство людей должно иметь возможность в какой-то мере влиять на принятие тех решений, которые им приходится выполнять, что повышает их заинтересованность.
4. Каждый человек должен иметь склонности к творчеству.

Рис. 7.1 Формирование команды на базе административной модели

Практическое занятие № 8

Теория "хаоса" - методологическая основа формирования адаптивных систем управления.

В современных организациях задача менеджеров состоит в том, чтобы понять систематические процессы, управляющие человеческим поведением, и использовать их. Искусство понимания системы заключается в умении доходить до причин, лежащих в основе изменений. Когда же менеджеры понимают динамику этих прототипов, они в состоянии действительно осуществить какие-то изменения. В соответствии с теорией хаоса небольшие изменения могут оказывать значительное влияние на физические системы. Так, решающей концепцией в теории систем является система "рычагов", т.е. идея о том, что небольшие, хорошо про-

думанные действия иногда могут вызвать значительные долгожданные улучшения.

Осмысление и использование систем позволит менеджерам создать "познающую, самообучающуюся организацию". Такая организация имеет характеристики, аналогичные сложным адаптивным системам, которые обнаруживаются в природе. Это высокодецентрализованная система, в которой при любом числе процессов принятия решений на местном (локальном) уровне сохраняется порядок во всей системе. Она постоянно адаптируется к изменениям.

Вместе с тем в организациях с участием людей предполагается наличие органичного контроля, который встречается и в природе. Подобный контроль заложен в моделях макромиров, созданных компьютерами для сложных деловых ситуаций. С ними руководители могут проводить эксперименты в своих организациях, чтобы обнаруживать скрытую динамику сложных систем.

Практическое занятие № 9

В ИТ-области интеграционный подход выражается в необходимости создания контура взаимосвязанных продуктов, в котором СУП связывается с другими системами предприятия информационными и пользовательскими интерфейсами.

В организационной области интеграционный подход выражается в необходимости формирования управленческих структур, лежащих над штатным расписанием (руководящий комитет, группа управления, рабочая группа), и организационно-распорядительных документов, описывающих сквозные процессы, затрагивающие не только персонал проекта, но и постоянные структурные подразделения предприятия (ресурсные подразделения, финансовая служба, служба логистики, служба безопасности и т. д.) (рис. 9.1).

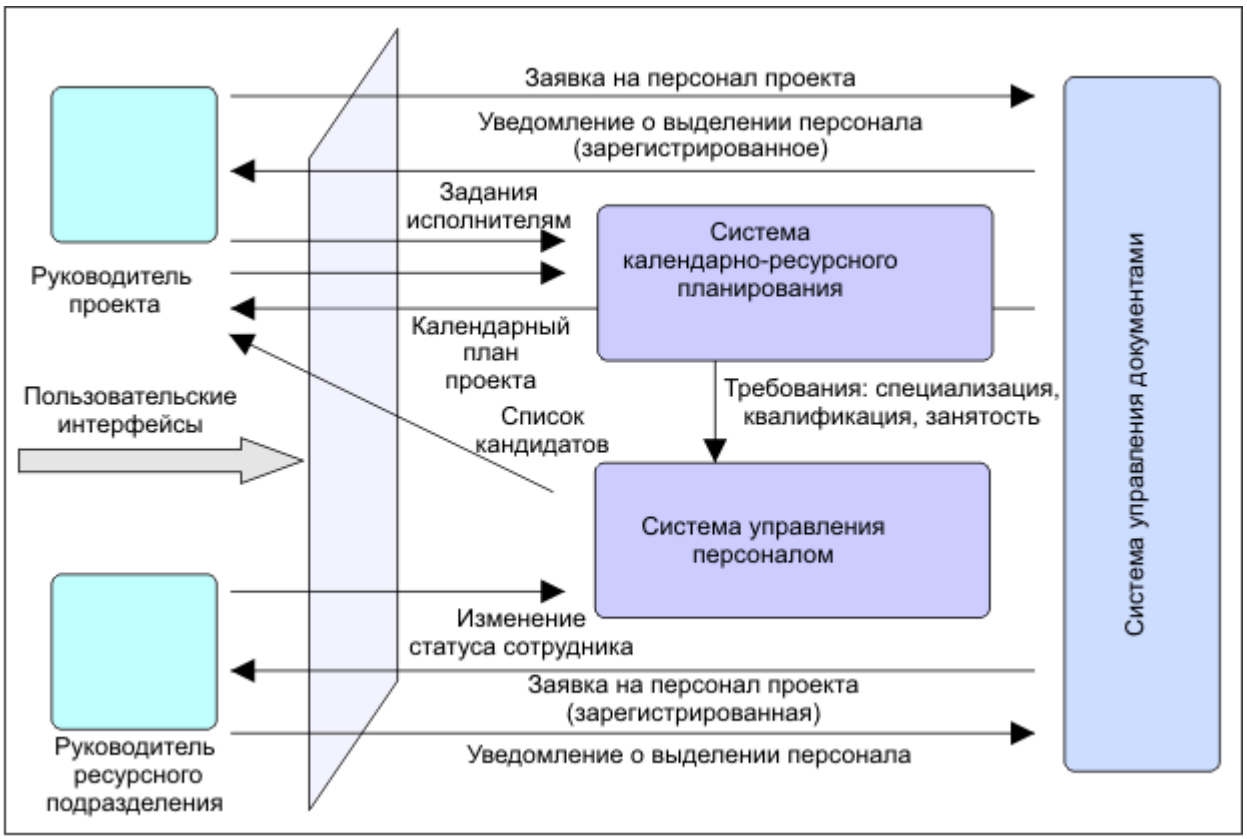


Рис. 9.1. Процесс формирования команды проекта

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Этапы разработки программного обеспечения при системном подходе. Стадия «Техническое задание». Моделирование и анализ процессов предметной области с целью выделения процесса для автоматизации. Формирование функциональных и качественных требований к проектируемому программному средству. Формирование технического задания на разработку программного средства.	4	Защита отчета	15-20.02

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
2	Этапы разработки программного обеспечения при системном подходе. Стадия «Эскизный проект» Создание контекстной диаграммы на основе нотации DFD. Формирование архитектуры программного средства и структуры данных. Разработка моделей данных на основе DFD-диаграммы.	4	Защита отчета (работа в малой группе)	15-20.03
3	Этапы разработки Программного обеспечения при системном подходе. Стадия «Технический проект». Формирование модульной структуры программы. Разработка алгоритмов модулей. Разработка интерфейса программы. Формирование функционально-технологических схем обработки данных для разрабатываемого ПС.	4	Защита отчета (работа в малой группе)	15-20.04
4	Этапы разработки Программного обеспечения при системном подходе. Стадия «Реализация». Формирование кода программы на языке программирования для разработанного проекта программного средства.	4	Защита отчета (работа в малой группе)	15-20.05
5	Тестирование программы на основе «черного ящика» и приемка программного средства. Разработка тестовых наборов и выполнение тестирования разработанного программного средства. Защита созданного программного средства.	2	Защита отчета (работа в малой группе)	15-20.05

Примерные варианты заданий к лабораторным работам

1) Информационная система «Мотель». Ведение справочников: Номера, Услуги, Клиенты, Стоянка.

Функции: Ведение справочников, заселение и выселение клиентов, бронирование мест, учёт оказанных услуг.

Выходные документы: Счёт за проживание и услуги, Список проживавших на момент времени, Список номеров, Прейскурант услуг.

2) Информационная система «Деканат». Ведение справочников: Институты, факультеты, кафедры, студенты, дисциплины.

Функции: ведение справочников, учёт успеваемости студентов.

Выходные документы: Ведомость успеваемости по группе студентов, Приложение к диплому, Аналитические отчёты.

3) Информационная система «Лечебное заведение». Ведение справочников: Пациенты, Болезни, Палаты, Врачи, История болезни.

Функции: Ведение справочников, приём пациента, ведение истории болезни, выписка.

Выходные документы: Список пациентов, Список врачей, Карточка больного.

4) Информационная система «Аптека». Ведение справочников: Группы лекарств, Лекарства, Производители, Поставщики.

Функции: ведение справочников, учёт прихода и продаж лекарств

Выходные документы: Отчёт по наличию лекарств на складе по группам, Отчёт по продажам по группам, Счёт-фактура.

5) Информационная система «Ресторан». Ведение справочников: Продукты, Блюда, Заказы.

Функции: Ведение справочников, хранение рецептов, расчёт себестоимости блюда, приём заказов.

Выходные документы: Меню, Счёт заказа, Отчёт по продуктам на складе, Заказы за период.

Лабораторная работа №1

Цель работы: Моделирование и анализ процессов предметной области с целью выделения процесса для автоматизации. Формирование функциональных и качественных требований к проектируемому программному средству. Формирование технического задания на разработку программного средства.

Программа работы

Выполнить анализ процессов предметной области с целью выделения процесса для автоматизации. Работа включает в себя подготовительный этап, в ходе выполнения которого выполняется анализ процессов предметной области (по выбору обучающегося). Выполняется формирование функциональных и качественных требований к проектируемому программному средству. В завершение формируется техническое задание на разработку программного средства.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Условие тестовой задачи.
3. Подготовительный этап, на котором выполняется анализ процессов предметной области.
4. Формирование функциональных и качественных требований к проектируемому программному средству.
5. Формирование технического задания на разработку программного средства.
6. Результаты выполнения п. 3-5.
7. Вывод о проделанной работе.

Лабораторная работа №2

Цель работы: Практическое освоение методов и создания контекстной диаграммы на основе нотации DFD. Формирование архитектуры программного средства и структуры данных. Разработка моделей данных на основе DFD-диаграммы.

Программа работы

На основе анализа потоков данных создать контекстную диаграмму и разработать соответствующие модели. Сформиро-

вать архитектуру программного средства и структуру данных.
Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Контекстную диаграмму.
3. Описание соответствующих моделей.
4. Архитектуру программного средства и структуру данных.
5. Вывод о проделанной работе.

Лабораторная работа №3

Цель работы: Формирование модульной структуры программы. Разработка алгоритмов модулей. Разработка интерфейса программы. Формирование функционально-технологических схем обработки данных для разрабатываемого ПС.

Программа работы

Представить модульную структуру программы. Разработать алгоритмы модулей. Разработать интерфейс программы. Сформировать функционально-технологическую схему обработки данных для разрабатываемого ПС.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Модульную структуру программы.
3. Описание алгоритмов модулей.
4. Интерфейс программы.
5. Функционально-технологическую схему обработки данных для разрабатываемого ПС.
6. Вывод о проделанной работе.

Лабораторная работа №4

Цель работы: Формирование кода программы на языке программирования для разработанного проекта программного средства.

Программа работы

Написать программу, реализующую алгоритмы модулей разрабатываемого программного средства.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Листинг с необходимыми комментариями.
3. Интерфейс программы.
4. Вывод о проделанной работе.

Лабораторная работа №5

Цель работы: Разработка тестовых наборов и выполнение тестирования разработанного программного средства. Защита созданного программного средства.

Программа работы

Разработать тестовые наборы и выполнить тестирование разработанного программного средства. Привести методы защиты созданного программного средства. Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Комплекты тестовых наборов.
3. Средства защиты программного продукта.
4. Вывод о проделанной работе.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 125,1 ч.

№	Наименование тем	Кол-во часов	Литерату-ра
1	**Тема 1.** Обзор методологий проектирования программных продуктов	12	1-7
2	**Тема 2.** Программный процесс, модель программного процесса. Методы программной инженерии: модель прецедентов, модель классов, модель сущность-связь, нотации модели.	14	1-7
3	**Тема 3.** CASE средства, их разновидности и классификация. Эволюция CASE-средств. Примеры реализации CASE-средств.	16	1-7
4	**Тема 4.** Критерии качества программных продуктов. Свойства хорошей программы. Основные трудности программной инженерии. Перспективы развития программной инженерии.	14	1-7
5	**Тема 5.** Технологии, стандартизация и сертификация. Основные разработчики стандартов: SE ISO, ACM, SEI, PMI, IEEE. Характеристики основных стандартов программной инженерии: ISO/IEC 12207 - Information Technology - Software Life Cycle Processes, SEI CMM - Capability Maturity Model (for Software), ISO/IEC 15504 - Software Process Assessment, PMBOK - Project Management Body of Knowledge, SWEBOK - Software Engineering Body of Knowledge, ACM/IEEE CC2001 - Computing Curricula 2001.	13,1	1-7
6	**Тема 6.** Стандарты ЖЦ ПО ISO 12207 (15504). Жизненный цикл ПП: структура и организация. Модели жизненного цикла программного продукта. Каскадная модель. Спиральная модель. Итерационная модель. V-образная модель. Инкрементная (пошаговая) модель. Модель быстрого прототипирования.	14	1-7
7	**Тема 7.** Сущность и особенности моделей: Microsoft Solution Framework, Rational Unified Process и Extreme Programming. Adaptive	14	1-7

№	Наименование тем	Кол-во часов	Литерату-ра
	Software Development (ASD). Семейство методологий Crystal. Методология SCRUM. Методология Feature Driven Development (FDD).		
8	**Тема 8.** Управление и проект. Основные характерные черты проекта. Примеры не проектов и их связь с проектами. Категории управления проектами. Треугольник ограничений проекта. PMBOK: 9 областей управленческих знаний. SQI: 34 компетенции IT-менеджера. Ролевая модель команды.	14	1-7
9	**Тема 9.** Модель управления командой, критерии выбора модели. Административная модель управления командой, модель хаоса, модель открытой архитектуры. Корпоративная политика.	14	1-7

Задания для самостоятельного изучения и конспектирования

Тема 1. Рассмотреть следующие вопросы:

- методологии проектирования программных продуктов.

Тема 2. Рассмотреть следующие вопросы:

- программный процесс;
- модель программного процесса;
- методы программной инженерии: модель прецедентов, модель классов, модель сущность-связь, нотации модели.

Тема 3. Рассмотреть следующие вопросы:

- CASE средства, их разновидности и классификация;
- эволюция CASE-средств;
- примеры реализации CASE-средств;

Тема 4. Рассмотреть следующие вопросы:

- критерии качества программных продуктов;
- свойства хорошей программы;
- основные трудности программной инженерии;
- перспективы развития программной инженерии.

Тема 5. Рассмотреть следующие вопросы:

- технологии, стандартизация и сертификация;
- основные разработчики стандартов: SE ISO, ACM, SEI, PMI, IEEE.
- характеристики основных стандартов программной инженерии: ISO/IEC 12207 - Information Technology - Software Life Cycle Processes, SEI CMM - Capability Maturity Model (for Software), ISO/IEC 15504 - Software Process Assessment, PMBOK - Project Management Body of Knowledge, SWEBOK - Software Engineering Body of Knowledge, ACM/IEEE CC2001 - Computing Curricula 2001.

Тема 6. Рассмотреть следующие вопросы:

- стандарты ЖЦ ПО ISO 12207 (15504);
- жизненный цикл ПП: структура и организация;
- модели жизненного цикла программного продукта;
- каскадная модель;
- спиральная модель;
- итерационная модель;
- v-образная модель;
- инкрементная (пошаговая) модель;
- модель быстрого прототипирования.

Тема 7. Рассмотреть следующие вопросы:

- сущность и особенности моделей: Microsoft Solution Framework, Rational Unified Process и Extreme Programming. Adaptive Software De-velopment (ASD). Семейство методологий Crystal. Методология SCRUM. Методология Feature Driven Development (FDD).

Тема 8. Рассмотреть следующие вопросы:

- управление и проект;
- основные характерные черты проекта;
- примеры не проектов и их связь с проектами;
- категории управления проектами;
- треугольник ограничений проекта;
- PMBOK: 9 областей управленческих знаний;
- SQI: 34 компетенции IT-менеджера;
- ролевая модель команды.

Тема 9. Рассмотреть следующие вопросы:

– модель управления командой, критерии выбора модели;
– административная модель управления командой, модель хаоса, модель открытой архитектуры;
– корпоративная политика.

Вопросы к зачету по курсу

1. Предпосылки возникновения программной инженерии. Проблемы разработки программ и фундаментальные методы их решения.
2. Что такое программное обеспечение (software)? Какие бывают типы ПО?
3. Что такое программная инженерия? В чем разница между программной инженерией (software engineering) и информатикой (computer science)?
4. Что такое программная инженерия? В чем отличие программной инженерии от других инженерий?
5. Что такое программный процесс? Какие существуют виды процессов? Что предполагает установление процесса?
6. Что такое модель программного процесса? Основные модели жизненного цикла?
7. Что такое методы программной инженерии?
8. Что такое CASE (Computer-Aided Software Engineering)? Классификация CASE средств.
9. Свойства хорошей программы.
10. Какие основные трудности стоят перед программной инженерией?
11. Объяснить, каким образом связаны следующие понятия: технология, стандарт и сертификация.
12. Что такое технология? Что такое стандарт? Что такое сертификация?
13. Что такое стандарт, какие бывают виды стандарты?
14. Какие организации являются основными разработчиками стандартов в области информационных технологий?
15. Дать краткую характеристику стандарта SEI CMM. Перечислить пять уровней зрелости процесса.

16. Дать краткую характеристику стандарта PMBOK. Перечислить 9 областей знаний управления проектами.
17. Дать краткую характеристику стандарта SWEBOK. Перечислить 10 областей знаний программной инженерии.
18. Что такое жизненный цикл программного продукта? Что такое фаза проекта, процесс, операция? Типы процессов, их назначение и классификация.
19. Что такое модель жизненного цикла ПО? Каскадная модель. Преимущества, недостатки, область применимости.
20. Что такое модель жизненного цикла ПО? Спиральная модель. Преимущества, недостатки, область применимости.
21. Что такое модель жизненного цикла ПО? Итерационная модель. Преимущества, недостатки, область применимости.
22. Что такое модель жизненного цикла ПО? V-образная модель. Преимущества, недостатки, область применимости.
23. Что такое модель жизненного цикла ПО? Инкрементная (пошаговая) модель. Преимущества, недостатки, область применимости.
24. Что такое модель жизненного цикла ПО? Модель быстрого прототипирования. Преимущества, недостатки, область применимости.
25. Промышленные модели жизненного цикла. Модель Microsoft Solution Framework (MSF).
26. Промышленные модели жизненного цикла. Модель Rational Unified Process (RUP).
27. Промышленные модели жизненного цикла. Модель Extreme Programming (XP).
28. Основная идея методологии управления SCRUM для гибкой разработки программного обеспечения.
29. Что такое доска задач и burndown-диаграмма в методологии SCRUM? Их назначение и способ использования.
30. Методология ASD, основные принципы.
31. Принципы классификации проектов в семействе методологий Crystal.
32. Особенности методологий Crystal Clear и Crystal Orange.
33. Методология FDD, основные принципы.
34. Что такое управление, обобщенная схема управления?

35. Понятие проекта. Семь основных характеристик проекта. Примеры непроектов и их связь с проектами.
36. Понятие проекта. Категории управления проектами.
37. Понятие проекта. Треугольник ограничений проекта.
38. Девяти областей управленческих знаний и их краткая характеристика.
39. На какие три категории разбиты 34 компетенции менеджера IT проекта. Описать категорию «Методика разработки продукта».
40. На какие три категории разбиты 34 компетенции менеджера IT проекта. Описать категорию «Навыки управления проектов».
41. На какие три категории разбиты 34 компетенции менеджера IT проекта. Описать категорию «Навыки управления персоналом».
42. Ролевая модель команды, ее смысл и назначение. Описать роли менеджера проекта, проектировщика и разработчика.
43. Ролевая модель команды, ее смысл и назначение. Описать роли тестировщика и инженера по качеству.
44. Ролевая модель команды, ее смысл и назначение. Описать роли технического писателя и технолога разработки ПО.
45. Ролевая модель команды при организации взаимодействия с заказчиком.
46. Модель управления командой и критерии выбора модели. Административная модель, преимущества и недостатки.
47. Модель управления командой и критерии выбора модели. Модель хаоса, преимущества и недостатки.
48. Модель управления командой и критерии выбора модели. Модель открытой архитектуры, преимущества и недостатки.
49. Роль общения в команде. Возможные способы общения в команде. Преимущества и недостатки различных способов общения.
50. Способы принятия решений. Компромисс и консенсус, отличие компромисса и консенсуса. Способы достижения компромисса и консенсуса.

Контактная внеаудиторная работа

СРС – групповые консультации с преподавателем в течение семестра – 0,9 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Информатика. Программная документация: учеб. пособие / Ренсков А. А., Родионов А. С., Чижов А. Ю., Янченко И. П.; Гос. образовательное учреждение высшего профессионального образования "НВВКУС (Военный ин-т)"; - Новочеркасск: Изд-во НВВКУС, 2010. 253 с.
2. Кудинов Ю.И. Основы современной информатики [Электронный ресурс] : учебное пособие / Ю.И. Кудинов, Ф.Ф. Пащенко. – Электрон. Дан. – СПБ. : Лань, 2011. – 256 с. Режим доступа: http:e//lanbook.com/book/element.php?pl1_id=68468-Загл. с экрана.
3. Кудинов Ю.И. Практикум по основам современной информатики [Электронный ресурс] : учеб. пособие / Ю.И. Кудинов, Ф.Ф. Пащенко, А.Ю. Келина. – Электрон. Дан. – СПБ.: Лань, 2011. – 351 с. Режим доступа: http:e//lanbook.com/book/element.php?pl1_id= 68471-Загл. с экрана.

Дополнительная учебная литература

4. Крылов Е.В. Техника разработки программ: учебник для вузов : в 2 кн.: Кн. 2. / Е.В. Крылов, В.А. Острейковский, Н.Г. Типикин. – М.: Высш. шк. 2008. – 469 с.
5. Иванченко А. Н. Основы программирования: учеб. пособие / А. Н. Иванченко, П. А. Иванченко, А. А. Масленников ; Юж-Рос. гос. политехн. ун-т (НПИ) им. М.И. Платова. – Новочеркасск : ЮРГПУ (НПИ), 2014. - 108 с. – Режим доступа: http://lib/npi-tu.ru
6. Гринченков Д. В. Основы программирования в среде Windows на WinAPI: учеб. пособие / Д. В. Гринченков, П. В. Шлыков ; Юж-Рос. гос. политехн. ун-т (НПИ) им. М.И. Платова. – Новочеркасск : ЮРГПУ (НПИ), 2014. - 106 с. – Режим доступа: http://lib/npi-tu.ru

Методические указания к лабораторным и практическим занятиям

7. Иванченко А. Н. Функциональное и логическое программирование: учеб. пособие для вузов / А. Н. Иванченко, Д. В. Гринченков, С. И. Потоцкий; ЮРГТУ(НПИ). - 2-е изд., испр.. - Новочеркасск: Изд-во ЮРГТУ(НПИ), 2006. - 240 с.

Учебно-методическое издание

Дьяченко Владимир Борисович
Мохов Василий Александрович

Основы разработки и реализации программных проектов

Методические указания
к практическим занятиям, лабораторным работам
и самостоятельной работе студентов

Редактор *Н.А.Юшко*
Подписано в печать 12.10.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,68. Уч.-изд.л. 1,75 . Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

Основы управления данными

Методические указания к курсовой работе, практическим занятиям, лабораторным работам и самостоятельной работе студентов

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 004.451.5 (076.5)
ББК 32.973-018.2я73
Ш69

Рецензент

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Панфилов Андрей Николаевич

Шлыков П.В. Гасанов Р.А.

Основы управления данными: методические указания к курсовой работе, практическим занятиям, лабораторным работам и самостоятельной работе студентов / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2017. – 20с.

В данном издании представлены методические указания к курсовой работе, практическим занятиям, лабораторным работам и самостоятельной работе по курсу «Основы управления данными».

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 004.451.5 (076.5)
ББК 32.973-018.2я73

Содержание

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ 1
1.1. Разработка концептуальной модели данных 1
1.2. Разработка логической модели данных 1
1.3. Разработка физической модели данных 2
1.4. Изучение основных выражений языка SQL 2
2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ 3
Варианты заданий для лабораторных работ 3
Лабораторная работа №1 4
Лабораторная работа №2 5
Лабораторная работа №3 5
Лабораторная работа №4 6
Лабораторная работа №5 6
3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС) 7
4. КУРСОВАЯ РАБОТА 10
4.1. Указания к выполнению курсовой работы 10
4.2. Примерные задания на курсовую работу 11
5. КОНТАКТНАЯ ВНЕАУДИТОРНАЯ РАБОТА 11
5.1. Вопросы для оценивания знаний 12
5.2. Вопросы для оценивания умений и навыков 14
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 19

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Номер компетенции	Лит-ра
1	Разработка концептуальной модели данных.	4	Тест	10-15.10	ОПК-4, ПК-2	7 [1-5]
2	Разработка логической модели данных.	4	Тест	10-15.10	ОПК-4, ПК-2	7 [1-5]
3	Разработка физической модели данных.	4	Тест	15-20.11	ОПК-4, ПК-2	7 [1-5]
4	Изучение основных выражений из подмножества DDL, DML, PL языка SQL.	6	Тест	15-20.12	ОПК-4, ПК-2	7 [1-5]

Исходные данные к практическим занятиям:

Информационная система учета успеваемости студентов. Система должна обрабатывать информацию о дисциплинах, зачетах, экзаменах и других формах отчетности по дисциплинам, информацию о преподавателях, которые выставляют оценки, информацию о студентах и студенческих группах.

1.1. Разработка концептуальной модели данных

Разработать концептуальную модель информационной системы учета успеваемости студентов. Для разрабатываемой модели «сущность-связь» необходимо использовать нотацию Чена. В разрабатываемой модели должно быть не более 10 сущностей. Обязательно указать ключевые атрибуты сущностей, типы связей и их кардинальность.

1.2. Разработка логической модели данных

На основе концептуальной модели созданной на практическом занятии №1, разработать логическую модель информационной системы учета успеваемости студентов. Для разрабатываемой модели «сущность-связь» необходимо использовать нотацию Мартина или близкую к ней. Обязательно указать первичные и внешние ключи.

1.3. Разработка физической модели данных

На основе логической модели созданной на практическом занятии №2, разработать физическую модель информационной системы учета успеваемости студентов. Для разрабатываемой модели «сущность-связь» необходимо использовать нотацию Мартина или близкую к ней. Обязательно указать первичные и внешние ключи, типы данных атрибутов. В качестве СУБД выбрать Firebird.

1.4. Изучение основных выражений языка SQL

Для физической модели информационной системы учета успеваемости студентов, созданной на практическом занятии №3, создать:

Изучение основных выражений из подмножества DDL, DML, PL языка SQL.

1. Использую подмножество DDL языка SQL, написать скрипт создания БД;
2. Использую подмножество DML языка SQL, написать операторы по работе с информацией, хранящейся в БД. Для написания использовать операторы INSERT, UPDATE, DELETE и SELECT;
3. Использую подмножество PL языка SQL, написать триггер для обеспечения ссылочной целостности между двумя связанными таблицами в БД;

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Номер компетенции	Лит-ра
1	Установка СУБД FireBird и утилиты IB Expert. Изучение средств администрирования БД с консольным и графическим интерфейсом.	4	Защита отчета	10-15.10	ОПК-4, ПК-2	7 [3,5]
2	Создание таблиц, полей, связей между таблицами, проверок средствами SQL. Разработка скрипта для генерации БД .	8	Защита отчета	15-20.10	ОПК-4, ПК-2	7 [3,5]
3	Использование операторов языка SQL: SELECT, INSERT, UPDATE, DELETE. Применение механизма транзакций.	8	Защита отчета	15-20.11	ОПК-4, ПК-2	7 [3,5]
4	Использование операторов процедурного подмножества языка SQL, для создания триггеров и хранимых процедур.	8	Защита отчета	15-20.12	ОПК-4, ПК-2	7 [3,5]
5	Разработка простого приложения - клиента для взаимодействия с сервером БД.	8	Защита отчета	15-20.12	ОПК-4, ПК-2	7 [3,5]

Варианты заданий для лабораторных работ

В ходе всех лабораторных работ студент работает над выполнением одного задания из перечисленных ниже вариантов. Цели и задачи разработки уточняются студентом самостоятельно. Концептуальная модель данных должна содержать не менее 4 и не более 8 сущностей. Приложение-клиент для взаимодействия с сервером и база данных должны быть разработаны с учетом возможности одновременной работы нескольких пользователей.

Варианты заданий:

1. Информационная система учета успеваемости студентов. Система может разрабатываться либо для использования в деканате, либо для использования студентами вуза. Система должна обрабатывать информацию о дисциплинах, зачетах, экзаменах и других формах отчетности по дисциплинам, информацию о преподавателях, которые выставляют оценки, информацию о студентах и студенческих группах.

2. Информационная система учета домашних финансов. Система разрабатывается для применения в условиях домашнего хозяйства. Система должна обеспечивать учет поступлений от всех членов семьи, учет расходов на совершение различных покупок, подготовку планов покупок.

3. Информационная система «Органайзер». Система разрабатывается для использования группой лиц. Каждый пользователь имеет возможность составлять списки значимых событий, списки задач, планы по решению задач (в форме списка действий), расписание на день, отмечать выполненные и невыполненные задачи.

4. Информационная система учета удобрения почвы и полива комнатных растений. Система разрабатывается для использования группой лиц. Система должна обеспечивать обработку сведений о схемах внесения удобрений и полива в грунт для различных видов комнатных растений, учет выполнения операций по поливу и внесению удобрений, напоминание о необходимости выполнения соответствующих операций по выбранной схеме.

Лабораторная работа №1

Цель работы: освоение на практике процесса установки СУБД Firebird; освоение на практике утилит предназначенных для администрирования баз данных с консольным интерфейсом; освоение на практике основ использования утилит с графическим интерфейсом пользователя на примере утилиты IB Expert.

Программа работы

Установить СУБД Firebird. Создать базу данных, создать несколько таблиц в базе данных, выбрать первичные ключи в них, создать связи между таблицами.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Описание работ по установке СУБД
3. Описание возможностей используемых утилит.
4. Команды утилит с консольным интерфейсом, которые были использованы при выполнении работы.

Лабораторная работа №2

Цель работы: Освоение подмножества DDL языка SQL и использования скрипа для генерации базы данных.

Программа работы

Используя утилиту IB Expert составить скрипт для создания нескольких таблиц, полей, связей между таблицами, проверок средствами SQL.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Краткое изложение теории.
3. Концептуальная модель данных.
4. Логическая модель данных.
5. Физическая модель данных.
6. Скрипт SQL для генерации данных.
7. Описание полученных результатов.
8. Вывод о проделанной работе.

Лабораторная работа №3

Цель работы: Освоение подмножества DML языка SQL.

Программа работы

Используя утилиту IB Expert составить запросы на языке SQL с использованием SELECT, INSERT, UPDATE, DELETE. Познакомится с механизмом транзакций.

Отчет по лабораторной работе включает в себя:
1. Цель работы.
2. Краткое изложение теории.
3. Запросы к базе данных, разработанные во время выполнения лабораторной работы.
4. Описание полученных результатов.
5. Выводы о проделанной работе.

Лабораторная работа №4

Цель работы: Освоение основных операторов процедурного подмножества языка SQL и их использования.

Программа работы

Используя операторы процедурного подмножества языка SQL создать хранимые процедуры для извлечения и добавления данных в базу данных, создать триггеры для проверки ограничений предметной области.

Отчет по лабораторной работе включает в себя:
1. Цель работы.
2. Краткое изложение теории.
3. Запросы к базе данных, разработанные во время выполнения лабораторной работы.
4. Описание полученных результатов.
5. Выводы о проделанной работе.

Лабораторная работа №5

Цель работы: Освоение принципов разработки приложений-клиентов для работы с клиент-серверными СУБД.

Программа работы

Разработать простое приложение-клиент, которое предназначено для заполнения базы данных, извлечения и изменения данных в базе данных.

В ходе работы требуется: разработать интерфейс приложения с пользователем; реализовать подключение к базе данных с использованием одной из перечисленных технологий - ODBC, GDBC, Ado, AdoNet, linq.

Отчет по лабораторной работе включает в себя:

1. Цель работы.
2. Описание интерфейса приложения с пользователем.
3. Краткое описание используемой технологии подключения к СУБД.
4. Листинг программы.
5. Описание полученных результатов.
6. Выводы о проделанной работе.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 69,1 ч.

№	Наименование тем	Кол-во часов	Номер компетенции	Литература
1	Тема 1. Структуры данных и методы доступа к ним	8	ОПК-4, ПК-2	7 [1-4]
2	Тема 2. Модель "сущность-связь"	8	ОПК-4, ПК-2	7 [1-4]
3	Тема 3. Иерархическая и сетевая модели данных СУБД	6	ОПК-4, ПК-2	7 [1-4]
4	Тема 4. Нормальные формы более высоких порядков	10	ОПК-4, ПК-2	7 [1-4]
5	Тема 5. Реляционная алгебра	10	ОПК-4, ПК-2	7 [1-4]
6	Тема 6. Язык SQL	8	ОПК-4, ПК-2	7 [1-4]
7	Тема 7. Этапы проектирования данных	10	ОПК-4, ПК-2	7 [1-4]
8	Тема 8. Вопросы практического программирования	9,1	ОПК-4, ПК-2	7 [1-4]

3.1. Задания для самостоятельного изучения и конспектирования

Тема 1. Структуры данных и методы доступа к ним.
Рассмотреть следующие вопросы:
- Типы и структуры данных;
- Обобщенные структуры или модели данных;
- Методы доступа к данным;
- Методы поиска по дереву;
- Методы хеширования.

Тема 2. Модель "сущность-связь".
Рассмотреть следующие вопросы:
- Модель "сущность-связь";
- Элементы модели "сущность-связь";
- Диаграмма "сущность-связь";
- Обзор нотаций, используемых при построении диаграмм "сущность-связь".

Тема 3. Иерархическая и сетевая модели данных СУБД.
Рассмотреть следующие вопросы:
- Иерархическая и сетевая модели данных СУБД;
- Иерархическая модель данных. Сетевая модель данных.

Тема 4. Нормальные формы более высоких порядков.
Рассмотреть следующие вопросы:
- Нормальная форма Бойса-Кодда;
- Четвертая нормальная форма;
- Многозначная зависимость;
- Пятая нормальная форма;
- Зависимость соединения;
- Алгоритм нормализации.

Тема 5. Реляционная алгебра.
Рассмотреть следующие вопросы:
- Обзор реляционной алгебры и основные определения;
- Теоретико-множественные операторы;
- Специальные реляционные операторы;

- Примеры использования реляционных операторов;
- Зависимые реляционные операторы;
- Примитивные реляционные операторы.

Тема 6. Язык SQL.

Рассмотреть следующие вопросы:

- Порядок выполнения оператора SELECT;
- Реализация реляционной алгебры средствами оператора SELECT;
- Использование представлений. Хранимые процедуры. Триггеры.

Тема 7. Этапы проектирования данных.

Рассмотреть следующие вопросы:

- Представления предметной области;
- Концептуальное проектирование;
- Логическое проектирование;
- Физическое проектирование;
- Инструментальные средства проектирования информационных систем;
- Методологии функционального моделирования;
- Правила порождения реляционных отношений из модели "сущность-связь";
- Логическая модель данных;
- Физическая модель данных;
- Проектирование реляционной базы данных на основе декомпозиции универсального отношения.

Тема 8. Вопросы практического программирования.

Рассмотреть следующие вопросы:

- Способы организации доступа прикладной программы к серверу базы данных;
- Использование специализированных библиотек и встраиваемого SQL;
- CLI - интерфейс уровня вызовов;
- ODBC - открытый интерфейс к базам данных;
- JDBC - мобильный интерфейс к базам данных на платформе Java;
- Архитектура "клиент-сервер";
- Модели взаимодействия клиент-сервер.

4. КУРСОВАЯ РАБОТА

Объем курсовой работы: 25-30 листов.

Структура курсовой работы:

Титульный лист – представляет собой первую страницу курсовой, выполняется по определенным правилам.

Лист задания – является стандартным листом. Как и титульный лист выполняется по определенным правилам.

Оглавление – содержит в себе все разделы и подразделы курсовой работы, с обязательным указанием соответствующих страниц.

Введение – включает основные цели и задачи выполнения работы, конкретизируется объект исследования и предмет.

Теоретический раздел – предполагает исследование методик изучения проблемы, применяющихся в отечественной и зарубежной практике. Здесь рассматриваются основные понятия проблемы, причины возникновения. Выполняется анализ объекта исследования, выявлением «слабых мест», математическим просчетом основных параметров и показателей.

Практический раздел – выполняется на основе теоретического раздела. Включает в себя подробное описание разработанного приложения со скриншотами.

Заключение – содержит общие выводы по курсовой работе. Должно быть выполнено последовательно, исходя из поставленных задач во введении. Также может содержать оценку, удалось ли достичь поставленной цели или нет.

Список используемых источников – должен содержать библиографию, которая использовалась при написании работы и имеет сноски на конкретный источник.

Приложения. Содержит полный листинг программы и иллюстративные материалы, которые позволяют визуально воспринять результаты работы.

4.1. Указания к выполнению курсовой работы

Цели и задачи разработки уточняются студентом самостоятельно. Концептуальная модель данных должна содержать не менее 8 и не более 20 сущностей. Приложение-клиент для взаимодействия с сервером и база

данных должны быть разработаны с учетом возможности одновременной работы нескольких пользователей.

4.2. Примерные задания на курсовую работу

1. Информационная система ресторана.
2. Информационная система интернет-магазина мобильной электроники и ПК.
3. Информационная система библиотеки вуза.
4. Информационная система «Зоопарк», для решения задачи о сбалансированном кормлении животных.
5. Информационная система материального обеспечения производственных процессов машиностроительного завода.
6. Информационная система платной поликлиники.
7. Информационная система мебельного магазина.
8. Информационная система школы.
9. Информационная система отеля.
10. Информационная система «Отдел кадров».
11. Информационная система по учету продаж, оптимизации расценок и ассортимента (с учетом его характеристик) магазина материалов, деталей и инструментов для выполнения строительных работ.
12. Информационная система ветеринарной аптеки.
13. Информационная система дачного кооператива.
14. Информационная система оружейного магазина.
15. Информационная система агентства по продаже недвижимости.
16. Информационная система строительной компании.
17. Информационная система компании по изготовлению продаже и установке пластиковых окон.
18. Информационная система зоомагазина.
19. Информационная система фитнес-клуба.
20. Информационная система склада.
21. Информационная система продуктового магазина.
22. Информационная система велосипедного завода.

5. КОНТАКТНАЯ ВНЕАУДИТОРНАЯ РАБОТА

СРС – групповые консультации с преподавателем в течение

семестра – 0,9 ч.

Групповая консультация к защите курсовой работы – 1,5 ч.

Групповая консультация перед экзаменом – 2 ч.

СРС экз. – сдача экзамена – 0,35 ч.

5.1. Вопросы для оценивания знаний

1. Для чего нужны базы данных.
2. Основные термины и определения.
3. Компоненты СУБД.
4. Системы, основанные на файлах.
5. Основные типы данных и их классификация.
6. Модели данных и их компоненты.
7. Типы моделей СУБД.
8. Методы доступа к данным.
9. Поиск по дереву.
10. Хеширование.
11. Назначение модели «сущность-связь».
12. Элементы модели «сущность-связь».
13. Диаграмма «сущность-связь».
14. Нотации, используемых при построении диаграмм «сущность-связь».
15. Иерархическая модель данных.
16. Сетевая модель данных.
17. Реляционная модель данных.
18. Отношения.
19. Потенциальные ключи.
20. Целостность сущностей.
21. Внешние ключи.
22. Целостность внешних ключей.
23. Операции, влияющие на целостность внешних ключей.
24. Стратегии поддержания ссылочной целостности.
25. Этапы разработки баз данных.
26. Критерии оценки качества логической модели данных.
27. Первая нормальная форма (1НФ).
28. Функциональные зависимости.
29. Вторая нормальная форма (2НФ).
30. Третья нормальная форма (3НФ).

31. Сравнение нормализованных и ненормализованных моделей данных.
32. OLTP и OLAP-системы.
33. Нормальная форма Бойса-Кодда (НФБК).
34. Многозначная зависимость.
35. Четвертая нормальная форма (4НФ).
36. Зависимость соединения.
37. Пятая нормальная форма (5НФ).
38. Обзор реляционной алгебры.
39. Операции реляционной алгебры: объединение, пересечение, вычитание.
40. Операции реляционной алгебры: декартово произведение, выборка, проекция.
41. Операции реляционной алгебры: соединение.
42. Операции реляционной алгебры: деление.
43. Язык SQL. История развития. Структура языка. Типы данных SQL.
44. Операторы создания и модификации схемы базы данных (DATABASE, TABLE).
45. Операторы создания индексов. Операторы управления правами доступа.
46. Команды модификации данных, условия отбора записей.
47. Оператор SELECT. Синтаксис оператора, простые формы оператора, условия отбора записей.
48. Выборка из нескольких таблиц, синтаксис соединения таблиц.
49. Использование алиасов и псевдонимов, вложенные запросы (подзапросы).
50. Вычисления внутри оператора, группировка данных, сортировка данных, операция объединения.
51. Формальный порядок выполнения оператора SELECT.
52. Как на самом деле выполняется оператор SELECT.
53. Реализация реляционной алгебры средствами оператора SELECT.
54. Использование представлений (View).
55. Хранимые процедуры.
56. Триггеры.
57. Транзакции, блокировки и многопользовательский доступ к данным.

58. Этапы проектирования данных.
59. Трехуровневая схема (модель ANSI/SPARC).
60. Концептуальное, логическое и физическое проектирование.
61. Инструментальные средства проектирования информационных систем. CASE-технология.
62. Методология функционального моделирования.
63. Концептуальное моделирование.
64. Правила порождения реляционных отношений из модели «сущность-связь».
65. Логическая модель данных.
66. Физическая модель данных.
67. Проектирование реляционной базы данных на основе декомпозиции универсального отношения.
68. Создание приложений, работающих с базами данных при помощи языка SQL. Общие подходы.
69. Специализированные библиотеки доступа.
70. CLI-интерфейс уровня вызова.
71. Вопросы практического программирования. ODBC и JDBC - интерфейсы доступа к базам данных.
72. Архитектура "клиент-сервер".
73. Структура сервера базы данных.

5.2. Вопросы для оценивания умений и навыков

При проверке умений и навыков следует указать ответ и привести решение задачи.

Спроектируйте базу данных, состоящую из нескольких таблиц таким образом, чтобы в многотабличной системе были таблицы со связью 1:1, 1:N. Используя созданную базу данных, спроектируйте указанный запрос на языке SQL:

1. Исходные данные: Данные для простой складской системы. База данных должна содержать следующую информацию: уникальный номер поставщика, фамилию, имя, отчество поставщика, название города местонахождения поставщика, а также уникальный номер детали, ее название, цвет, вес и название города хранения деталей этого типа.

2. Условия запроса: Запрос для вывода списка деталей, хранящихся в Москве, с указанием уникального номера детали, ее веса, цвета, фамилии поставщика.

3. Исходные данные: Сведения об участниках конкурса бальных танцев. База данных должна содержать следующую информацию: фамилию, имя, отчество участника, город, фамилию тренера, оценки за каждый танец.

4. Условия запроса: Запрос, для вывода списка участников конкурса, приехавших из Ульяновска, содержащий фамилию, имя, отчество участника, фамилию тренера.

5. Исходные данные: Сведения об успеваемости студентов. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер группы, в которой обучается студент, название учебной дисциплины, номер задания, коэффициент сложности, оценку данного студента по данной дисциплине за данное задание от 0 до 1 (как доля сделанной работы).

6. Условия запроса: Запрос для вывода списка студентов, у которых оценка выше или равна 0,5, с указанием ФИО студента, номера группы, названия дисциплины, варианта задания и коэффициента его сложности.

7. Исходные данные: Сведения о месячной зарплате рабочих. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха, в котором он работает, дату поступления на работу. По заработной плате необходимо хранить информацию о ее размере, стаже работника, его разряде и должности.

8. Условия запроса: Запрос для вывода списка рабочих, имеющих одну и ту же должность, с указанием ФИО рабочего, названия цеха, стажа и разряда.

9. Исходные данные: Учет изделий, собранных в цехе за неделю. База данных должна содержать следующую информацию: фамилию, имя, отчество сборщика, количество изготовленных изделий за каждый день недели раздельно, название цеха, а также тип изделия и его стоимость.

10. Условия запроса: Запрос для вывода списка изделий за каждый день недели с указанием типа изделия и его стоимости.

11. Исходные данные: Учет изделий категорий A, B, C, собранных рабочим цеха за месяц. База данных должна содержать следующую информацию: фамилию, имя, отчество рабочего, название цеха,

количество изделий по категориям, количество рабочих в цехе и фамилию начальника цеха.

12. Условия запроса: Запрос для вывода списка рабочих в каждом цехе с указанием количества изделий по категориям, выполненных каждым рабочим.

13. Исходные данные: Сведения об абонентах АТС. База данных должна содержать следующую информацию: фамилию, имя, отчество владельца телефона, год установки телефона, номер телефона, тип установки телефона (спаренный или нет), льготу (процентную скидку при оплате).

14. Условия запроса: Запрос для вывода номеров телефонов абонентов, у которых есть льготы, с указанием ФИО и адреса владельца, года установки и льготы.

15. Исходные данные: Сведения об ассортименте игрушек в магазине. База данных должна содержать следующую информацию: название игрушки, ее цену, количество, возрастную категорию детей, для которых она предназначена, а также название фабрики и города, где изготовлена игрушка.

16. Условия запроса: Запрос для вывода списка игрушек, предназначенных для одной и той же возрастной категории, с указанием названия игрушки, ее цены и количества.

17. Исходные данные: Результаты сессии на первом курсе кафедры ВТ. База данных должна содержать следующую информацию: индекс группы, фамилию, имя, отчество студента, пол студента, семейное положение и оценки по пяти экзаменам.

18. Условия запроса: Запрос для вывода списка студентов, у которых есть своя семья.

19. Исходные данные: Учет рейтинга теннисистов за 5 лет. Каждая запись содержит поля: фамилия, имя, отчество спортсмена, пол, год рождения, фамилия, имя, отчество тренера, названия стран и пять полей с рейтингом.

20. Условия запроса: Запрос для вывода списка Российских спортсменов, с указанием фамилии, имени, отчества спортсмена, пола, года рождения.

21. Исходные данные: Сведения о рейсах Аэрофлота. База данных должна содержать следующую информацию: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест, тип самолета и его вместимость.

22. Условия запроса: Запрос для вывода номеров всех рейсов, у которых есть хотя бы одно свободное место, с указанием пункта назначения, времени вылета и времени прибытия.

23. Исходные данные: Сведения об ассортименте обуви в магазине. База данных должна содержать следующую информацию: артикул, наименование обуви, количество пар, стоимость одной пары, имеющиеся размеры, название фабрики и срок поставки обуви в магазин.

24. Условия запроса: Запрос для вывода списка наименований обуви, имеющей один и тот же размер, с указанием артикула, стоимости, названия фабрики-изготовителя и количества пар.

25. Исходные данные: Сведения о нападающих команд “Спартак” и “Динамо”. База данных должна содержать следующую информацию: фамилию, имя, отчество, название команды, дату приема в команду, число заброшенных шайб, количество голевых передач, штрафное время и количество сыгранных матчей.

26. Условия запроса: Запрос для вывода списка нападающих, у которых количество сыгранных матчей больше трех, а число заброшенных шайб больше 15, с указанием ФИО нападающего и названия команды.

27. Исходные данные: Сведения о выборе дисциплины студентом. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, номер зачетной книжки и сведения о том, живет ли студент в общежитии, индекс группы, а также пять дисциплин (1 – желает изучать, 0 – не желает).

28. Условия запроса: Запрос для вывода списка студентов, обучающихся в одной и той же группе, с указанием ФИО студента, номера его зачетной книжки и выбранных дисциплин.

29. Исходные данные: Журнал регистрации расходов в бухгалтерии. База данных должна содержать следующую информацию: номер пункта, дату перечисления, название организации-получателя, ее адрес и сведения о том, является ли организация коммерческой, а также вид затрат перечисления и общую сумму перечисления.

30. Условия запроса: Запрос для вывода списка названий и адресов коммерческих организаций- получателей, а также суммарные перечисления по видам затратам.

31. Исходные данные: Учет оптовых продаж. База данных должна содержать следующую информацию: наименование товара, цену единицы товара и дату его поступления, номер партии, размер партии, названии

фирмы-покупателя, размер проданной партии, цену единицы товара и дату продажи.

32. Условия запроса: Запрос для вывода списка фирм, когда-либо покупавших товары.

33. Исходные данные: Учет лекарств в аптеке. База данных должна содержать следующую информацию: наименование лекарства, стоимость одной единицы, количество единиц, дату изготовления, срок годности, а также название фабрики, где производится данное лекарство, ее адрес.

34. Условия запроса: Запрос для вывода списка наименований лекарств, произведенных на какой- либо фабрике.

35. Исходные данные: Сведения о ветеранах спорта. Ассоциация ветеранов спорта проводит Всероссийские соревнования ветеранов. Для организации соревнований составляются списки участников, которые используются для размещения спортсменов в гостиницах. Для каждого спортсмена указывается гостиница, номер комнаты и количество мест в комнате. Для нужд самой ассоциации ветеранов спорта необходимо хранить информацию следующего вида: фамилию, имя, отчество спортсмена, возрастную группа, название города и вид спорта.

36. Условия запроса: Запрос для ограничения числа отображаемых записей по фамилиям ветеранов спорта с указанием пола спортсмена, его возрастной группы, города, в котором он проживает, и вида спорта.

37. Исходные данные: Учет рождаемости в роддоме. База данных должна содержать следующую информацию: фамилию, имя, отчество матери, пол ребенка, его вес, рост и дату рождения ребенка, а также ФИО лечащего врача и номер палаты, в которой находится мать ребенка.

38. Условия запроса: Запрос для вывода списка рожениц, лечащихся у одного и того же врача, с указанием ФИО матери, номера палаты, в которой она лежит, и даты рождения ребенка.

39. Исходные данные: Сведения об обучающихся на курсах повышения квалификации. База данных должна содержать следующую информацию: фамилию, имя, отчество слушателя, его пол и адрес, тип организации (коммерческая, государственная и т.д.), наименование организации, должность слушателя и оценки по прослушанным дисциплинам (маркетинг, финансы и кредит) для каждого слушателя.

40. Условия запроса: Запрос для вывода списка слушателей, которые не сдали хотя бы одну дисциплину, с указанием ФИО слушателя и названия организации.

41. Исходные данные: Сведения о размере стипендии студента. База данных должна содержать следующую информацию: фамилию, имя, отчество студента, группу, адрес, по которому проживает студент, размер стипендии, а также фамилию преподавателя, читаемую дисциплину, дату экзамена и оценку, полученную студентом.

42. Условия запроса: Запрос для вывода списка студентов, получающих стипендию, с указанием группы и размера стипендии.

43. Исходные данные: Учет поступления больных. База данных должна содержать следующую информацию: фамилию, имя, отчество больного, его пол, дату рождения, дата поступления, начальный диагноз, степень тяжести состояния больного, номер палаты, куда был помещен больной, и фамилию, имя, отчество лечащего врача.

44. Условия запроса: Запрос для вывода списка больных, лечащихся у одного и того же врача, с указанием ФИО больного, даты его рождения, пола, начального диагноза и степени тяжести состояния больного.

45. Исходные данные: Учет призывников. База данных должна содержать следующую информацию: фамилию, имя, отчество призывника, год его рождения, адрес, семейное положение, дату прохождения медкомиссии, заключение о пригодности к службе.

46. Условия запроса: Запрос для вывода списка призывников, не пригодных к службе, с указанием ФИО призывника и года его рождения.

47. Исходные данные: Учет золотых изделий в ювелирном магазине. База данных должна содержать следующую информацию: вид изделия, его вес, пробу, дату поступления и стоимость, а также фамилию, имя, отчество мастера-изготовителя, стаж его работы и разряд (1,2,3).

48. Условия запроса: Запрос для вывода списка изделий, выполненных одним и тем же мастером-ювелиром, с указанием вида изделия, его веса, пробы и стоимости.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Основная учебная литература

1. Кузнецов С.Д. Введение в реляционные базы данных [Электронный ресурс] – М.: Национальный Открытый Университет «ИНТУИТ», 2016 – 248 с. – Режим доступа: http://www.knigafund.ru/books/.

2. Карпова Т.С. Базы данных: модели, разработка, реализация [Электронный ресурс] – М.: Национальный Открытый Университет «ИНТУИТ», 2008 – 357 с. - Режим доступа: http://www.knigafund.ru/books/.

Дополнительная учебная литература

3. Полякова Л.Н. Основы SQL: Курс лекций. Учебное пособие. [Электронный ресурс] – М.: ИНТУИТ.РУ, Интернет – Университет Информационных Технологий, 2004 – 368 с. – Режим доступа: http://www.knigafund.ru/.

4. Кариев Ч.А. Технология Microsoft ADO.NET: учебное пособие [Электронный ресурс] – М.: Интернет – Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007 – 544 с.: с ил., табл. – Режим доступа: http://www.knigafund.ru/books/.

5. Ачкасов В. Ю. Программирование баз данных в Delphi [Электронный ресурс] – М.: Национальный Открытый Университет «ИНТУИТ», 2010 – 382 с. – Режим доступа: http://www.knigafund.ru/books/.

Учебно-методическое издание

Шлыков Павел Васильевич

Гасанов Рафуг Ахадович

Основы управления данными

Методические указания
к курсовой работе, практическим занятиям, лабораторным работам и самостоятельной работе студентов

Редактор *Н.А.Юшко*
Подписано в печать 17.04.2017
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,39. Уч.-изд.л. 1,5 . Тираж 50. Заказ____ .

Южно-Российский государственный политехнический университет (НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова

Д.В. Гринченков, В.Б. Дьяченко, Д.Н. Кущий

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 681.3.06(075.8)
ББК 22.1я73
Г82

Рецензенты:

доктор технических наук, профессор А.А. Ткачев
(Новочеркасский инженерно-мелиоративный институт им. А.К. Кортунова филиал ФГБОУ ВО «Донской ГАУ»)

кандидат технических наук, профессор Д.В. Янченко
(Новочеркасский инженерно-мелиоративный институт им. А.К. Кортунова филиал ФГБОУ ВО «Донской ГАУ»)

Гринченков Д.В., Дьяченко В.Б., Кущий Д.Н.

Г82 Введение в программную инженерии: учебное пособие / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2016. – 91 с.

Пособие посвящено вопросам создания, эксплуатации и сопровождения программных продуктов. Содержит разделы, раскрывающие современные методы разработки сложного программного обеспечения. Содержит справку об истории становления, развития и основах одной из областей деятельности человека - программной инженерии, сведения об основных тенденциях, методах, этапах и инструментах разработки программного обеспечения, описание основных этапов жизненного цикла программного продукта, управление промышленными технологиями создания программных систем. Первая глава включает общие вопросы программной инженерии и основы построения программного обеспечения. Во второй главе раскрыты модели жизненного цикла как набор фаз проекта по созданию программного обеспечения, в которых выполняются отдельные процессы, разбитые на операции и задачи. Приведены примеры и методологии управления проектами разработки, рассмотрены промышленные технологии создания программного продукта. В третьей главе подробно рассмотрены процессы завершающего этапа жизненного цикла - сопровождение программного продукта, позволяющее продлить срок его эксплуатации.

Пособие предназначено для студентов всех форм и уровней обучения технических направлений подготовки вуза, связанных с изучением вопросов разработки программного обеспечения.

УДК 681.3.06(075.8)
ББК22.1я73

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ…….. 5
1.1. История возникновения программной инженерии, предпосылки, сущность, основные понятия и общая терминология……... 5
1.2.Предпосылки и история возникновения программной инженерии……. 11
1.3.Продолжение кризиса программирования……..… 14
1.4. Итоги становления и развития программной инженерии……. 16
2. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА… 17
2.1. Модель жизненного цикла программного продукта… 17
2.1.1.Каскадная модель…… 19
2.1.2. Спиральная (циклическая) модель……. 21
2.1.3. Другие типы моделей ЖЦ ПО…….. 26
2.2. Промышленные технологии создания программных продуктов…… 30
2.2.1. Модель Microsoft Solution Framework…… 32
2.2.2. Модель Rational Unified Process…… 35
2.2.3. Модель Extreme Programming…… 37
2.2.4. Методология SCRUM…… 40
2.2.5. Adaptive Software Development (ASD) методология…… 46
2.2.6. Семейство методологий Crystal…… 56
2.2.7. Методология Feature Driven Development (FDD)… 61
2.3. Сравнение методологий разработки ПО……..…. 62
2.3.1.Структурные методы…….. 63
2.3.2.Гибкие методологии……... 64
2.3.3. Модели зрелости процесса разработки…… 67
2.3.4. *Методология RUP*…….…….. 68
3. СОПРОВОЖДЕНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SOFTWARE MAINTENANCE)……. 71
3.1. Основы сопровождения программного обеспечения (Software Maintenance Fundamentals)……..… 73
3.1.1. Определения и терминология (Definitions and Terminology)……..…… 73
3.1.2. Природа сопровождения (Nature of Maintenance)… 74

3.1.3.Потребность в сопровождении (Need for Maintenance)…… 75
3.1.4. Приоритет стоимости сопровождения (Majority of Maintenance Costs)……..…….. 76
3.1.5. Эволюция программного обеспечения (Evolution of Software)……..… 76
3.1.6.Категории сопровождения (Categories of Maintenance)……..…….. 77
3.2. Ключевые вопросы сопровождения программного обеспечения (Key Issues in Software Maintenance)…… 77
3.2.1. Технические вопросы (Technical Issues)…….. 78
3.2.2. Управленческие вопросы (Management Issues)…… 79
3.2.3. Оценка стоимости сопровождения (Maintenance Cost Estimation)…….. 82
3.2.4. Измерения в сопровождении программного обеспечения (Software Maintenance Measurement)… 83
3.3. Процесс сопровождения (Maintenance Process)…… 85
3.4. Техники сопровождения (Techniques for Maintenance) 87
Библиографический список 90

1. ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

1.1. История возникновения программной инженерии, предпосылки, сущность, основные понятия и общая терминология

Программная инженерия есть применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения (*IEEE Standard Glossary of Software Engineering Terminology*). IEEE - Institute of Electrical and Electronics Engineers (Институт инженеров по электротехнике и электронике).

Термин software (программное обеспечение, ПО) ввел в 1958 году всемирно известный математик и статистик Джон Тьюки (John Tukey).

Джон Тьюки

Биография: Джон Уайлдер Тьюки (16.06.1915-26.07.2000). Стал бакалавром в 1936 г. и магистром в 1937 г. в Брауновском университете. Затем поступил в Принстонский университет, где получил степень доктора философии по математике. Во время Второй мировой войны работал в Fire Control Research Office, позже вернулся в университет. Параллельно участвовал в разработках при Bell Labs.

Премии и награды:

- 1965 - Samuel S. Wilks Award;

- 1973 - Национальная научная медаль США;
- 1976 - Shewhart Medal;
- 1982 - Deming Medal;
- 1982 - Медаль Почета IEEE «за его вклад в исследование спектрального анализа случайных процессов и алгоритм быстрого преобразования Фурье»;
- 1983 – James Madison Medal.

Термин software engineering (программная инженерия) впервые был озвучен в октябре 1968 года на конференции подкомитета НАТО по науке и технике (г. Гармиш, Германия). Присутствовало 50 профессиональных разработчиков ПО из 11 стран. Рассматривались проблемы проектирования, разработки, распространения и поддержки программ. Там впервые и прозвучал термин «программная инженерия» как некоторая дисциплина, которую надо создавать и которой надо руководствоваться в решении перечисленных проблем.

В 1969 г. Лондоне состоялась встреча 22-х руководителей проектов по разработке ПО. На встрече анализировались проблемы и перспективы развития ПО. Отмечалась возрастающее воздействие ПО на жизнь людей. Впервые серьезно заговорили о надвигающемся кризисе ПО. Применяющиеся принципы и методы разработки ПО требовали постоянного усовершенствования.

Именно на этой встрече была предложена концепция жизненного цикла ПО (SLC – Software Lifetime Cycle), как последовательности шагов-стадий, которые необходимо выполнить в процессе создания и эксплуатации ПО.

В 1970 г. произведена идентификация нескольких стадий в типичном цикле и высказано предположение, что контроль выполнения стадий приведет к повышению качества ПО и сокращению стоимости разработки.

С 1990-го по 1995 год велась работа над международным стандартом, который должен был дать единое представление о процессах разработки программного обеспечения. В результате был выпущен стандарт ISO/IEC 12207 (IEEE Standard for Developing Software Life Cycle Processes).

В 2004 году в отрасли был создан основополагающий труд «Руководство к своду знаний по программной инженерии». (The

Guide to the Software Engineering Body of Knowledge, сокращенно – SWEBOK), в котором были собраны основные теоретические и практические знания, накопленные в этой отрасли. При его создании были учтены около 9 тыс. замечаний, предложенных более чем 500 заинтересованными лицами из 42 стран.

Одним из основных принципов реализации проекта является формирование содержания SWEBOK при активном участии представителей различных направлений профессионального сообщества. Только при условии согласия между ними можно говорить о значимости и легитимности такого документа.

По утверждению авторов, к анализу и доработке SWEBOK были привлечены сотни специалистов, работающих в различных организациях и странах.

Ряд организаций вошли в консультативный орган Industrial Advisory Board, который осуществляет финансовую поддержку проекта, благодаря чему все его результаты публикуются в свободном доступе.

Домашняя страница SWEBOK приведена на рисунке 1.1. (Режим доступа http://www.computer.org/portal/web/swebok). Первую попытку изложить SWEBOK на русском языке предпринял Сергей Орлик, менеджер по бизнес-решениям представительства компании Borland. Автор позиционирует свою работу как «общедоступный перевод с замечаниями и комментариями». «Данная работа является персональной профессиональной инициативой, не финансируется и не ассоциирована в какой-либо форме ни с какой компанией или организацией», - подчеркивает Орлик.

Совместно с С. Орликом переводом SWEBOK занимается также директор департамента стратегических технологий российского представительства Microsoft Владимир Павлов, участвующий, помимо этого, в проекте по переводу образовательного стандарта по программной инженерии Curricula Software Engineering. Работа велась при поддержке компаний Microsoft, Intel и российской Ассоциации предприятий компьютерных и информационных технологий (АП КИТ).

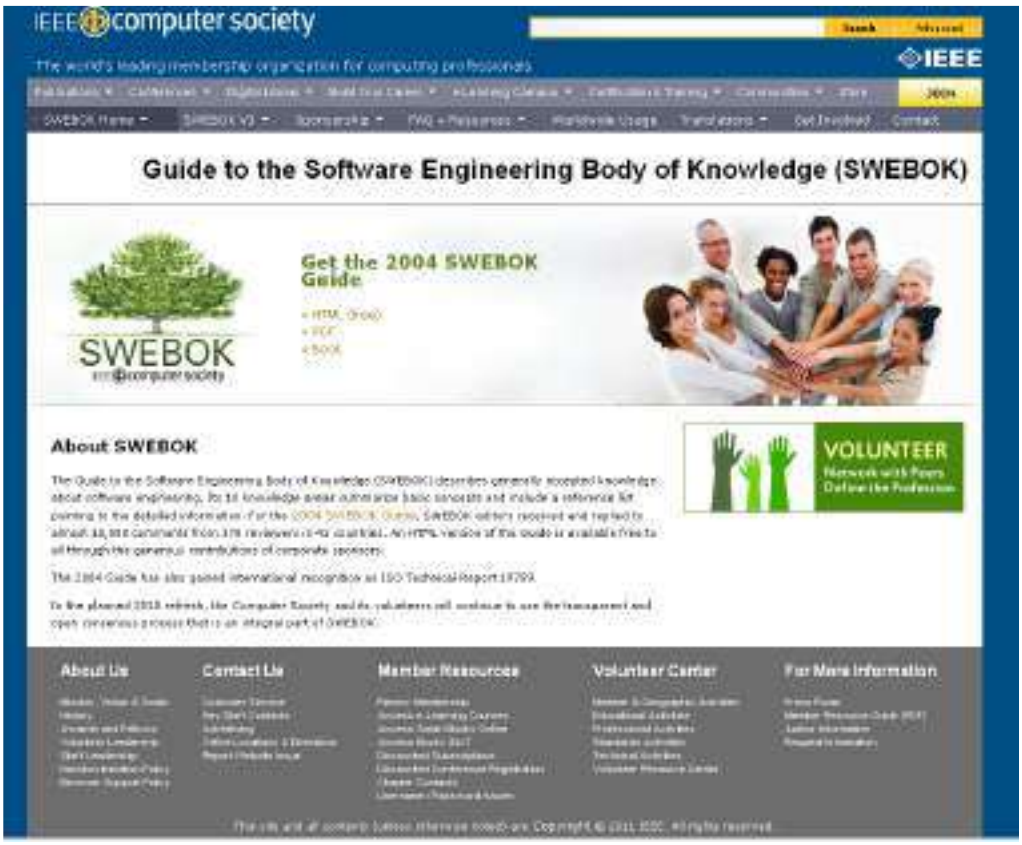


Рисунок 1.1 - Домашняя страница SWEBOK

Сергей Орлик

Рисунок 1.2 - Содержание SWEBOK

Профессия инженера подразумевает обязательное начальное профессиональное образование в соответствии с аккредитованным учебным планом, сертификацию, демонстрирующую пригодность специалиста к практической деятельности, и постоянное повышение квалификации.

SWEBOK аккумулирует в себе представление сообщества о том, что должен знать и какими навыками владеть инженер-программист, и задает базу с целью формирования экзаменационных программ для лицензирования специалистов, а также критериев аккредитации этих программ и учебных планов непрерывного образования (рисунок 1.2).

Для того чтобы задать границы программной инженерии как сферы профессиональной деятельности, SWEBOK вводит десять областей знаний, каждой из которых соответствует одна глава Руководства.

Десять областей знаний по SWEBOK:

1. Требования к программному обеспечению (Software Requirements).
2. Проектирование программного обеспечения (Software Design).

3. Конструирование программного обеспечения (Software Construction).
4. Тестирование программного обеспечения (Software Testing);
5. Сопровождение программного обеспечения (Software Maintenance).
6. Управление конфигурацией программного обеспечения (Software Configuration Management).
7. Управление в программной инженерии (Software Engineering Management).
8. Процесс программной инженерии (Software Engineering Process).
9. Инструменты и методы программной инженерии (Software Engineering Tools and Methods).
10. Качество программного обеспечения (Software Quality).

Отдельная глава посвящена описанию смежных по отношению к программной инженерии дисциплин, среди которых математика, компьютерные науки, менеджмент, управление проектами и др.

Одним из важнейших вопросов, которые необходимо было решить в ходе реализации проекта SWEBOK, был вопрос о глубине охвата знаний по программной инженерии в создаваемом Руководстве.

В SWEBOK представлены так называемые «общепринятые» знания, получившие широкое распространение в индустрии знания и навыки, применение которых не является обязательным во всех возможных случаях, но владение которыми необходимо для любого квалифицированного разработчика.

Еще одно определение общепринятых знаний в SWEBOK отражает практику подготовки инженеров-программистов в США: это знания, которыми должны обладать разработчики, имеющие степень бакалавра и четыре года практической работы, для успешного прохождения лицензионного экзамена.

Инженерный подход к описанию дисциплины в SWEBOK проявляется в том, что в документе никак не затрагиваются конкретные аспекты ИТ-технологий, подверженные в последние годы постоянным изменениям.

Предполагается, что, обладая багажом знаний, определенным

в SWEBOK, разработчик сможет выбрать оптимальное технологическое решение.

SWEBOK не претендует на исчерпывающее представление всех знаний по программной инженерии. Документ описывает необходимую, но, очевидно, недостаточную их совокупность, оставляя простор для дополнения этого обязательного багажа в учебных планах и рекомендациях по практической деятельности информацией о конкретных технологиях, методиках и компонентах смежных предметных областей.

Авторы SWEBOK подчеркивают, что «Содержание документа неизбежно будет эволюционировать».

1.2. Предпосылки и история возникновения программной инженерии

В конце 60-х – начале 70-х годов прошлого века произошло событие, которое вошло в историю как первый кризис программирования.

Событие состояло в том, что стоимость программного обеспечения стала приближаться к стоимости аппаратуры («железа»), а динамика роста этих стоимостей позволяла прогнозировать, что к середине 90-годов все человечество будет заниматься разработкой программ для компьютеров. Тогда и заговорили о программной инженерии (или технологии программирования, как это называлось в России) как о некоторой дисциплине, целью которой является сокращение стоимости программ.

С тех пор программная инженерия прошла достаточно бурное развитие. Каждый этап связан с появлением (или осознанием) очередной проблемы и нахождением путей и способов решения этой проблемы. Эти методы составляют основу подходов к проектированию программных продуктов.

В настоящее время существуют следующие проблемы.

Повторное использование кода (модульное программирование).

Проблема 1. На первых этапах становления программной инженерии было отмечено, что высокая стоимость программ связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах. Вызвано это было тем, что в различных

программах как части этих программ решались одинаковые (или похожие) задачи: решение нелинейных уравнений, расчет заработной платы и т.п. Использование при создании новых программ ранее написанных фрагментов сулило существенное снижение сроков и стоимости разработки.

Решение проблемы 1. Модульное программирование. Главный принцип модульного программирования состоял в выделении таких фрагментов и оформлении их в виде модулей. Каждый модуль снабжался описанием, в котором устанавливались правила его использования – интерфейс модуля.

Интерфейс задавал связи модуля с основной программой – связи по данным и связи по управлению. При этом возможность повторного использования модулей определялась количеством и сложностью этих связей, или насколько эти связи удалось согласовывать с организацией данных и управления основной программы.

Например, наиболее простыми в этом отношении оказались модули решения математических задач: решения уравнений, систем уравнений, задач оптимизации. К настоящему времени накоплены и успешно используются большие библиотеки таких модулей.

Для многих других типов модулей возможность их повторного использования оказалась проблематичной в виду сложности их связей с основной программой.

Например, модуль расчета зарплаты, написанный для одной фирмы, может не подойти для другой, т.к. зарплата в этих фирмах рассчитывается не во всем одинаково. Повторное использование модулей со сложными интерфейсами является достаточно актуальной и по сей день. Для ее решения разрабатываются специальные формы (структуры) представления модулей и организации их интерфейсов.

Рост сложности программ (структурное программирование).

Проблема 2. Следующий этап возрастания стоимости ПО был связан с переходом от разработки относительно простых программ к разработке сложных программных комплексов. К числу таких сложных программ относятся: системы управления космическими объектами, управления оборонным комплексом,

автоматизации крупного финансового учреждения и т.д. Сложность таких комплексов оценивалась следующими показателями:

- Большой объем кода (миллионы строк).
- Большое количество связей между элементами кода.
- Большое количество разработчиков (сотни человек).
- Большое количество пользователей (сотни и тысячи).
- Длительное время использования.

Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию. По аналогии с промышленной технологией стали говорить о жизненном цикле программного продукта, как о последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения.

Решение проблемы 2. Рост сложности программ (структурное программирование). Структурное программирование. Этап сопровождения программного комплекса включал действия по исправлению ошибок в работе программы и внесению изменений в соответствии с изменившимися требованиями пользователей. Основная причина высокой стоимости (а порой и невозможности выполнения) этапа сопровождения состояла в том, что программы были плохо спроектированы – документация была не понятна и не соответствовала программному коду, а сам программный код был очень сложен и запутан. Появилась необходимость создания технологии, которая обеспечит «правильное» проектирование и кодирование.

Основные принципы технологии структурного проектирования и кодирования:

- Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т.д. (принцип «разделяй и властвуй»).
- Применение специальных языков проектирования и средств автоматизации использования этих языков.
- Дисциплина проектирования и разработки: планирование и документирование проекта, поддержка соответствие кода проектной документации.

· Структурное кодирование без goto.

Модификация программ, объектно-ориентированное программирование (ООП). Проблема 3.

Следующая проблема роста стоимости программ была вызвана тем, что изменение требований к программе стали возникать не только на стадии проектирования, но и на стадии сопровождения – проблема заказчика, который не знает, что он хочет.

Создание программного продукта превратилось в его перманентное перепроектирование.

Возник вопрос как проектировать и писать программы, чтобы обеспечить возможность внесений изменений в программу, не меняя ранее написанного кода.

Модификация программ (ООП). Решение проблемы 3. Решением этой проблемы стало использование подхода или метода, который стали называть объектно-ориентированным проектированием и программированием.

Суть подхода состоит в том, что вводится понятие класса как развитие понятия модуля с определенными свойствами и поведением, характеризующими обязанностями класса.

Каждый класс может порождать объекты – экземпляры данного класса. При этом работают основные принципы (парадигмы) ООП:

Инкапсуляция – объединение в классе данных (свойств) и методов (процедур обработки).

Наследование – возможность вывода нового класса из старого с частичным изменением свойств и методов.

Полиморфизм – определение свойств и методов объекта по контексту.

1.3. Продолжение кризиса программирования

Несмотря на то, что программная инженерия достигла определенных успехов, перманентный кризис программирования продолжается. Связано это с тем, что рубеж 80–90-х годов отмечается как начало информационно-технологической революции, вызванной взрывным ростом использования информационных средств: персональных компьютеров, локальных и глобальных вычислительных сетей, мобильной связи

и электронной почты.

Цена успеха – кризис программирования принимает хронические формы, в США тратит ежегодно более $300 млрд. на более чем 200 тыс. проектов разработки ПО в сфере ИТ. Потери от недополученного эффекта внедрения ПО измеряются триллионами.

Проекты при этом делят на три категории:

- Успешные – вовремя и в рамках бюджета был выполнен весь намеченный фронт работ.
- Проблемные – нарушение сроков, перерасход бюджета и/или выполняется не все, что требовалось.
- Проваленные – не были доведены до конца из-за перерасхода средств, бюджета, качества.

Статистика по разработке ПО в американских компаниях, например, распределение проектов для The Standish Group International Inc., приведена на рисунке 1.3.

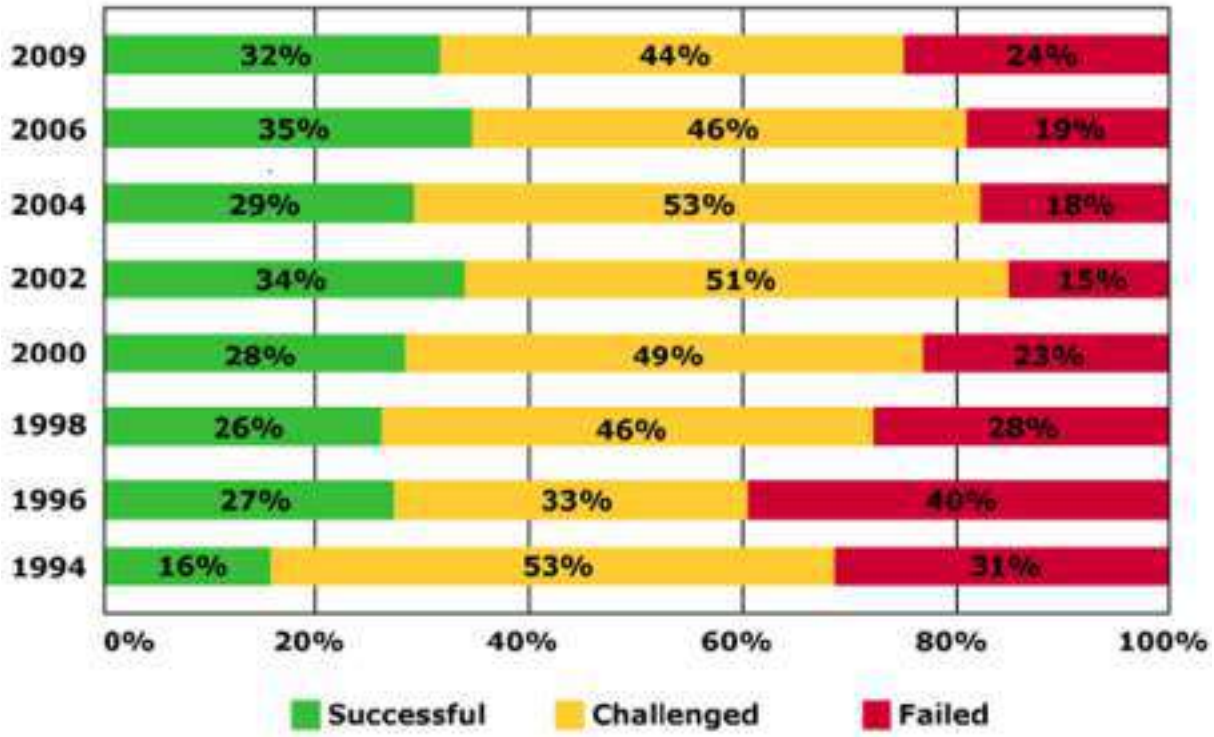


Рисунок 1.3 - Распределение проектов между категориями в компании The Standish Group International, Inc.

1.4. Итоги становления и развития программной инженерии

Программная инженерия (или технология программирования) как некоторое направление возникло и формировалось под давлением роста стоимости создаваемого программного обеспечения. Главная цель этой области знаний - сокращение стоимости и сроков разработки программ.

Программная инженерия прошла несколько этапов развития, в процессе которых были сформулированы фундаментальные принципы и методы разработки программных продуктов.

Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта.

Фундаментальными методами проектирования и разработки являются модульное, структурное и объектно-ориентированное проектирование и программирование.

Таким образом, понятие «Программная инженерия» получает следующие взаимодополняющие определения:

- установление и использование обоснованных инженерных принципов (методов) для экономного получения ПО, которое надежно и работает на реальных машинах. [Bauer 1972];
- та форма инженерии, которая применяет принципы информатики (computer science) и математики для рентабельного решения проблем ПО [CMU/SEI-90-TR-003];
- применение систематического, дисциплинированного, измеряемого подхода к разработке, использованию и сопровождению ПО [IEEE 1990];
- дисциплина, целью которой является создание качественного ПО, которое завершается вовремя, не превышает выделенных бюджетных средств и удовлетворяет выдвигаемым требованиям.

2. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА

2.1. Модель жизненного цикла программного продукта

Модель жизненного цикла (ЖЦ) ПО описывает набор фаз (этапов, стадий) проекта по созданию ПО, в которых выполняются отдельные процессы, разбитые на операции и задачи. Рассмотрим определения этих понятий, которые даются в глоссарии PMI (PMI. Глоссарий http://www.pmi.ru/glossary/).

Жизненный цикл проекта. Набор обычно последовательных фаз проекта, количество и состав которых определяется потребностями управления проектом организацией или организациями, участвующими в проекте (рисунок 2.1).

Фаза проекта. Объединение логически связанных операций проекта, обычно завершающихся достижением одного из основных результатов.

Процесс. Набор взаимосвязанных ресурсов и работ, благодаря которым входные воздействия преобразуются в выходные результаты.

Операция, работа. Элемент работ проекта. У операций обычно имеется ожидаемая длительность, потребность в ресурсах, стоимость. Операции могут далее подразделяться на задачи.

В этих определениях существенным является следующее:

- состав, количество и, как дополнение, порядок выполнения фаз определяется особенностью проекта;
- каждая фаза завершается получением одного из основных результатов, в то время как процесс или задача – просто получение значимого результата.

Рисунок 2.1 - Процессы и фазы программных проектов

Для схемы модели ЖЦ ПО характерно следующее.

Результатом выполнения каждой фазы является некоторая модель ПО. Описание требований – модель того, что должен делать программный продукт; результат анализа – модель основных архитектурных решений и т.д.

Результат выполнения каждой фазы является входом следующей фазы, при этом фазы должны выполняться в определенной моделью ЖЦ последовательности.

Некоторые процессы могут выполняться на нескольких фазах, некоторые – в пределах одной.

В стандарте ISO 12207 модель жизненного цикла (life cycle model) определяется как структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения ее использования.

При этом, конкретные модели определяются особенностью задач, ограничениями на ресурсы, опытом разработчиков и т.д.

Известны некоторые типовые модели ЖЦ ПО, которые проявили себя в определенных условиях, имеют определенные преимущества, недостатки и условия применимости. Эти типовые модели устанавливают некоторые принципы организации модели жизненного цикла ПО.

Каскадная модель (водопад - waterfall) включает выполнение следующих фаз (рисунок 2.2):

Исследование концепции — происходит исследование требований, разрабатывается видение продукта и оценивается возможность его реализации.

Определение требований — определяются программные требования для информационной предметной области системы, предназначение, линии поведения, производительность и интерфейсы.

Разработка проекта — разрабатывается и формулируется логически последовательная техническая характеристика программной системы, включая структуры данных, архитектуру ПО, интерфейсные представления и процессуальную (алгоритмическую) детализацию.

2.1.1. Каскадная модель

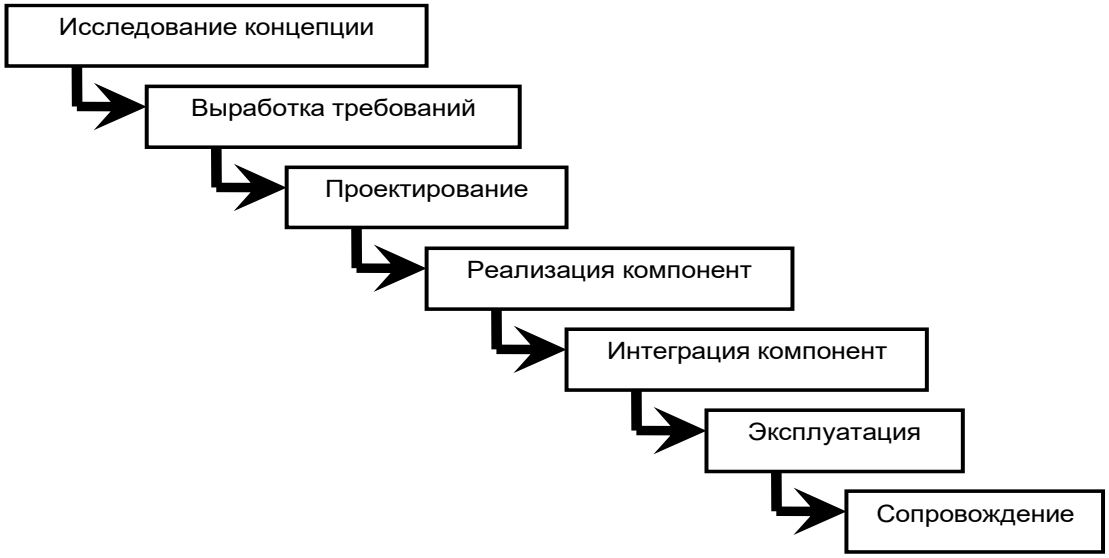


Рисунок 2.2 - Каскадная или водопадная модель

Реализация — эскизное описание ПО превращается в полноценный программный продукт. Результат: исходный код, база данных и документация.

В реализации обычно выделяют два этапа: реализацию компонент ПО и интеграцию компонент в готовый продукт. На обоих этапах выполняется кодирование и тестирование, которые тоже иногда рассматривают как два подэтапа.

Эксплуатация и поддержка - подразумевает запуск и текущее обеспечение, включая предоставление технической помощи, обсуждение возникших вопросов с пользователем, регистрацию запросов пользователя на модернизацию и внесение изменений, а также корректирование или устранение ошибок.

Сопровождение — устранение программных ошибок, неисправностей, сбоев, модернизация и внесение изменений. Состоит из итераций разработки.

Основными принципами каскадной модели являются:

1. Строго последовательное выполнение фаз.
2. Каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы.
3. Каждая фаза имеет определенные критерии входа и выхода: входные и выходные данные.
4. Каждая фаза полностью документируется.
5. Переход от одной фазы к другой осуществляется

посредством формального обзора с участием заказчика.

6. Основа модели – сформулированные требования – техническое задание (ТЗ), которые меняться не должны.

7. Критерий качества результата – соответствие продукта установленным требованиям.

Каскадная модель имеет следующие преимущества:

–проста и понятна заказчикам, т.к. часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО;

–проста и удобна в применении;

–процесс разработки выполняется поэтапно;

–ее структурой может руководствоваться даже слабо подготовленный или неопытный в техническом плане персонал;

–она способствует осуществлению строгого контроля менеджмента проекта;

–каждую стадию могут выполнять независимые команды (все документировано);

–позволяет достаточно точно планировать сроки и затраты.

При использовании каскадной модели для «неподходящего» проекта могут проявляться следующие ее основные недостатки:

–попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо проблему или недостаток, приведет к значительному увеличению затрат и сбою в графике;

–интеграция компонент, на которой обычно выявляется большая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость устранения ошибок;

–запаздывание с получением результатов – если в процессе выполнения проекта требования изменились, то получится устаревший результат;

–недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования, которые также могут меняться в процессе выполнения проекта. В этом случае разработка ПО имеет принципиально циклический характер.

Каскадная модель впервые четко сформулирована в 1970 году. На начальном периоде она сыграла ведущую роль как метод регулярной разработки сложного ПО.

В семидесятых-восьмидесятых годах XX века модель была

принята как стандарт министерства обороны США. Со временем недостатки каскадной модели стали проявляться все чаще и возникло мнение, что она безнадежно устарела.

Каскадная модель не утратила своей актуальности при решении следующих типов задач, у которых требования и их реализация максимально четко определены и понятны, используется неизменяемое определение продукта и вполне понятные технические методики. Это задачи типа:

- научно-вычислительного характера (пакеты и библиотеки научных программ типа расчета несущих конструкций зданий, мостов и иных сооружений);
- операционные системы и компиляторы;
- системы реального времени управления конкретными объектами.

Кроме того, каскадная модель применима в следующих условиях:

- повторная разработка типового продукта (автоматизированного бухгалтерского учета, начисления зарплаты);
- выпуск новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы (перенос уже существующего продукта на новую платформу)

Принципы каскадной модели находят применение как элементы моделей других типов.

2.1.2. Спиральная (циклическая) модель

На практике, при решении достаточно большого количества задач разработка ПО имеет циклический характер, когда после выполнения некоторых стадий приходится возвращаться на предыдущие. Можно указать две основные причины таких возвратов:

- ошибки разработчиков, допущенные на ранних стадиях и выявленные на поздних стадиях – ошибки анализа, проектирования, кодирования, выявляемые, как правило, на стадии тестирования.
- изменение требований в процессе разработки («ошибки» заказчиков). Это или неготовность заказчиков сформулировать требования («Сказать, что должна делать программа я смогу

только после того, как увижу, как она работает»), или изменения требований, вызванные изменениями ситуации в процессе разработки (изменения рынка, новые технологии).

Спиральная (циклическая) модель – процесс также разбивается на стадии, но стадии выполняются циклическим повторением. На первом цикле создается прототип продукта, выполняющий часть требований. Дальнейшие циклы связаны с наращиванием прототипа до полного удовлетворения требований.

Спиральная модель была предложена как альтернатива каскадной модели, учитывающая повторяющийся характер разработки ПО.

Основные принципы спиральной модели можно сформулировать следующим образом:

- разработка вариантов продукта, соответствующих различным вариантам требований с возможностью вернуться к более ранним вариантам;
- создание прототипов ПО как средства общения с заказчиком для уточнения и выявления требований;
- планирование следующих вариантов с оценкой альтернатив и анализом рисков, связанных с переходом к следующему варианту;
- переход к разработке следующего варианта до завершения предыдущего в случае, когда риск завершения очередного варианта (прототипа) становится неоправданно высок;
- использование каскадной модели как схемы разработки очередного варианта;
- активное привлечение заказчика к работе над проектом - заказчик участвует в оценке очередного прототипа ПО, уточнении требований при переходе к следующему, оценке предложенных альтернатив очередного варианта и оценке рисков.

Разработка вариантов продукта представляется как набор циклов раскручивающейся спирали. Каждому циклу спирали соответствует такое же количество стадий, как и в модели каскадного процесса. При этом начальные стадии связанные с анализом и планированием, представлены более подробно с добавлением новых элементов (рисунок 2.3).

Рисунок 2.3 - Спиральная модель

В каждом цикле выделяются четыре базовые фазы:

- определение целей, альтернативных вариантов и ограничений;
- оценка альтернативных вариантов, идентификация и разрешение рисков;
- разработка продукта следующего уровня;
- планирование следующей фазы.

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку ПО. Здесь на первой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативы подходов к решению задачи. Далее проводится оценка подходов, устанавливаются их риски. На шаге разработки создается концепция (видение) продукта и путей его создания.

Следующий цикл начинается с планирования требований и деталей ЖЦ продукта для оценки затрат. На фазе определения целей устанавливаются альтернативные варианты требований,

связанные с ранжированием требований по важности и стоимости их выполнения. На фазе оценки устанавливаются риски вариантов требований. На фазе разработки – спецификация требований (с указанием рисков и стоимости), готовится демо-версия ПО для анализа требований заказчиком.

Следующий цикл – разработка проекта – начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков. На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демо-версия, отражающая основные проектные решения.

Следующий цикл – реализация ПО – также начинается с планирования. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков на этом цикле определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом – действующим вариантном (прототипом) продукта.

Отметим некоторые особенности спиральной модели:

– до начала разработки ПО есть несколько полных циклов анализа требований и проектирования;

– количество циклов модели (как в части анализа и проектирования, так и в части реализации) не ограничено и определяется сложностью и объемом задачи;

– в модели предполагаются возвраты на оставленные варианты при изменении стоимости рисков.

Спиральная модель (по отношению к каскадной) имеет следующие очевидные *преимущества*:

– более тщательное проектирование (несколько начальных итераций) с оценкой результатов проектирования, что позволяет выявить ошибки проектирования на более ранних стадиях;

– поэтапное уточнение требований в процессе выполнения

итераций, что позволяет более точно удовлетворить требованиям заказчика;

–участие заказчика в выполнении проекта с использованием прототипов программы. Заказчик видит, что и как создается, не выдвигает необоснованных требований, оценивает реальные объемы финансирования;

–планирование и управление рисками при переходе на следующие итерации позволяет разумно планировать использование ресурсов и обосновывать финансирование работ;

–возможность разработки сложного проекта «по частям», выделяя на первых этапах наиболее значимые требования.

Основные *недостатки* спиральной модели связаны с ее сложностью:

–сложность анализа и оценки рисков при выборе вариантов;

–сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий);

–сложность оценки точки перехода на следующий цикл;

–бесконечность модели – на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Спиральную модель целесообразно применять при следующих условиях:

–когда пользователи не уверены в своих потребностях или, когда требования слишком сложны и могут меняться в процессе выполнения проекта и необходимо прототипирование для анализа и оценки требований;

–когда достижение успеха не гарантировано и необходима оценка рисков продолжения проекта;

–когда проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения;

–когда речь идет о применении новых технологий, что связано с риском их освоения и достижения ожидаемого результата;

–при выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

2.1.3. Другие типы моделей ЖЦ ПО

Каскадная и спиральная модели устанавливают некоторые принципы организации жизненного цикла создания программного продукта. Каждая из них имеет преимущества, недостатки и области применимости. Каскадная модель проста, но применима в случае, когда требования известны и меняться не будут. Спиральная модель учитывает такие важные показатели проекта как изменяемость требований, невозможность оценить заранее объем финансирования, риски выполнения проекта. Но спиральная модель сложна и требует больших затрат на сопровождение.

Существуют другие типы моделей, которое можно рассматривать как «промежуточные» между каскадной и спиральной моделями. Эти модели используют отдельные преимущества каскадной и спиральной моделей и позволяют достичь успеха для определенных типов задач.

Итерационная модель. Итерационная модель жизненного цикла является развитием классической каскадной модели и предполагает возможность возвратов на ранее выполненные этапы. При этом, причинами возвратов в классической итерационной модели являются выявленные ошибки, устранение которых и требует возврата на предыдущие этапы в зависимости от типа (природы) ошибки – ошибки кодирования, проектирования, спецификации или определения требований.

Реально итерационная модель является более жизненной, чем классическая (строгая) каскадная модель, т.к. создание ПО всегда связано с устранением ошибок. Следует отметить, что уже в первой статье, посвященной каскадной модели, Боэм отмечал это обстоятельство и описал итерационный вариант каскадной модели. Практически все применяемые модели жизненного цикла имеют итерационный характер, но цели итераций могут быть разными (рисунок 2.4).

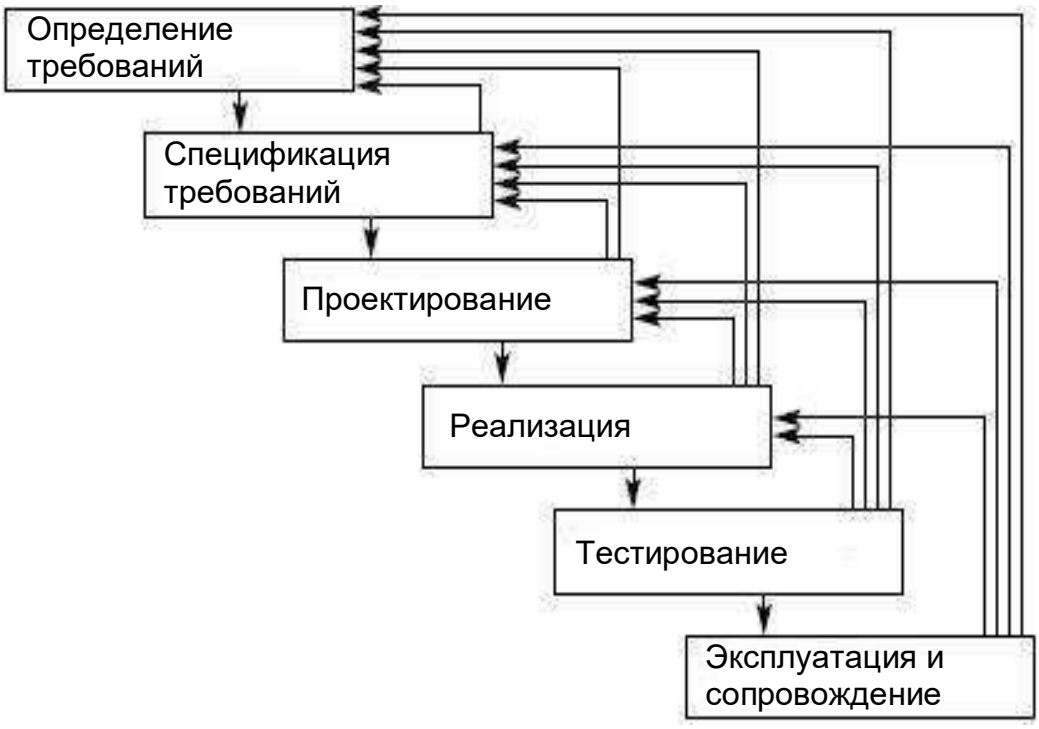


Рисунок 2.4 - Итерационная модель

V-образная модель. V-образная модель была создана как итерационная разновидность каскадной модели. Целями итераций в этой модели является обеспечение процесса тестирования.

Тестирование продукта обсуждается, проектируется и планируется на ранних этапах жизненного цикла разработки. План испытания и приемки заказчиком разрабатывается на этапе планирования, а компоновочного испытания системы - на фазах анализа, разработки проекта и т.д.

Этот процесс разработки планов испытания обозначен пунктирной линией между прямоугольниками V-образной модели. Помимо планов, на ранних этапах разрабатываются также и тесты, которые будут выполняться при завершении параллельных этапов (рисунок 2.5).

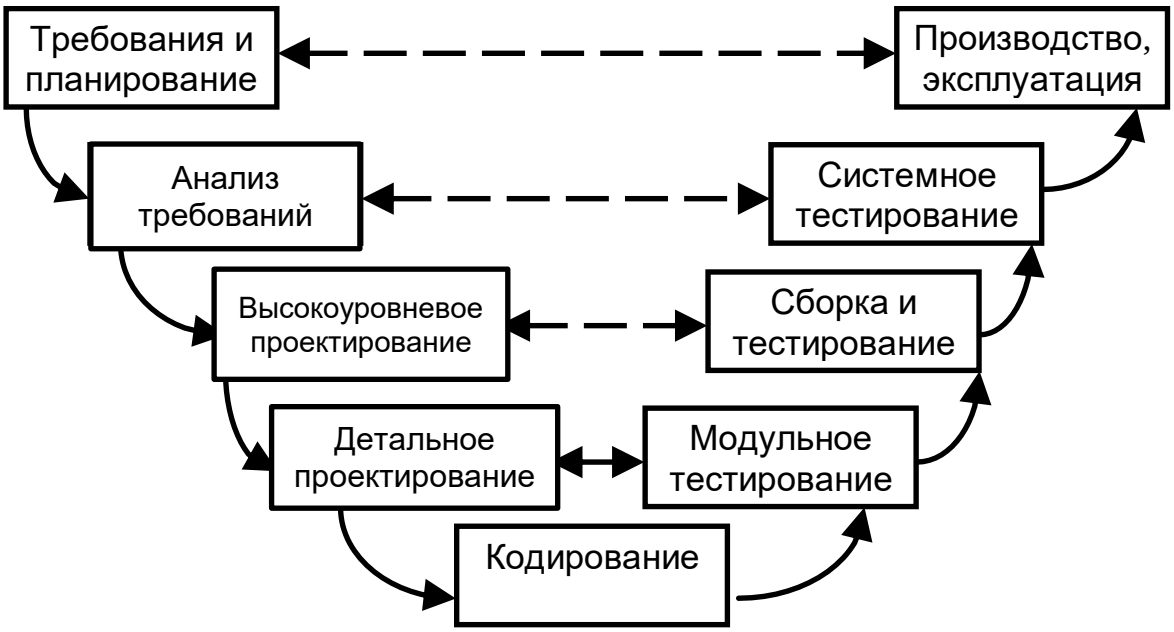


Рисунок 2.5 - V-образная модель

Инкрементная (пошаговая) модель (рисунок 2.6). Инкрементная разработка представляет собой процесс поэтапной реализации всей системы и поэтапного наращивания (приращения) функциональных возможностей. На первом шаге необходим полный заранее сформулированный набор требований, которые делятся по некоторому признаку на части. Далее выбирается первая группа требований и выполняется полный проход по каскадной модели. После того, как первый вариант системы, выполняющий первую группу требований сдан заказчику, разработчики переходят к следующему шагу (второму инкременту) по разработке варианта, выполняющего вторую группу требований.

Особенностью инкрементной модели является разработка приемочных тестов на этапе анализа требований, что упрощает приемку варианта заказчиком и устанавливает четкие цели разработки очередного варианта системы.

Инкрементная модель особенно эффективна в случае, когда задача разбивается на несколько относительно независимых подзадач (разработка подсистем «Зарплата», «Бухгалтерия», «Склад», «Поставщики»). Кроме того, инкрементная модель для внутренней итерации может использовать не только каскадную, но и другие типы моделей.

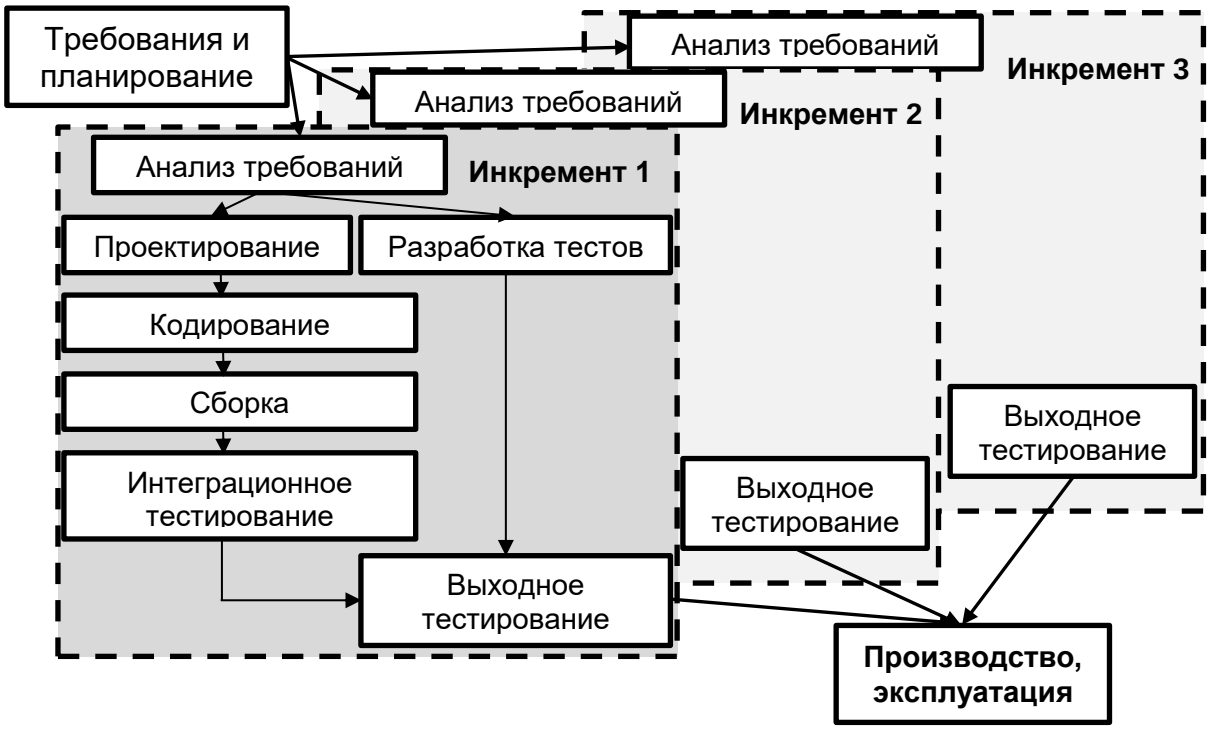


Рисунок 2.6 - Инкрементная (пошаговая) модель

Модель быстрого прототипирования. Модель быстрого протитипирования предназначена для быстрого создания прототипов продукта с целью уточнения требований и поэтапного развития прототипов в конечный продукт. Скорость выполнения проекта обеспечивается планированием разработки прототипов и участием заказчика в процессе разработки.

Начало жизненного цикла разработки помещено в центре эллипса. Совместно с пользователем разрабатывается предварительный план проекта на основе предварительных требований. Результат начального планирования - документ, описывающий в общих чертах примерные графики и результативные данные.

Следующий уровень – создание исходного прототипа на основе быстрого анализа, проекта база данных, пользовательского интерфейса и некоторых функций. Затем начинается итерационный цикл быстрого прототипирования (рисунок 2.7).

Разработчик проекта демонстрирует очередной прототип, а пользователь оценивает его функционирование, затем совместно определяются проблемы и пути их преодоления для перехода к следующему прототипу. Этот процесс продолжается до тех пор, пока пользователь не согласится, что очередной прототип в точности отображает все требования.

Рисунок 2.7 - Модель быстрого прототипирования

Получив одобрение пользователя, быстрый прототип преобразуют в детальный проект, и систему настраивают на производственное использование. Именно на этом этапе настройки ускоренный прототип становится полностью действующей системой.

При разработке производственной версии программы, может понадобиться более высокий уровень функциональных возможностей, различные системные ресурсы, необходимых для обеспечения полной рабочей нагрузки, или ограничения во времени. После этого следуют тестирование в предельных режимах, определение измерительных критериев и настройка, а затем, как обычно, функциональное сопровождение.

2.2. Промышленные технологии создания программных продуктов

В настоящее время широкое применение получают так называемые промышленные технологии создания программного продукта. Эти технологии были разработаны фирмами, накопившими большой опыт создания ПО. Технологии представлены описаниями принципов, методов, применяемых процессов и операций. Такие технологии, как правило, поддерживаются набором CASE-средств, охватывают все этапы жизненного цикла продукта и успешно применяются для решения

практических задач.

Рассмотрим особенности моделей жизненного цикла наиболее известных промышленных технологий:

- Microsoft Solution Framework (MSF) – разработка фирмы Microsoft, предназначенная для решения широкого круга задач. Технология масштабируема, т.е. настраиваема на решение задач любой сложности коллективом любой численности.

- Rational Unified Process (RUP) – разработка фирмы Rational, долгое время успешно занимавшейся созданием CASE-средств, применяемых на различных этапах жизненного цикла продукта от анализа до тестирования и документирования. Аналогично MSF, RUP универсальна, масштабируема и настраиваема на применение в конкретных условиях.

- Extreme Programming (XP) – активно развивающаяся в последнее время технология, предназначенная для решения относительно небольших задач, относительно небольшими коллективами профессиональных разработчиков в условиях жестко ограниченного времени.

Каждая из этих технологий имеет свои особенности организации модели жизненного цикла создания продукта.

Adaptive Software Development (ASD) - одна из новых методологий, которые появились как альтернатива традиционным, ориентированным на процесс, методам управления разработкой ПО. Во главу угла в ней ставится человеческий фактор, результаты работы и минимизация самого процесса при максимальном увеличении взаимодействия между людьми. Методология была разработана исходя из объективных реалий современного высокотехнологичного бизнеса, который отличается огромной скоростью развития и высокой изменчивостью.

Практики ASD базируются на принципе непрерывной адаптации, благодаря которой возникает другая философия и другой жизненный цикл проекта, когда постоянные изменения становятся нормой.

Семейство методологий Crystal. Семейство методологий, разработанное Алистером Коберном, который отличается оригинальными взглядами на создание ПО.

Коберн рассматривает процесс создания ПО как конечную целенаправленную игру и утверждает, что у этой игры есть всего

две цели: главная и вспомогательная.

Главная цель заключается в том, чтобы успешно закончить проект, то есть создать работающий продукт. Второстепенная цель - подготовиться к следующей игре. Для достижения этой цели может понадобиться документация, качественное написание кода, то есть все, что облегчает дальнейшее развитие и поддержку продукта.

Если в процессе разработки достигнуты обе цели, то проект считается успешным. Помимо конечного продукта, всегда создаются вспомогательные промежуточные продукты: модели, схемы, описания и т.п. Логично, что их должно быть ровно столько, сколько необходимо для достижения конечной цели.

Проблема в том, что очень сложно заранее предсказать, какие промежуточные продукты нужны, а какие - нет.

Методология SCRUM. Как и в прочих гибких методологиях, здесь подчеркивается, что определенный и повторяющийся процесс годится только для решения определенных и повторяющихся проблем в определенной и повторяющейся среде, в которой работают определенные и повторяющиеся разработчики.

Согласно методу SCRUM, проект делится на итерации (которые здесь называются "спринт"), по 30 дней каждая. Перед началом спринта вы определяете функциональность, которая требуется на данном этапе, после чего уступаете место команде разработчиков, которые выполняют поставленную вами задачу. Главное в том, чтобы в течение одного спринта требования оставались неизменными.

Посвященные методологии SCRUM работы, главное основное внимание уделяют итеративному планированию и процессу отслеживания. В целом же SCRUM очень близок к другим гибким методологиям, и должен хорошо сочетаться с правилами кодирования XP.

2.2.1. Модель Microsoft Solution Framework

Одна из особенностей технологии MSF состоит в том, что она ориентирована не просто на создание программного продукта, удовлетворяющего перечисленным требованиям, а на поиск решения проблем, стоящих перед заказчиком. Различие состоит в

том, что перечисляемые заказчиком требования являются проявлениями некоторых более глубоких проблем и неточность, неполнота, изменение требований в процессе разработки – следствие недопонимания проблем. Поэтому, в технологии MSF большое внимание уделяется анализу проблем заказчика и разработке вариантов системы для поиска решения этих проблем.

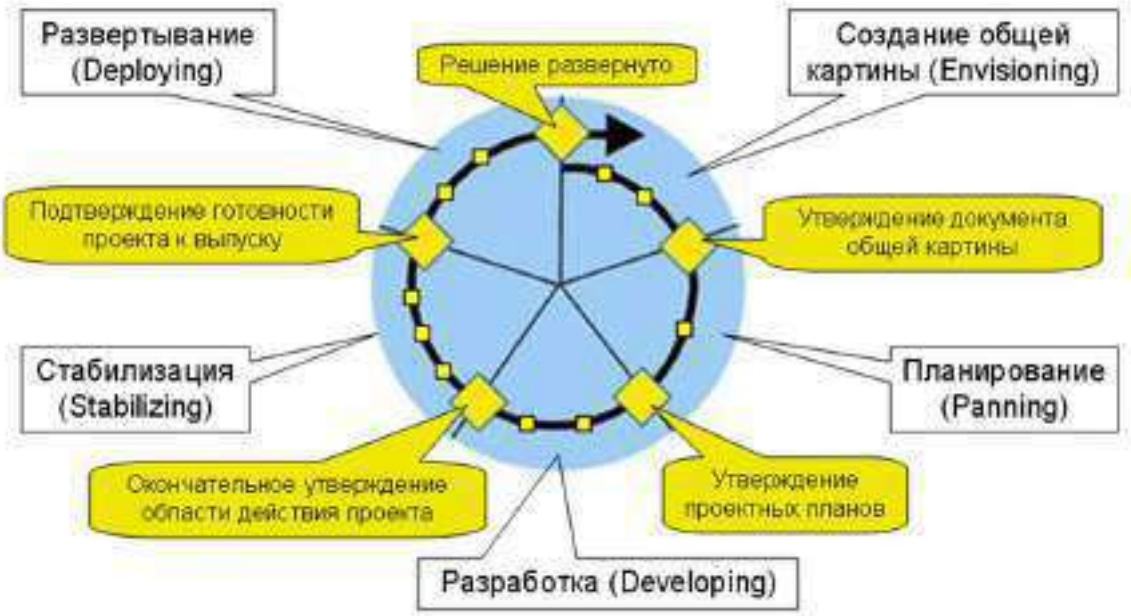


Рисунок 2.8 - Модель жизненного цикла MSF

Модель жизненного цикла MSF является некоторым гибридом каскадной и спиральной моделей, сочетая простоту управления каскадной модели с гибкостью спиральной (рисунок 2.8).

Модель жизненного цикла MSF ориентирована на “вехи” (milestones) – ключевые точки проекта, характеризующие достижение какого-либо существенного результата. Этот результат может быть оценен и проанализирован, что подразумевает ответ на вопрос: “А достигли ли мы целей, поставленных на этом шаге?». В модели предусматривается наличие основных вех (завершение главных фаз модели) и промежуточных, отражающих внутренние этапы главных фаз.

Примеры CASE-средств.

Основными фазами модели MSF являются:

1. Создание общей картины приложения (Envisioning). На этом этапе решаются следующие основные задачи: оценка существующей ситуации; определение состава команды,

структуры проекта, бизнес-целей, требований и профилей пользователей; разработка концепции решения и оценка риска. Устанавливаются две промежуточные вехи: "Организован костяк команды" и "Создана общая картина решения".

2. Планирование (Planning). Включает планирование и проектирование продукта. На основе анализа требований разрабатывается проект и основные архитектурные решения, функциональные спецификации системы, планы и календарные графики, среды разработки, тестирования и пилотной эксплуатации.

Этап проектирования продукта состоит из трех стадий: концептуального, логического и физического проектирования.

На стадии концептуального проектирования задача рассматривается с точки зрения пользовательских и бизнес-требований и заканчивается определением набора сценариев использования системы.

При логическом проектировании задача рассматривается с точки зрения проектной команды, решение представляется в виде набора сервисов.

На стадии физического проектирования задача рассматривается с точки зрения программистов, уточняются используемые технологии и интерфейсы.

3. Разработка (Developing). Создается вариант решения проблемы, в виде кода и документации очередного прототипа, включая спецификации и сценарии тестирования. Основная веха этапа - "Окончательное утверждение области действия проекта". Продукт готов к внешнему тестированию и стабилизации. Кроме того, заказчики, пользователи, сотрудники службы поддержки и сопровождения, а также ключевые участники проекта могут предварительно оценить продукт и указать все недостатки, которые нужно устранить до его поставки.

4. Стабилизация (Stabilizing). Подготовка к выпуску окончательной версии продукта, доводка его до заданного уровня качества. Здесь выполняется комплекс работ по тестированию (обнаружение и устранение дефектов), проверяется сценарий развертывания продукта. Когда решение становится достаточно устойчивым, проводится его пилотная эксплуатация в тестовой среде с привлечением пользователей и применением реальных

сценариев работы.

5. Развертывание (Deploying). Выполняется установка решения и необходимых компонентов окружения, проводится его стабилизация в промышленных условиях и передача проекта в руки группы сопровождения. Кроме того, анализируется проект в целом на предмет уровня удовлетворенности заказчика.

2.2.2. Модель Rational Unified Process

Модель жизненного цикла RUP является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели.

В модели RUP выделяются 4 основные фазы, 9 видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта (рисунок 2.9).

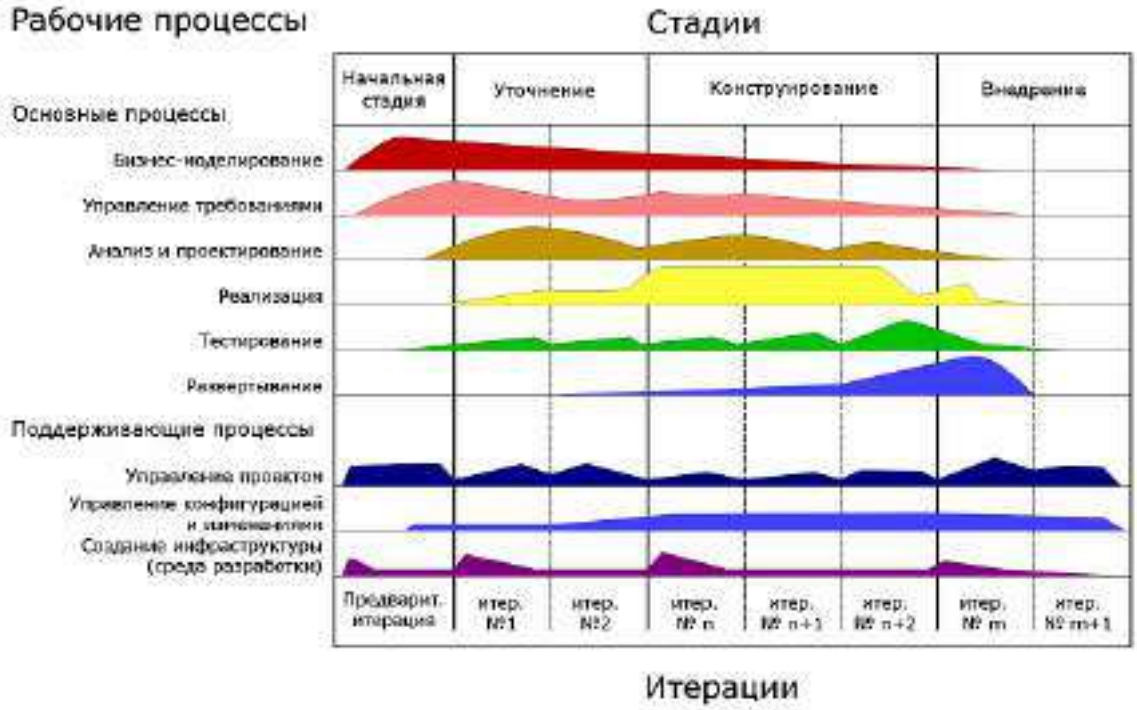


Рисунок 2.9 - Модель жизненного цикла RUP

RUP ориентирован на поэтапное моделирование создаваемого продукта с помощью языка UML.

Основными фазами RUP являются:

1. Фаза начала проекта (Inception). Определяются основные цели проекта, бюджет проекта, основные средства его выполнения - технологии, инструменты, ключевой персонал, составляются

предварительные планы проекта. Основная цель этой фазы - достичь компромисса между всеми заинтересованными лицами относительно задач проекта.

2. Фаза проработки (уточнение) (Elaboration). Основная цель этой фазы - на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы.

3. Фаза построения (Construction). Основная цель этой фазы - детальное уяснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.

4. Фаза передачи (внедрение) (Transition). Цель фазы - сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

В рамках каждой фазы возможно проведение нескольких итераций, количество которых определяется сложностью выполняемого проекта.

Деятельности (основные процессы) RUP делятся на шесть рабочих и три поддерживающие.

К рабочим видам деятельности относятся:

Моделирование предметной области (бизнес-моделирование, Business Modeling). Цели этой деятельности - понять бизнес-контекст, в котором должна будет работать система (и убедиться, что все заинтересованные лица понимают его одинаково), понять возможные проблемы, оценить возможные их решения и их последствия для бизнеса организации, в которой будет работать система.

Определение требований (Requirements). Цели - понять, что должна делать система, определить границы системы и основу для планирования проекта и оценок ресурсозатрат в нем.

Анализ и проектирование (Analysis and Design). Выработка архитектуры системы на основе ключевых требований, создание проектной модели, представленной в виде диаграмм UML, описывающих продукт с различных точек зрения.

Реализация (Implementation). Разработка исходного кода,

компонент системы, тестирование и интегрирование компонент.

Тестирование (Test). Общая оценка дефектов продукта, его качество в целом; оценка степени соответствия исходным требованиям.

Развертывание (Deployment). Цели - развернуть систему в ее рабочем окружении и оценить ее работоспособность.

Поддерживающими деятельностями являются:

Управление проектом (Project Management). Включает планирование, управление персоналом, обеспечения связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.

Управление конфигурациями и изменениями (Configuration and Change Management). Определение элементов, подлежащих хранению и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.

Создание инфраструктуры (управление средой проекта) (Environment). Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в проекте.

2.2.3. Модель Extreme Programming

Модель Extreme Programming (Экстремальное программирование) является примером так называемого метода «живой» разработки.

Модель жизненного цикла XP является итерационно-инкрементной моделью быстрого создания (и модификации) протопопов продукта, удовлетворяющих очередному требованию (user story) (рисунок 2.10).

Основными фазами модели можно считать следующие фазы.

«Вброс» архитектуры – начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.

Истории использования (User Story) – этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций.

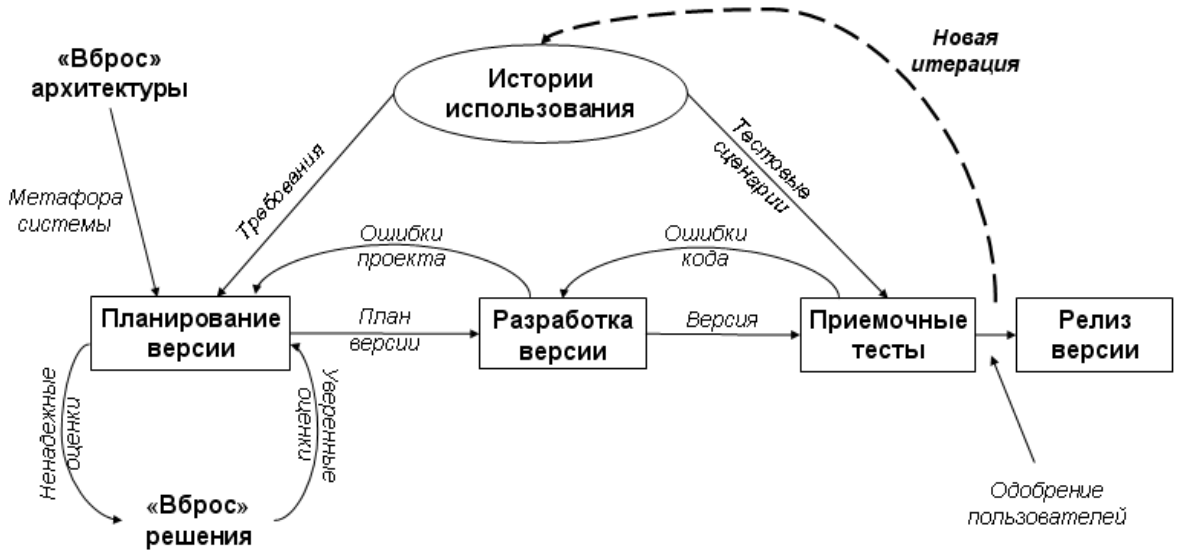


Рисунок 2.10 - Модель жизненного цикла XP

Истории использования являются требованиями для планирования очередной версии и одновременной разработки приемочных тестов (Acceptance tests) для ее проверки.

Планирование версии (релиза). Проводится на собрании с участием заказчика путем выбора User Stories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования - получение оценок того, что и как можно сделать за 1-3 недели создания следующей версии продукта.

Разработка проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.

Тестирование проводится с участием заказчика, который участвует в составлении тестов.

Выпуск релиза – разработанная версия передается заказчику для использования или бета-тестирования.

По завершению цикла делается переход на следующую итерацию разработки.

Принципы Extreme Programming.

Особенности модели жизненного цикла XP проясняют следующие принципы этого метода. Прежде всего, это принципы «живой» разработки ПО, зафиксированные в манифесте «живой» разработки:

1. Люди и их общение более важны, чем процессы и

инструменты.

2. Работающая программа более важна, чем исчерпывающая документация.

3. Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта.

4. Отработка изменений более важна, чем следование планам.

Кроме того, в XP есть несколько правил (техник), характеризующих особенности модели его жизненного цикла:

1. Живое планирование (planning game) - как можно быстрее определить объем работ, который нужно сделать до следующей версии ПО. Решение принимается на основе, в первую очередь, бизнес-приоритетов заказчика и, во-вторую, технических оценок. Планы изменяются, как только они начинают расходится с действительностью или пожеланиями заказчика.

2. Частая смена версий (small releases) - первая работающая версия должна появиться как можно быстрее, и тут же должна начать использоваться. Следующие версии подготавливаются через достаточно короткие промежутки времени.

3. Простые проектные решения (simple design) - в каждый момент времени система должна быть сконструирована так просто, насколько это возможно. Новые функции добавляются только после ясной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.

4. Разработка на основе тестирования (test-driven development) - сначала пишутся тесты, потом реализуются модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.

5. Постоянная переработка (refactoring) системы для устранения излишней сложности, увеличения понятности кода, повышения его гибкости. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат.

6. Программирование парами (pair programming) - весь код пишется двумя программистами на одном компьютере, что повышает его качество (отсутствие ошибок, понятность, читаемость).

7. Постоянная интеграция (continuous integration) - система

собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.

8. 40-часовая рабочая неделя - сверхурочная работа рассматривается как признак больших проблем в проекте. Не допускается сверхурочная работа 2 недели подряд — это истощает программистов и делает их работу значительно менее продуктивной.

2.2.4. Методология SCRUM

Одна из проблем управления программными проектами состоит в том, что значительное число требований изменяется в процессе разработки.

В традиционных подходах к разработке план создается в самом начале процесса, и команда старается реализовать этот план в заданное время и с заданным бюджетом. Однако недавнее исследование показало, что около 65% требований к программному продукту изменяется в ходе работы над ним. Это означает, что 65% созданных функций никогда не используется, они не востребованы заказчиками, а на разработку продукта уходит почти в два раза больше времени, чем нужно, и обходится она вдвое дороже.

В больших проектах с традиционными методами управления формируется специальный совет для контроля за изменениями, который, фактически, предназначен для того, чтобы помешать заказчикам получить то, что им действительно нужно.

SCRUM (по-русски читается СКРАМ) - методология управления процессом для гибкой разработки программного обеспечения.

В центре внимания методики Scrum - обеспечение максимальной ценности для бизнеса заказчика.

Заказчики активно вовлекаются в процесс инспектирования хода разработки, регулярно проверяют работающий код и имеют возможность менять требования по мере необходимости, так чтобы программный продукт реализовывал действительно нужные им функции.

Когда заказчик понимает, что уже не осталось ничего, что надо сделать непременно, проект может завершиться.

Для лучшего понимания методологии рассмотрим этапы создания программного продукта согласно методологии SCRUM (рисунок 2.11).

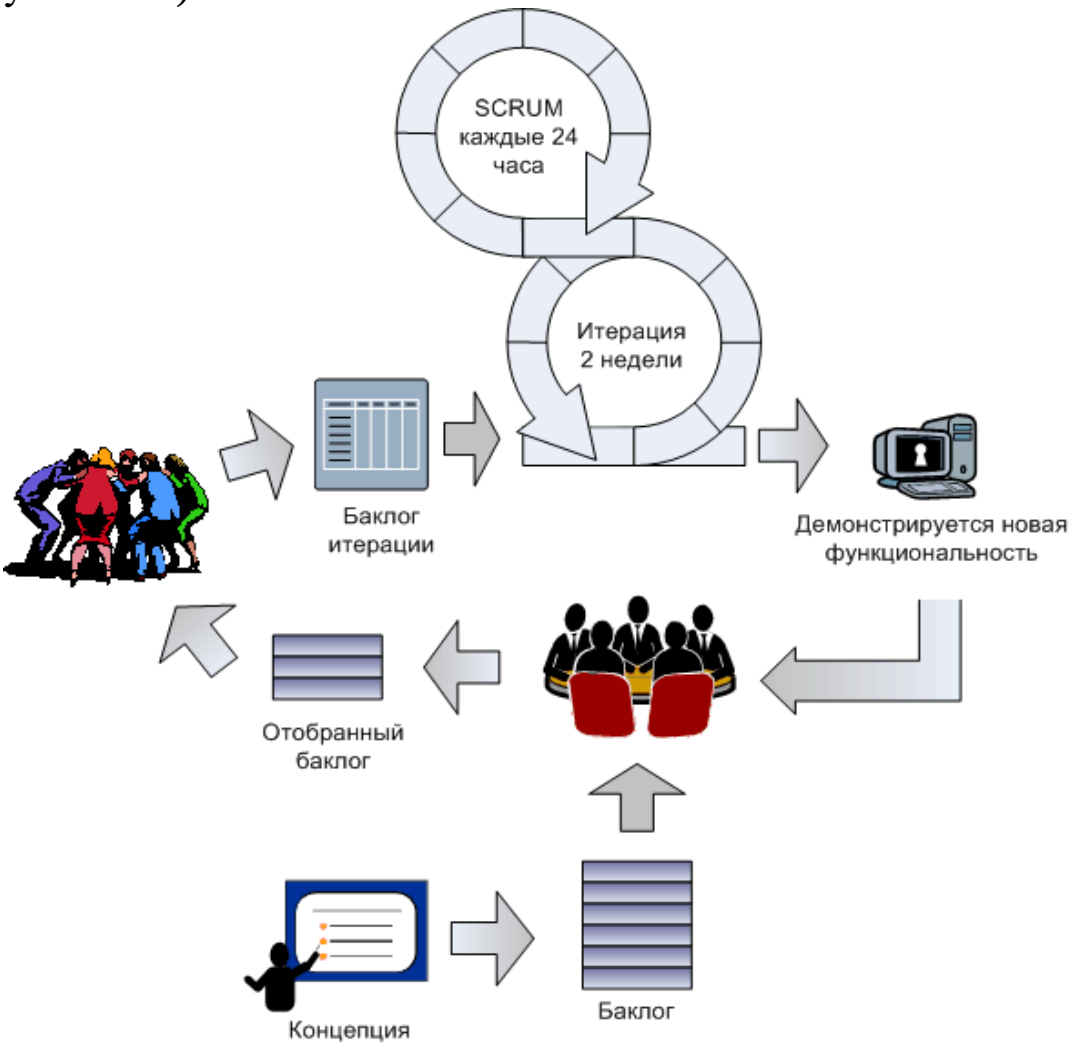


Рисунок 2.11 - Методология SCRUM

На встрече с клиентом определяется концепция и список требований к функциональности продукта. При этом все требования описаны на понятном для заказчика языке.

На основании этого Product Owner - человек, который представляет интересы заказчика, составляет Product Backlog - список требований к функциональности, упорядоченный по степени важности так, чтобы в первую очередь были реализованы наиболее ценные для бизнеса заказчика возможности.

Product Backlog должен быть ориентирован на бизнес, т.е. на то "что надо", а не на то "как сделать".

Затем список обсуждается с командой (Scrum Team) для

оценки объема работ.

В результате у каждого элемента Backlog'a должны быть заполнены следующие поля:

1. ID - уникальный идентификатор - порядковый номер.

Название – краткое описание. Должно быть однозначным, чтобы разработчики и product owner могли примерно понять, о чем идет речь, и отличить один элемент Backlog'a от другого. Обычно от 2 до 10 слов.

2. Важность (Importance) - степень важности, по мнению product owner'a. Например, 10. Или 150. Чем больше значение, тем выше важность. Предварительная оценка (initial estimate) - начальная оценка объема работ, необходимого для реализации элемента Backlog'a. Приблизительно соответствует числу "идеальных человеко-дней". Как продемонстрировать (how to demo) - краткое пояснение того, как завершённый элемент Backlog'a будет продемонстрирована в конце спринта. По сути, это простой тестовый сценарий типа "Сделайте это, сделайте то - должно получиться то-то".

3. Примечания - любая другая информация: пояснения, ссылки на дополнительные источники информации, и т.д.

Официально документ принадлежит product owner’у, однако другим членам команды разрешается его редактировать.

У каждого продукта должен быть только один product backlog и только один product owner.

Все наиболее важные элементы Backlog'a должны быть классифицированы по уровню важности, а их числовые значения не должны совпадать.

Product owner должен понимать каждый элемент Backlog'a.

Примечание: Хотя заинтересованные лица могут добавлять требования в product backlog, они не имеют права присваивать им уровень важности. Это прерогатива product owner’a. Они также не могут добавлять оценки трудозатрат, поскольку это прерогатива команды.

Весь процесс разработки продукта в SCRUM разбит на итерации, называемые СПРИНТами. Продолжительность итерации 1 - 4 недели.

Перед каждой итерацией производится планирование спринта.

Задачи планирования спринта – это определение цели спринта, выбор требований из Product Backlog'a для перемещения их в Sprint backlog, определение даты демонстрации (или длины спринта), определение команды, реализующей спринт. По окончании спринта на выходе должна быть работающая программа, которую владелец продукта может попробовать своими руками.

Цель спринта должна отвечать на главный вопрос “Зачем команда работает над этим спринтом"?

Sprint backlog – это список верхних элементов элементов из Product Backlog'a, которые команда обязалась выполнить в течение спринта.

Дата «демо», это дата окончания спринта, дата демонстрации работающего функционала. Важно, чтобы и product owner, и команда совместными усилиями определили критерий готовности элемента Backlog'a.

Каждый элемент Sprint backlog'a может быть разбит на задачи. Разница между “задачами” и “элементами Backlog'a” в том, что элементы Backlog'a это нечто, что можно продемонстрировать, что представляет ценность для product owner’a, а задачи либо нельзя продемонстрировать, либо они не представляют ценности для product owner’a.

Также во время планирования спринта определяются время и место проведения ежедневных Scrum-митингов.

Scrum-митинг - ежедневная короткая встреча команды, где каждый участник разработки отчитывается, что он сделал вчера, что должен сделать сегодня, какие проблемы у него возникли. В ходе общей встречи ищутся способы разрешения этих проблем. Задача Scrum Master’a (лидера команды) — состоит в том, чтобы предоставить команде все возможности для реализации выбранных задач, но не диктовать, что должен делать каждый участник разработки.

Scrum Master контролирует продвижение работ. Эффективный инструмент для этого - доска задач. Доска задач состоит из трех колонок: "В планах", "В процессе", "Готово" (рисунок 2.12).

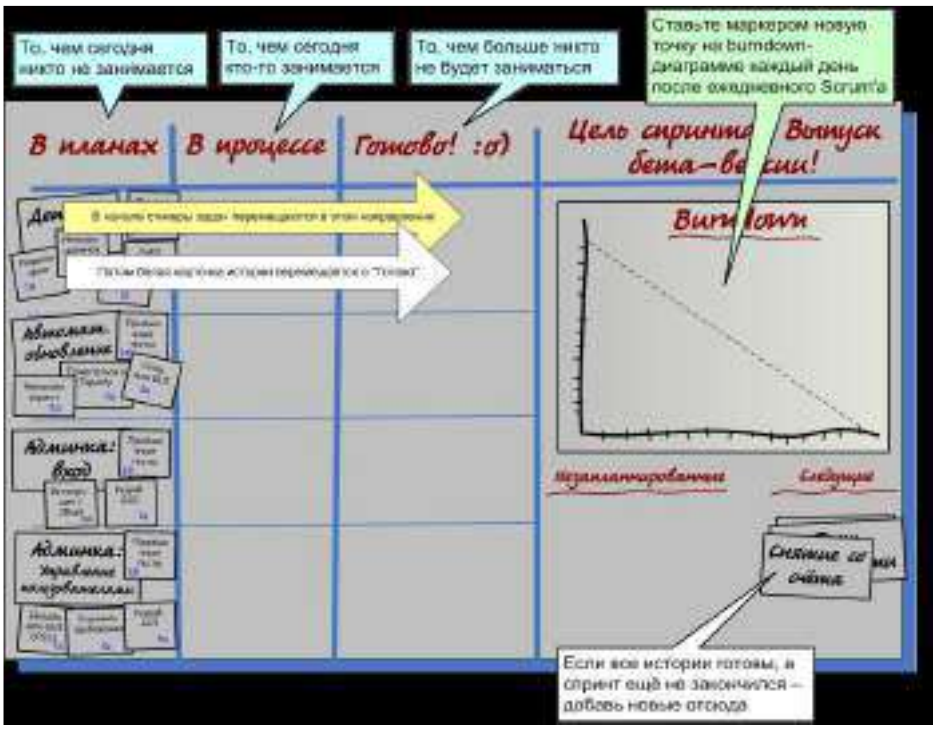


Рисунок 2.12 - Доска задач SCRUM

При старте спринта все задачи находятся в колонке "В планах".

Кроме доски задач Scrum Master создает burndown-диаграмму. Burndown-диаграмма создается следующим образом: по оси Y отмечается прогнозируемый объем работ в story-point'ax; по оси X даты до дня демонстрации. Выходные дни при этом лучше пропустить. После окончания каждого Scrum-митинга Scrum Master отмечает на burndown-диаграмме точку, в которой находится команда. Таким образом взглянув на диаграмму, всегда можно сказать где находится команда. При обнаружении отклонений Scrum Master принимает меры.

По завершению спринта проводится демонстрация продукта Product owner'y.

Через некоторое время после старта работ доска задач может выглядеть примерно так, как показано на рисунке 2.13.

По окончании спринта рекомендуется проводить собрание по ретроспективе разработки, с целью закрепления лучших практик и устранения препятствий.

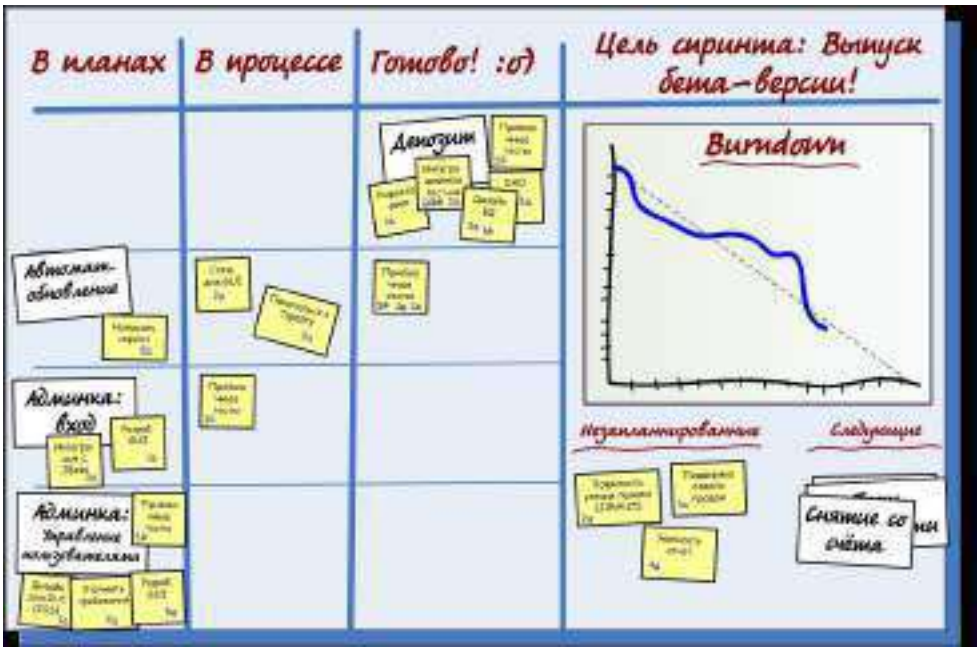


Рисунок 2.13 - Доска задач SCRUM

Далее производится планирование следующего спринта и процесс повторяется.

На определенном этапе оказывается, что все необходимые заказчику возможности реализованы, что является сигналом к окончанию проекта.

Команда в Scrum работает на принципах самоорганизации: сама оценивает, какие функции из общего списка она в состоянии реализовать в тридцатидневный срок, и формирует соответствующий список работ, тем самым всегда стараясь выбрать наиболее быстрый способ построения приложения.

Задача лидера команды — мастера Scrum — состоит в том, чтобы предоставить команде все возможности для реализации выбранных задач, но не диктовать, что должен делать каждый участник разработки.

Директивный стиль управления, при котором менеджер определяет задачи членов команды, часто замедляет процесс, мешает людям самим выбирать, как выполнить работу наилучшим образом.

С помощью специальных диаграмм мастер Scrum анализирует ход итерации, оценивает объем оставшихся задач и определяет текущий статус работ. Минимальная отчетность позволяет руководителю команды постоянно отслеживать, что происходит в проекте, обеспечивает полную прозрачность процесса.

Опросы американских ИТ-директоров, показали, насколько они осведомлены о состоянии проектов и ходе их выполнения. Как правило, ИТ-руководитель знает, что проект начался и что он с опозданием, но завершился. Оказалось, что 96% ИТ-директоров ничего не знают о промежуточном состоянии проекта, то есть фактически движутся вслепую. В Scrum руководитель постоянно наблюдает за состоянием проекта.

И наконец, цель Scrum, как и других «скорых» процессов - получить по завершении итерации работающую программу, которую владелец продукта может попробовать.

Scrum - первый подобный метод: он был разработан в 1993 году в компании Easel.

Платформа MSF для гибкой разработки программного обеспечения версии 5.0 основана на методологии Scrum. Поэтому команды могут внедрять процессы Scrum, используя средства, интегрированные о основными действиями разработки, в частности Visual Studio Application Lifecycle Management (ALM).

2.2.5. Adaptive Software Development (ASD) методология

Adaptive Software Development (ASD) - одна из новых методологий, которые появились как альтернатива традиционным, ориентированным на процесс методам управления разработкой ПО.

ASD, Extreme Programming (XP), Lean Development, SCRUM и семейство методологий Crystal, конечно, во многом отличаются друг от друга, однако у них всех есть одна общая черта - во главу угла в них ставится человеческий фактор, результаты работы и минимизация самого процесса при максимальном увеличении взаимодействия между людьми. Все эти методологии были разработаны исходя из объективных реалий современного высокотехнологичного бизнеса, который отличается огромной скоростью развития и высокой изменчивостью.

Практики ASD базируются на принципе непрерывной адаптации, благодаря которой возникает другая философия и другой жизненный цикл проекта, когда постоянные изменения становятся нормой.

В ASD обычный статический жизненный цикл Plan-Design-Build (Планирование - Проектирование - Конструирование) заменен на динамичный - Speculate-Collaborate-Learn (Обдумывание - Взаимодействие - Обучение), рисунок 2.14.

Рисунок 2.14 - Динамичная ASD-методология

Этот цикл ставит своей целью непрерывное обучение. Он связан с постоянными изменениями, повторными оценками, попытками предугадать неизвестное на текущий момент будущее проекта и требует тесного взаимодействия между разработчиками, тестировщиками и заказчиками.

Этот цикл не всегда представляет собой правильный круг. Даже при итеративном процессе можно иногда отклоняться в сторону, чтобы изучить неисследованные до сих пор области.

Методология ASD построена на концептуальной базе теории сложных адаптивных систем. Она рассчитана на использование в экстремальных проектах, в которых превалируют быстрый темп разработок, непредсказуемость и частые изменения.

Есть проекты, которые не могут считаться экстремальными, однако для всех остальных ASD подходит гораздо лучше, чем любой традиционный подход к разработке ПО.

У адаптивного жизненного цикла есть шесть базовых характеристик: Mission Focused (Целенаправленность), Component Based (Компонентный подход), Iterative (Итеративность), Timeboxed (Жесткие временные рамки), Risk Driven (Оценка с точки зрения рисков) и Change Tolerant (Допустимость

изменений).

В сфере электронного бизнеса, разработчики редко с самого начала четко понимают, какими должны быть результаты их работы. Тем не менее, они ясно осознают общую цель, и эта цель достаточно ясно формулируется команде разработчиков. Основные положения цели служат направляющими для первоначального исследования, а далее, в процессе работы над проектом, начинают сужаться.

Цель определяет, скорее, границы проекта, а не его конечный результат. Итеративный проект без правильно поставленной общей цели и постоянного пересмотра текущих задач становится "проектом-маятником" - разработки движутся туда-сюда, итерация за итерацией, без видимого прогресса. Артефакты, в которых зафиксирована цель всего проекта (их может быть несколько типов), не только помогают указать нужное направление работ, но и используются для того, чтобы в случае острой необходимости найти компромиссное решение. Если такие артефакты не помогают в принятии решений, то значит, они были созданы неправильно.

Адаптивный жизненный цикл проекта строится исходя из результатов, а не задач, причем в качестве результатов выступают компоненты системы.

В этом контексте под словом компонент понимается некий набор свойств программы (или некоторых элементов, входящих в поставку системы), который должен быть разработан в ходе итерации.

Конечно, документы (например, модель данных) тоже можно считать поставляемым компонентом, однако по отношению к свойству программы документация всегда второстепенна, поскольку только работающий программный код дает заказчику возможность ощутить реальные результаты работы.

В итеративном жизненном цикле "переделка" продукта занимает не меньше места, чем его "создание".

На производстве все обстоит по-другому. Там специальные программы, следящие за качеством работы станков, помогают избежать "брака" (результата ошибки в процессе изготовления какой-либо вещи) и последующей его переделки.

Однако в области программных (и прочих интеллектуальных)

разработок дело обстоит несколько по-иному, чем на конвейере, где бездушный автомат каждые несколько секунд штампует одинаковые детали.

Компоненты же, как правило, претерпевают изменения на протяжении нескольких итераций. Это происходит из-за того, что заказчики сообщают разработчикам свои комментарии относительно продукта по мере его развития.

Жесткие временные рамки, или установление фиксированных сроков поставки устанавливаются для каждого итеративного цикла. Говоря о жестких временных рамках, следует иметь в виду не столько время и сроки, сколько принятие сложных компромиссных решений.

Когда разработчика окружают постоянные изменения, нужно периодически задействовать некий фактор, который будет заставлять доводить начатое до конца. Кроме того, существование временных рамок заставляет команду разработчиков и клиентов постоянно пересматривать основные показатели проекта: его границы, расписание работ и поставок очередных версий, ресурсы, дефекты и т.д.

Те команды, которые не могут (или не хотят) принимать быстрые решения и делать выбор, никогда не добьются успеха в экстремальных условиях.

Как и в спиральной модели, планирование в адаптивном жизненном цикле ведется исходя из осознания и анализа критических рисков. Кроме того, адаптивный способ разработки ПО приемлет и приветствует все возникающие изменения. С адаптивной точки зрения, возможность внесения изменений - это конкурентное преимущество, а не просто еще одна "проблема".

Итеративный цикл "Speculate/Collaborate/Learn" ("Обдумывание/Взаимодействие/Обучение") годится для общего описания процесса, но для того, чтобы действительно производить на его основе программный продукт, нужно подробнее остановиться на некоторых деталях.

На рисунке 2.15 изображена базовая модель адаптивного процесса с его компонентами.

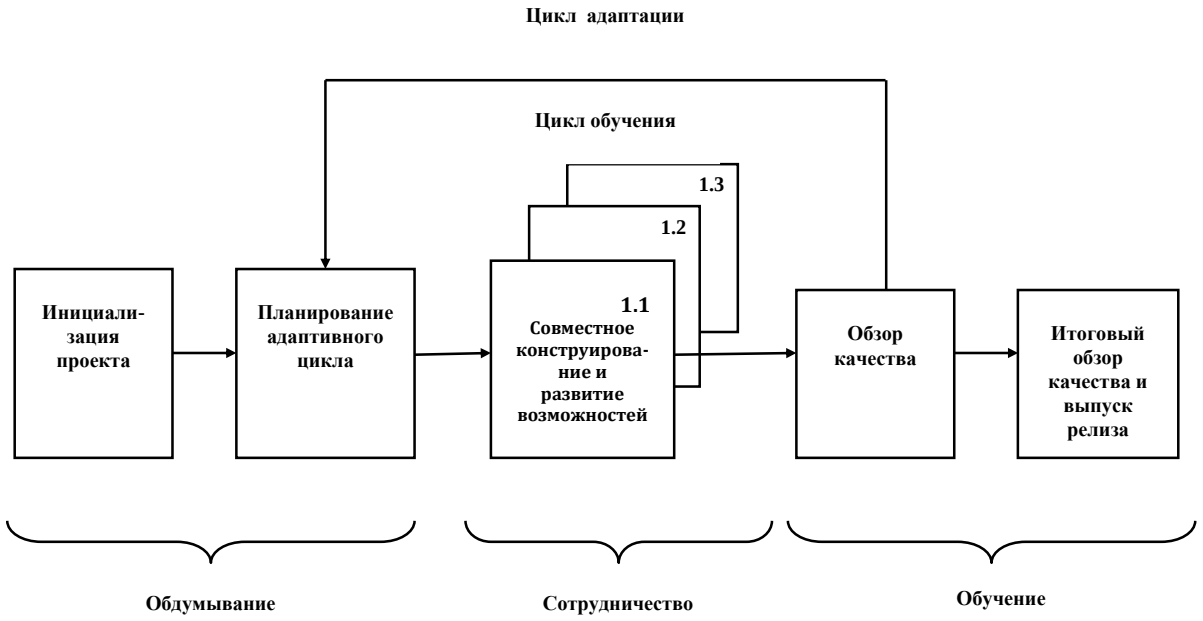


Рисунок 2.15 - Базовая модель адаптивного процесса

Необходимо остановиться более подробно на компонентах процесса.

Фаза «Обдумывание». Начальная фаза проекта и планирование циклов разработки состоит из семи последовательных шагов:

1. Выполнение начальной фазы проекта.
2. Определение временных рамок проекта.
3. Определение оптимального количества циклов и временных рамок каждого из них.
4. Расписывание основных задач по всем циклам.
5. Увязка разрабатываемые компоненты системы с соответствующими циклами.
6. Увязка с циклами технологических компонент и компонент поддержки.
7. Разработка списка задач по проекту.

В адаптивном процессе начальная фаза проекта немногим отличается от той, которая существует в любом нормальном процессе управления проектами. Она включает в себя определение целей и задач проекта, осмысление и документирование различных ограничений, изучение расстановки сил в проекте (то есть организацию самого проекта и определение ключевых фигур), выявление и краткое описание требований к системе, первоначальную оценку размера и масштабности проекта, а также

определение ключевых рисков. Поскольку одним из важнейших факторов, принимаемых на рассмотрение при использовании адаптивного подхода, является скорость разработки системы, то большая часть начальной информации по проекту должна собираться во время сессий - Совместная разработка программного продукта.

Для небольших проектов всю начальную фазу можно уложить в недельный срок. Для более крупных целесообразно запланировать так называемый "нулевой цикл", который позволит разработчикам провести более обширные предварительные исследования и подготовительные работы. ("Нулевой цикл" отличается от всех последующих тем, что его результатом будет не реальная часть системы, а некие подготовительные материалы).

Вторым шагом должно стать установление временных рамок для всего проекта. Эта оценка базируется на изначальном понимании масштаба проекта, требованиях к функциональности системы, оценках трудозатрат, ресурсах (иначе говоря, на материалах, которые собираются во время начальной фазы). Обдумывание не означает отказа от проведения оценок трудозатрат, просто необходимо понимать, что любые предварительные подсчеты могут оказаться неточными, и не нужно бояться их менять.

На третьем шаге необходимо решить, за сколько циклов можно создать всю систему. Кроме того, нужно задать каждому из циклов временные рамки. Для небольших и средних систем длина каждого цикла варьируется в пределах от четырех до восьми недель.

В некоторых проектах лучше всего использовать двухнедельные циклы, но есть и такие, для которых больше подойдут восьминедельные и даже более продолжительные циклы (впрочем, таких проектов совсем немного). Определять продолжительность циклов нужно исходя из общего графика разработки проекта и степени неопределенности намеченных задач.

Определив количество необходимых циклов и график выполнения каждого из них, команда переходит к четвертому шагу: определяет цели и задачи для каждого цикла разработки.

Четко поставленная цель должна быть не только у всего

проекта в целом, но и у каждого цикла в отдельности. Промежуточные контрольные точки в конце каждого цикла призваны обеспечить доступность и обозримость создаваемого программного продукта для участников проекта. Без этой доступности и обозримости невозможно увидеть и исправить различные дефекты и ошибки, без которых не обходится ни один проект.

В конце каждого цикла заказчик получает определенный набор компонентов системы, которые может (и должен) просмотреть и прокомментировать.

Это позволяет ему реально увидеть и опробовать разрабатываемую систему. Для интеграции отдельных компонентов системы в течение каждого из циклов служат промежуточные сборки (builds). Они позволяют увидеть и опробовать систему самой команде разработчиков. Если система постоянно доступна для обозрения, то тестирование становится непрекращающейся и неотъемлемой частью каждого цикла, а не лихорадочной деятельностью в самом конце работ, перед сдачей продукта.

Во время пятого и шестого шагов производится соотнесение определенных компонентов системы с циклами разработки.

Главным критерием здесь должно быть такое правило: в конце каждого цикла заказчику поставляется видимая, осязаемая, работающая часть системы.

Среди многих факторов, которые оказывают влияние на принятие того или иного решения, можно назвать следующие:

1. В конце каждого цикла заказчику должен поставляться некий осязаемый "кусок" системы.
2. В первую очередь в разработку должны идти компоненты с наиболее высоким риском.
3. График разработки компонентов должен учитывать естественные зависимости между ними.
4. Балансировка расходов ресурсов.

Фаза «Взаимодействие»: согласованная разработка компонентов.

Работающие компоненты системы появляются в результате их параллельной разработки. При этом руководителей больше должно беспокоить то, как добиться наиболее эффективного

взаимодействия и обеспечить согласованность выполняемых работ, нежели вопросы проектирования, тестирования и кодирования.

Сегодня для многих проектов (например, для тех, которые разрабатывают физически удаленные друг от друга команды или постоянно меняющиеся партнеры) наибольшую важность имеет то, как люди взаимодействуют между собой, и как они справляются с неизбежно возникающей взаимозависимостью.

Для небольших команд, где все разработчики работают в непосредственной близости друг от друга, тесная взаимосвязь и согласованность работ может быть достигнута неформальными методами. Все можно решить во время беседы в коридоре или порисовав немного у доски. В больших проектах все не так просто. Чтобы достичь согласованности действий в крупномасштабном проекте, нужно применять расширенный вариант адаптивного процесса.

В небольших командах, работающих в одном месте, согласованность, параллелизм и взаимодействие можно усилить с помощью практик методологии Extreme Programming (XP). Так, например, при парном программировании два человека работают за одним компьютером: при этом каждый непроизвольно старается отличиться, отчего улучшается общее качество работы.

Совместное владение кодом (еще одна практика XP) выводит сотрудничество между разработчиками на новый уровень. Благодаря этому подходу каждый может вносить в любой код свои собственные изменения, что заставляет всю команду работать еще более сплоченно.

Все разработчики - и по отдельности, и парами - делают все возможное, чтобы максимально повысить качество дизайна системы, кода и тестовых сценариев.

Отношения между людьми являются самым главным моментом современного бизнеса. Отношения между людьми важны не только по чисто человеческим причинам, но и как способ для достижения лучшей адаптивности и успеха в бизнесе.

На сегодняшний день строгий контроль и процесс (то есть, те составляющие, которые ранее были необходимы для успешного осуществления проектов) должны заменяться на альянсы между различными компаниями, совместное преодоление проблем и

совместное принятие решений, а также на обмен знаниями и опытом.

Фаза «Обучение»: оценка качества продукта.

Адаптивный подход к управлению разработками заключается в следующем: нужно определить масштаб и цели проекта, показать команде, какие компоненты ей необходимо разработать, а затем отойти в сторону и дать разработчикам самим решать, как они будут это делать.

Чувство ответственности в команде поддерживается при помощи периодической оценки качества. В таком случае, качество будет расти не благодаря дотошному контролю на всех этапах работ, а благодаря формированию соответствующих критериев для выпускаемого продукта и критическому его анализу.

Постоянный анализ и оценка проделанной работы - ключ к обучению.

В конце каждого цикла разработки нужно знать:

1. Качество продукта с точки зрения заказчика.
2. Качество продукта с технической точки зрения.
3. Как работает команда, и какие практики она при этом использует.
4. Текущее положение дел в проекте.

В большинстве проектов самое главное - это обеспечить заказчика возможностью работать с промежуточными версиями системы и давать комментарии разработчикам.

Вторым показателем качества продукта является его оценка с технической точки зрения. Стандартный способ определить техническое качество системы - это провести ее технический анализ. Пересматривать дизайн системы можно в конце каждого цикла, тогда как пересмотр кода или плана тестирования может проходить в течение самого цикла. Целью анализа должно быть обучение, а не поиск виноватых.

В-третьих, необходимо следить за самим процессом работы команды, вернее, за людьми и за процессом. Для этого необходимы ретроспективные пересмотры в конце проекта, а также в конце каждого цикла.

Способность организации к обучению демонстрирует, как подобный анализ ответов на следующие вопросы:

1. Что работает?

2. Что не работает?
3. Чего нам нужно делать больше?
4. Чего нам нужно делать меньше?

Благодаря чему наступает понимание и самих себя и то, как идет работа работаем. Последним пунктом плана идет определение статуса проекта, что не относится напрямую к вопросам качества. Это ведет к повторному планированию в начале каждого последующего цикла.

Основные вопросы, на которые нужно ответить:

5. На какой стадии развития находится проект?
6. Чем его положение отличается от того, что ожидалось по плану?
7. В каком положении он должен находиться?

При использовании компонентного подхода, определение статуса проекта выглядит несколько по-другому.

В методе водопада, конец каждой фазы проекта отмечен окончанием работы над определенными его артефактами (например, результатом фазы разработки спецификации является документ, в котором полностью описаны все требования к системе).

При компонентном подходе статус проекта будет определяться не одним документом или артефактом, а целым рядом компонентов, которые могут находиться на разных стадиях готовности.

"Оптимизирующие" культуры ставят во главу угла эффективность, контроль и строгий процесс. Однако эра Интернета уже внесла изменения в фундаментальную основу этих положений - в предсказуемость. Никто уже не возьмется предсказывать (с любой степенью точности) результаты работы над проектом из разряда электронной коммерции, рассчитанного на двенадцать месяцев. Слишком много в нем будет составляющих, которые будут очень часто меняться.

"Оптимизирующие" культуры видят мир в черно-белом свете. "Адаптивные" культуры, в отличие от них, признают наличие серого. Они понимают, что планирование эфемерно, а контроль практически невозможен.

"Адаптивные" культуры осознают, что успеха можно достичь, лишь испробовав несколько различных альтернативных

вариантов, учась как на успехах, так и на ошибках. Они знают, что изучать новое часто не менее важно, чем использовать в работе давно привычное.

Далеко не каждый проект можно считать сложным. В большинстве организаций сложные проекты составляют не более 20% от общего числа (однако это очень важные 20%!).

Решение сложных проблем создания и тестирования ПО не означает отказа от старых добрых практик управления проектом и сложившихся методов работы, нужно просто по-новому посмотреть на то, как их использовать.

Нужно понимать, что разработка ПО - вовсе не механический, а органичный, нелинейный и недетерминированный процесс.

Нужно изменить базовые принципы, на которых организации строят свою работу при решении сложных проблем, а также начать применять новые, связанные с ними, практики. Перейти из разряда оптимизирующих культур в разряд адаптивных означает сделать шаг в будущее, а не прозябать в прошлом.

2.2.6. Семейство методологий Crystal

Семейство методологий Crystal построено по нескольким принципам.

Все проекты разделяются по двум параметрам: критичность и величина команды.

Под критичностью понимается величина ущерба, нанесенного в результате использования продукта.

Например, ошибка в ПО для космического корабля или для аппарата искусственного дыхания несравнима с ошибкой в ПО для форума на сайте.

Жизненно-важные проекты имеют категорию L, а те, ошибка в которых обозначает потерю удобств, категорию C.

Существует еще две категории D - незначительные финансовые потери и E - значительные финансовые потери.

Диаграмма, отражающая значимость проектов базируется на Степени важности (нарастает по вертикально оси) и Величине команды (нарастает по горизонтальной оси).

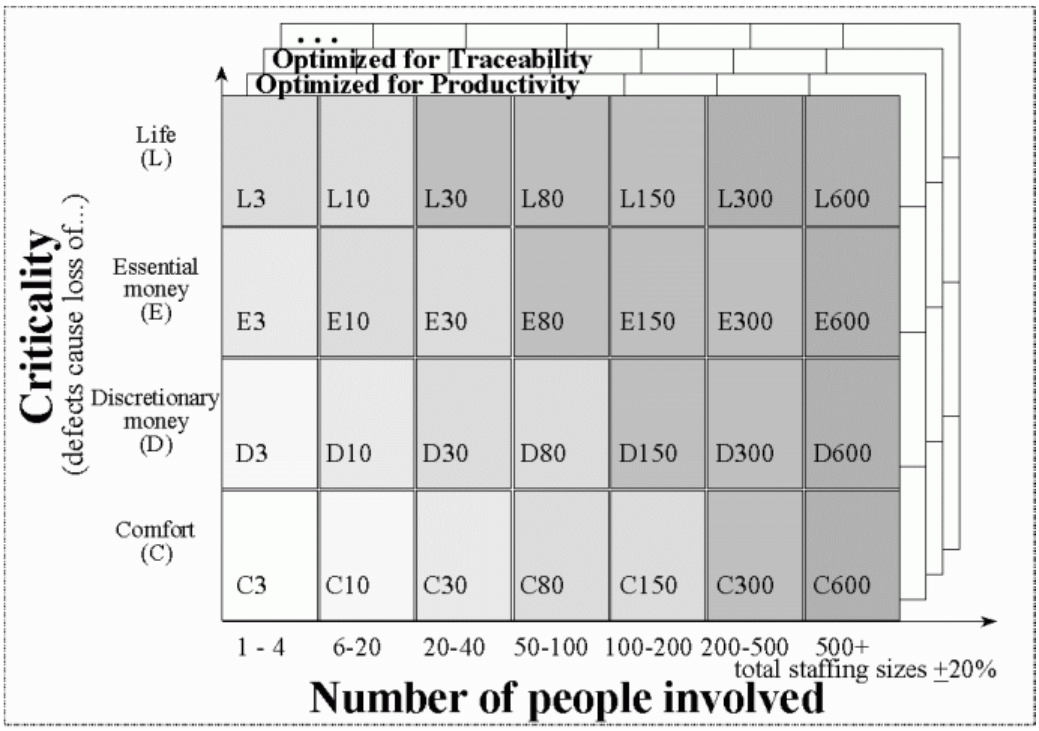


Рисунок 2.16 - Семейство методологий Crystal

Названия методологиям даются по величине команды (рисунок 2.16). Например, методологию Crystal Clear можно использовать в проектах класса C-D для команд с численностью до 6 человек. Методологию Crystal Orange для проектов класса C-E с численностью до 40 человек.

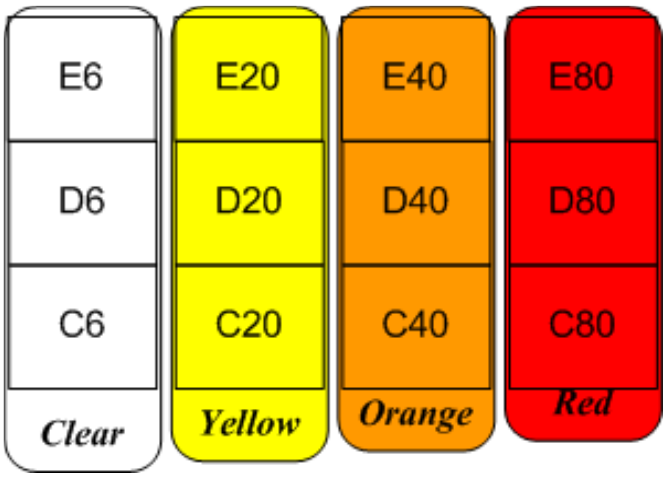


Рисунок 2.17 - Установление названий методологий

Чем ниже критичность и чем меньше команда, тем более "легкую" методологию нужно использовать. Самой легкой из всего семейства является методология Crystal Clear.

Crystal Clear. Подходит для команд численностью до 6

человек.

Главные принципы:

1. Вся команда в одном помещении. Это позволяет сократить временные затраты на коммуникацию.

2. Частые поставки продукта позволяют выработать "ритм" проекта. Ничто так не вдохновляет команду, как поставленный вовремя продукт.

3. Обмен информацией с реальными пользователями позволяет быстро получать обратную связь и ликвидировать недостатки на ранних стадиях.

4. Средства контроля версий кода обеспечивают коллективное владение кодом.

В команде выделяются следующие роли:

Спонсор, старший проектировщик-программист, проектировщик-программист, пользователь (должен присутствовать хотя бы часть времени).

Самым главным в команде является старший проектировщик-программист, он должен быть самым опытным и помогать другим членам команды.

Практики проекта:

1. Релизы выпускаются каждые два-три месяца.
2. Применяется автоматическое регрессионное тестирование.
3. Пользователи дважды просматривают каждую версию.

4. Собрания по адаптации продукта и методологии проводятся в начале и в середине каждой итерации.

5. Деятельность, зависящая от какого-либо промежуточного продукта, начинается сразу же, как только промежуточный продукт становится достаточно стабильным. Например, проектирование зависит от требований. Как только собраны главные требования, надо начинать проектирование, то есть нельзя ожидать момента, пока требования не станут совершенно стабильными.

Crystal Orange. Эта методология предназначена для проектов среднего размера в индустриальном окружении, обладающих следующими характеристиками:

- общее количество занятых в проекте людей от 10 до 40;
- продолжительность работ от 1 до 2 лет;
- важно своевременное появление продукта на рынке;

– нужно поддерживать общение между нынешними и будущими разработчиками, а также снижать временные и финансовые расходы;

– критичность: не жизненно-важная.

Это обычный тип проектов, для которых характерен поиск компромисса между количеством поставляемых компонентов (документации, руководств и пр.) и срочными изменениями в требованиях и дизайне.

Существует целый ряд проектов, для которых очень важна изменчивость, и все они могли бы с успехом использовать данную методологию.

"Оранжевая" методология семейства Crystal относится к методологии "D40". Это означает, что она подходит для команды общей численностью не выше 40 человек, работающей в одном здании над разработкой системы, сбой которой может привести к потере несущественной суммы (сюда можно отнести, например, программу учета платежей компании).

Для "Оранжевой" методологии нужна большая структуризация и координация команды, чем для проекта, над которым работают 20 человек. Однако она не предусматривает четкого деления команды на подгруппы (как это необходимо при работе над проектом, в котором задействовано, например, 80 человек) и не предполагает строгой проверки дизайна и кода системы (как это необходимо для жизненно-важных проектов).

В зависимости от команды, "Оранжевую" методологию можно использовать также для проектов типа "E50".

Для "Оранжевой" методологии семейства Crystal характерны следующие черты:

1. Роли включают в себя спонсора (Sponsor), эксперта в данном виде бизнеса (Business expert), эксперта по использованию системы (Usage expert), технического посредника (Technical facilitator), бизнес-аналитика/проектировщика (Business analyst/ designer), руководителя проекта (Project Manager), архитектора (Architect, Design Mentor), проектировщика/программиста (Designer/programmer), ведущего проектировщика/программиста (Lead designer/ programmer), инженера по повторному использованию (Reuse Point), писателя (Writer), тестировщика (Tester), дизайнера интерфейсов (UI designer).

2. Все сотрудники распределяются по командам, которые отвечают, соответственно, за планирование, руководство, архитектуру, технологии, функционирование, инфраструктуру и внешнее тестирование.

Методологические стандарты:

– программный продукт проходит процесс контроля качества каждые 3 + 1 месяц;

– прогресс в работе отслеживается по контрольным точкам поставки частей системы, а не по документации;

– применяется автоматическое регрессивное тестирование функциональности системы, в работах непосредственным образом участвуют пользователи системы, по два пользователя на релиз, вспомогательные виды деятельности начинаются только тогда, когда сама система оказывается достаточно стабильной для ее проверки пользователями.

"Настройка" методологии осуществляется на собраниях в начале и середине каждого инкремента.

Все эти правила обязательны, однако при необходимости их можно замещать другими, например, методами планирования Scrum, практиками eXtreme Programming или Adaptive Software Development.

К рабочим продуктам относятся: документ с требованиями к системе, план релизов, отчеты о состоянии работ по проекту, документ, описывающий дизайн интерфейсов, общая объектная модель системы, внутрикомандные спецификации, руководство для пользователя, исходный код, тесты и код для миграции с предыдущей системы. Каждый из упомянутых здесь продуктов дорабатывается до той степени, при которой его начнут понимать коллеги по работе, а именно до уровня детализации, допускающего попарную работу.

Все шаблоны для рабочих продуктов, стандарты кодирования, стандарты построения пользовательских интерфейсов и детали, касающиеся регрессивного тестирования, считаются локальными стандартами, то есть их должна вырабатывать и поддерживать сама команда разработчиков.

Что касается стиля работы для каждой конкретной роли, то он всецело зависит от индивида, который эту роль выполняет.

2.2.7. Методология Feature Driven Development (FDD)

Методология Feature Driven Development (FDD) была разработана экспертами в области объектно-ориентированных технологий. FDD делает основной упор на коротких итерациях, каждая из которых служит для проработки определенной части функциональности системы. Продолжительность итерации в FDD - две недели. В FDD всего пять основных процессов, из которых первые три шага относятся к началу проекта:

1. Разработка общей модели.
2. Составление списка требуемых свойств системы.
3. Планирование работы над каждым свойством.
4. Проектирование каждого свойства.
5. Конструирование каждого свойства.

Шаги 4 и 5 необходимо повторять в каждой итерации, где каждый процесс разбивается на задачи и имеет критерии верификации. В процессе разработки все разработчики делятся на две роли: «class owners» (владельцы классов) и «chief programmers» (старшие программисты).

Соответственно старшие программисты – это наиболее опытные разработчики, им поручается разработка конкретных свойств системы, но они не занимаются этим самостоятельно.

Рисунок 2.18 - Иерархия разработки Web-сайта

Задача старшего программиста – определить, какие классы заняты в реализации конкретного свойства будущей системы, и собрать команду из владельцев классов для реализации этого свойства. По сути, старший программист выполняет роль координатора и архитектора.

На рисунке 2.18 приведена иерархия разработки применительно к созданию Web-сайта.

2.3. Сравнение методологий разработки ПО

Для принятия оптимальных решений при выборе методологии разработки ПО, необходимо знать их характеристики и области применения. Методы правила и категории разработки ПО необходимо рассматривать в сравнении.

Самая сложная для описания категория «Как получится…» - поскольку в нее входят как продукт судорожных метаний новичка, пытающегося любой ценой выполнить свой первый проект, так и вполне зрелые и устоявшиеся методологии, вобравшие в себя многолетний и разнообразный опыт конкретных команд разработчиков и даже описанные во внутренних регламентах. Поскольку люди, способные разработать собственную методологию, как правило, могут сами оценить ее в плане итеративности и формализованности, то необходимо ориентироваться на новичков.

Чаще всего это означает, что правил ведения разработки либо не существует вообще, либо они разработаны и приняты, но не выполняются. Естественным в таких условиях является крайне низкий уровень формализма разработки.

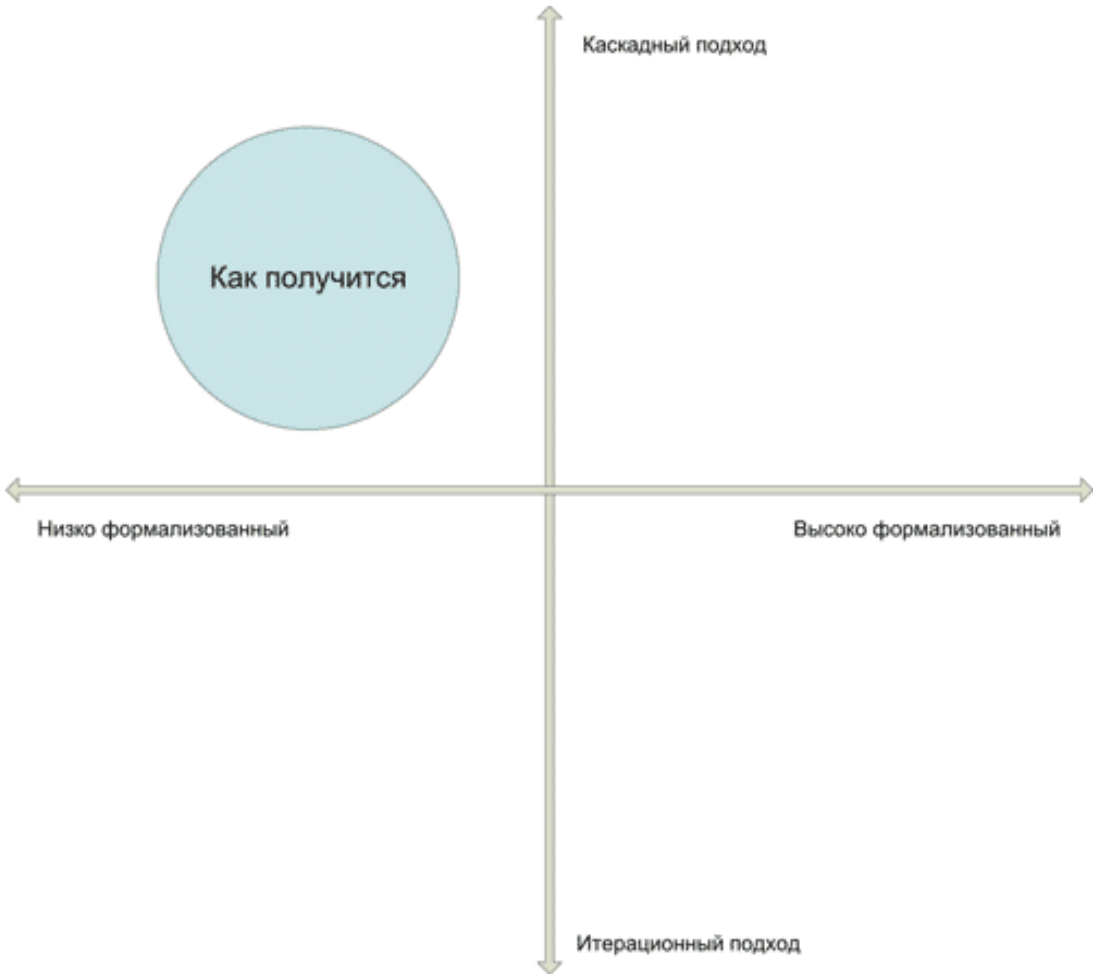


Рисунок 2.19 - Подходы к разработке ПО

Итеративный подход, как правило, в таких проектах не используется. Прежде всего потому, что он бы позволил еще на первых итерациях оценить проект как крайне сомнительный и требующий срочного вмешательства более высокого руководства для наведения порядка.

Известно, что в итеративном проекте традиционный ответ программиста, что у него уже все на 90% готово, проходит только до момента завершения первой итерации.

В рамках данного пособия необходимо рассмотреть хорошо зарекомендовавшие себя методы.

2.3.1. Структурные методы

Группа методологий, разработанных, как правило, еще до широкого распространения объектно-ориентированных языков. Все они предполагают каскадную разработку.

Основу этих методологий составляют последовательный

переход от работы к работе и передача результатов (документов) очередного этапа участникам последующего.

Также все эти методологии предполагают высокоформализованный подход, хотя высказывания о разумном количестве документации можно найти и в них (рисунок 2.20).

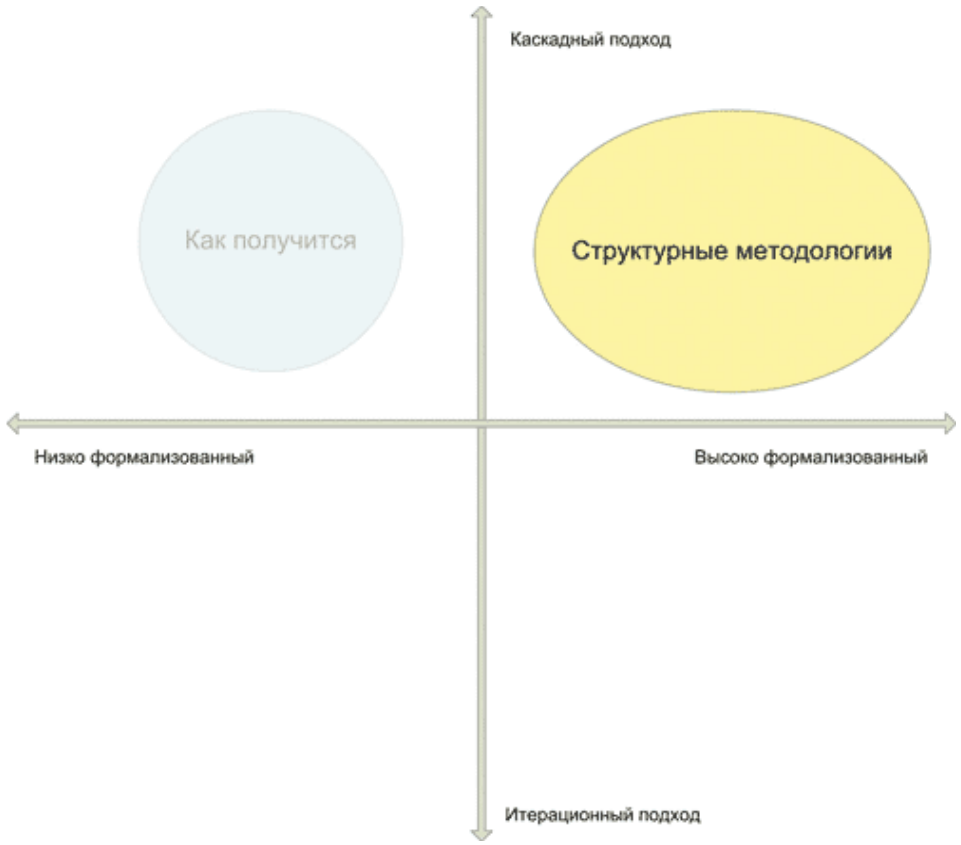


Рисунок 2.20 - Положение структурной методологии разработки ПО

2.3.2. Гибкие методологии

Гибкие методологии базируются на десяти принципах, из которых назовем лишь те, которые определяют оценку этих методологий по выбранным параметрам:

- главное - удовлетворить заказчика и предоставить ему продукт как можно скорее;
- новые выпуски продукта должны появляться раз в несколько недель, в крайнем случае - месяцев;
- наиболее эффективный способ передачи знаний участникам

разработки и между ними - личное общение;
- работающая программа - лучший показатель прогресса разработки.

Эти методы явно ориентированы на итеративную разработку ПО и на минимальную формализацию процесса.

Однако, относительно второго пункта необходимо сделать оговорку: названные методы ориентированы на минимально допустимый для данного проекта уровень формализации.

Например, в методологии Crystal имеются модификации, предназначенные для выполнения процессов с различным количеством участников и разной критичностью разрабатываемого ПО (критичность ПО определяется возможными последствиями ошибок, которые могут меняться в диапазоне от незначительных финансовых потерь на исправление ошибки до катастрофических).

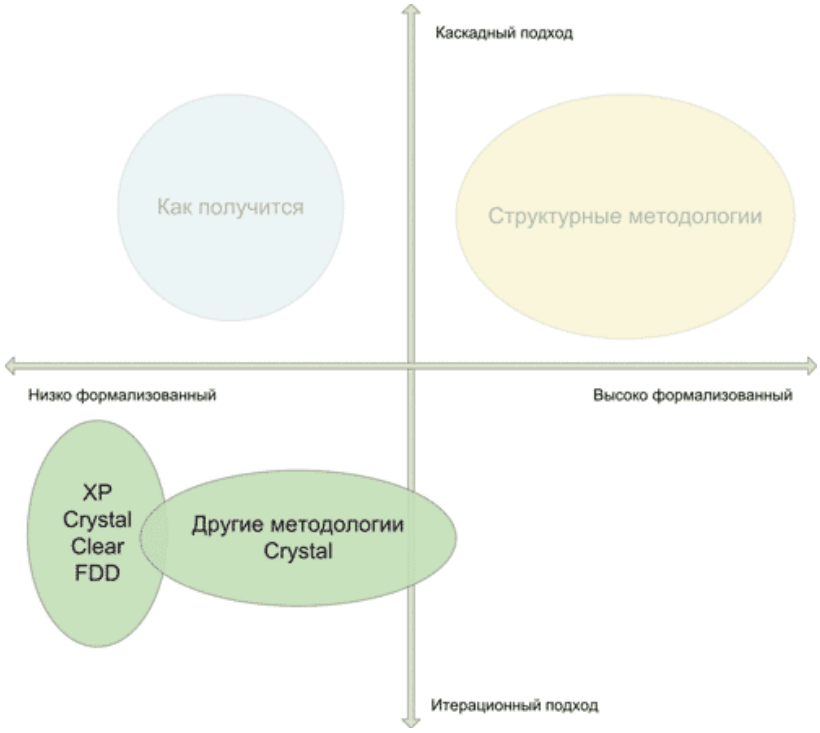


Рисунок 2.21 - Положение методологий Crystal

Список гибких методологий в настоящее время достаточно широк. Тем не менее описанные нами методологии дают весьма полное представление обо всем семействе (рисунок 2.21).

Практически все гибкие методологии используют итеративный подход, при котором детально планируется только ограниченный

объем работ, связанный с выпуском очередного релиза.

Практически все гибкие методологии ориентированы на максимально неформальный подход к разработке. Если проблему можно решить в ходе обычной беседы, то лучше именно так и поступить. Причем оформлять принятое решение в виде бумажного или электронного документа нужно только тогда, когда без этого невозможно обойтись.

ГОСТы, как и описываемые в следующем разделе требования модели CMM, не являются методологиями.

Они, как правило, не описывают сами процессы разработки ПО, а только формулируют определенные требования к процессам, которым в той или иной степени соответствуют различные методологии.

Сравнение требований по тем же критериям, по которым мы сравниваем методологии, поможет сразу определиться с тем, какими методологиями стоит пользоваться, если вам нужно выполнить разработку в соответствии с ГОСТ (рисунок 2.22).

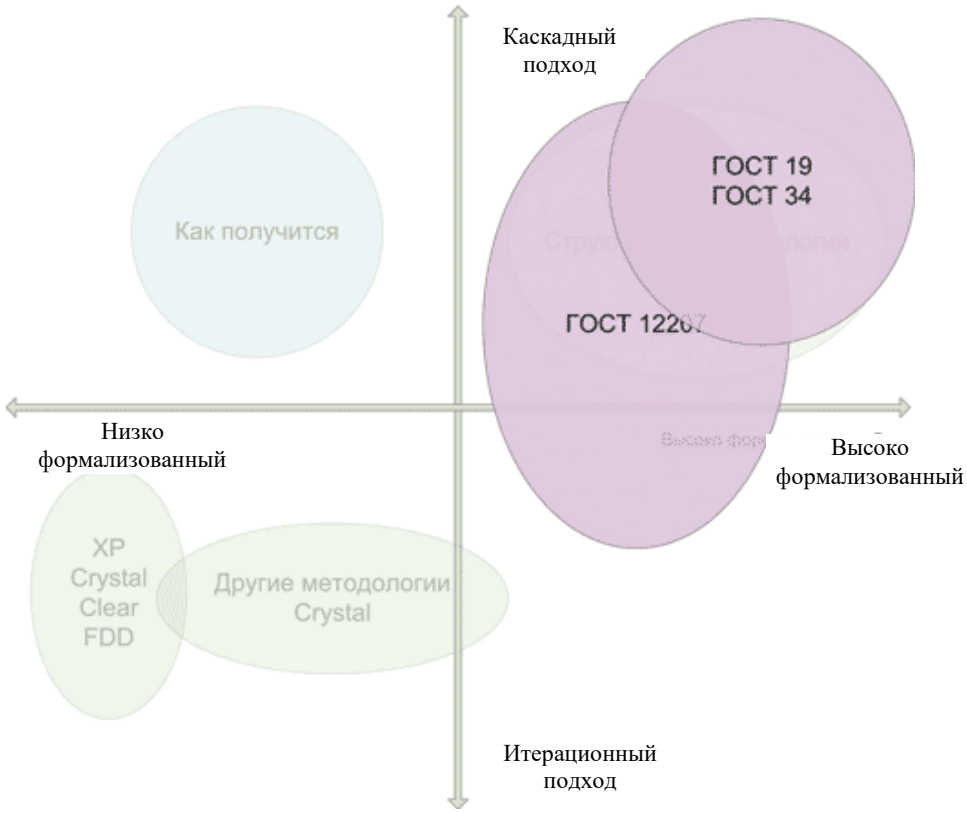


Рисунок 2.22 - Положение ГОСТов в разработке ПО

2.3.3. Модели зрелости процесса разработки

Помимо государственных и международных стандартов, существует несколько подходов к сертификации процесса разработки. Наиболее известными из них являются модели зрелости процесса разработки CMM и CMMI.

CMM (Capability Maturity Model) — модель зрелости процессов создания ПО, которая предназначена для оценки уровня зрелости процесса разработки в конкретной компании. В соответствии с этой моделью имеется пять уровней зрелости процесса разработки (рисунок 2.23).

Первый уровень соответствует разработке «как получится», когда на каждый проект разработчики идут как на подвиг.

Второй соответствует более-менее налаженным процессам, когда можно с достаточной уверенностью рассчитывать на положительный исход проекта.

Третий соответствует наличию разработанных и хорошо описанных процессов, используемых при разработке.

Четвертый соответствует активному использованию метрик в процессе управления для постановки целей и контроля их достижения.

Пятый уровень означает способность компании оптимизировать процесс по мере необходимости.

После появления CMM стали разрабатываться специализированные модели зрелости для создания информационных систем, для процесса выбора поставщиков и некоторые другие.

На их основе была разработана интегрированная модель CMMI (Capability Maturity Model Integration). Кроме того, в CMMI была предпринята попытка преодолеть проявившиеся к тому времени недостатки CMM — преувеличение роли формальных описаний процессов, когда наличие определенной документации оценивалось значительно выше, чем просто хорошо налаженный, но не описанный процесс. Тем не менее CMMI также ориентирован на использование весьма формализованного процесса.

Основой моделей CMM и CMMI является формализация процесса разработки. Они нацеливают разработчиков на

внедрение детально описанного в регламентах и инструкциях процесса, который, в свою очередь, не может не требовать разработки большого объема проектной документации для соответствующего контроля и отчетности.

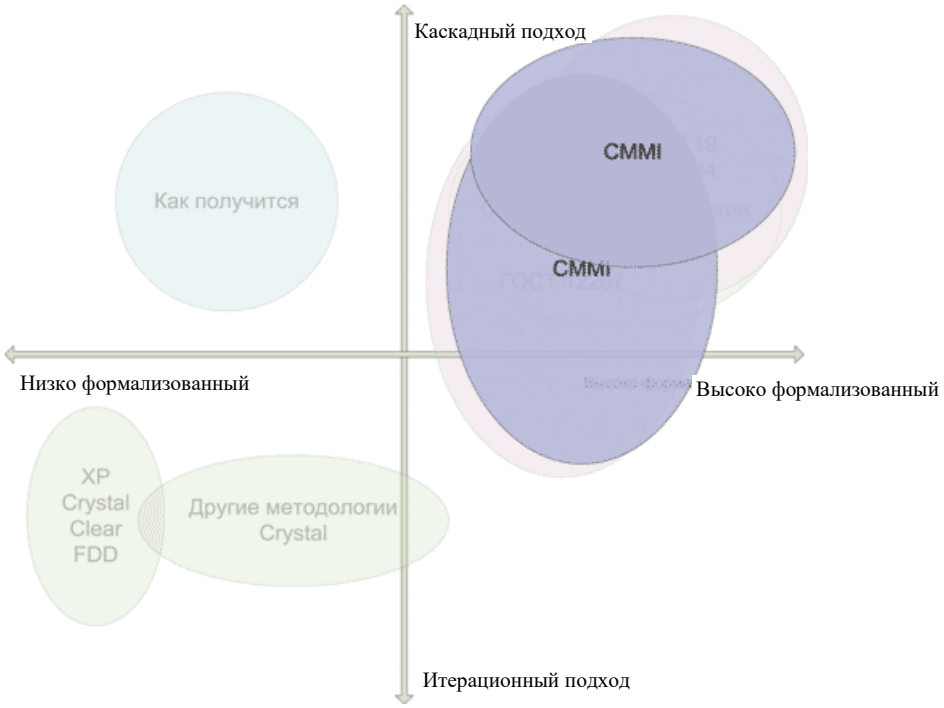


Рисунок 2.23 - Положение моделей зрелости в разработке ПО

Связь CMM и CMMI с итеративной разработкой более опосредованная. Формально ни та ни другая не выдвигают конкретных требований к тому, чтобы придерживаться каскадного или итеративного подхода. Однако, по мнению ряда специалистов, CMM в большей степени совместима с каскадным подходом, в то время как CMMI допускает также и применение итеративного подхода.

2.3.4. Методология RUP

Несомненно – RUP, это итеративная методология. Хотя формально обязательность выполнения всех фаз или какого-то минимального числа итераций нигде в RUP не обозначена, весь подход ориентирован на то, что их достаточно много.

Ограниченное количество итераций не позволяет в полной мере использовать все преимущества RUP.

Вместе с тем RUP можно применять и в практически

каскадных проектах, включающих реально всего пару итераций: одну в фазе «Построение», а другую в фазе «Передача». Кстати говоря, в каскадных проектах реально используется именно такое количество итераций. Ведь проведение испытаний и опытной эксплуатации системы предполагает внесение исправлений, которые могут подразумевать определенные действия, связанные с анализом, проектированием и разработкой, то есть фактически являются еще одним проходом через все фазы разработки (рисунок 2.24).

Что касается формальности методологии, то здесь RUP предоставляет пользователю весьма широкий диапазон возможностей. Если выполнять все работы и задачи, создавать все артефакты и достаточно формально (с официальным рецензентом, с подготовкой полной рецензии в виде электронного или бумажного документа и т.д.) проводить все рецензирования, RUP может оказаться крайне формальной, тяжеловесной методологией.

В то же время RUP позволяет разрабатывать только те артефакты и выполнять только те работы и задачи, которые необходимы в конкретном проекте. А выбранные артефакты могут выполняться и рецензироваться с произвольной степенью формальности. Можно требовать детальной проработки и тщательного оформления каждого документа, предоставления столь же тщательно выполненной и оформленной рецензии и даже, следуя старой практике, утверждать каждую такую рецензию на научно-техническом совете предприятия. А можно ограничиться электронным письмом или наброском на бумаге.

Кроме того, всегда остается еще одна возможность: сформировать документ в голове, то есть обдумать соответствующий вопрос и принять конструктивное решение. И если это решение касается только разработчика, то ограничиться, например, комментарием в коде программы.

Таким образом, RUP — итеративная методология с очень широким диапазоном возможных решений в части формализации процесса разработки.

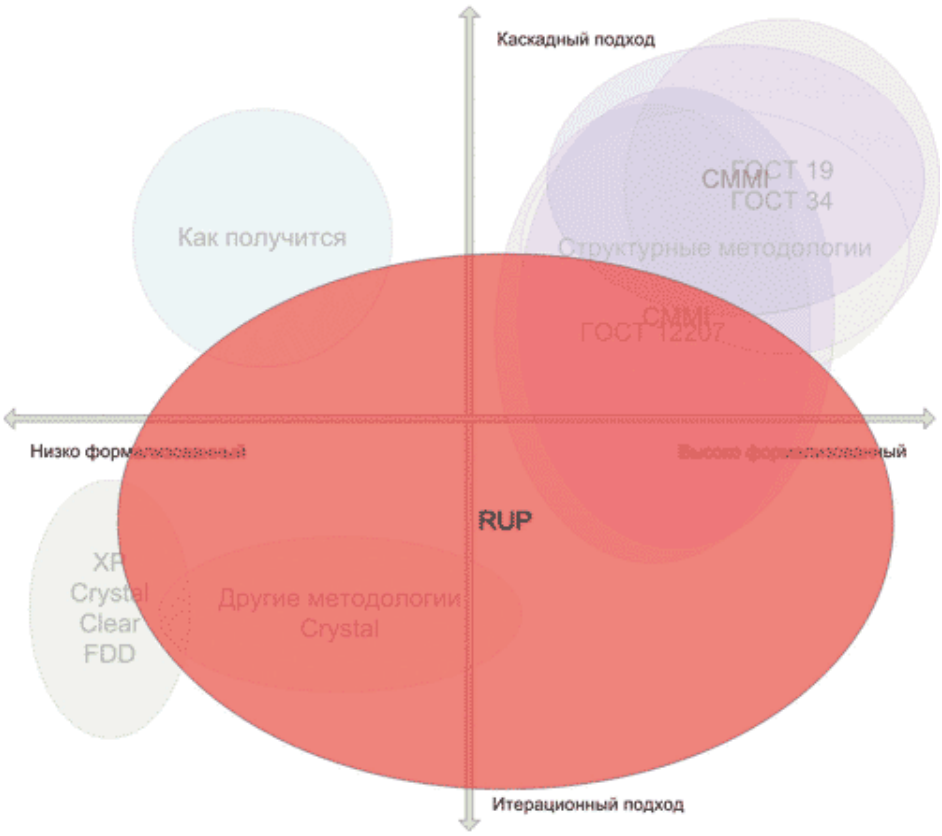


Рисунок 2.24 - Положение методологии RUP в разработке ПО

Подведем итоги второго раздела пособия. RUP, в отличие от большинства других методологий, позволяет в широком диапазоне выбирать степень формализации и итеративности процесса разработки в зависимости от особенностей проектов и разрабатывающей организации.

3. СОПРОВОЖДЕНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SOFTWARE MAINTENANCE)

Процесс разработки ПО заканчивается передачей продукта в эксплуатацию, то есть началом «самостоятельной жизни». Важным элементом жизненного цикла программного продукта, позволяющим отодвинуть границы морального устаревания, а значит увеличить срок эксплуатации является *сопровождение* ПО.

Соответственно, в процессе эксплуатации продукт будет изменяться или эволюционировать. Связано это с обнаружением при реальном использовании скрытых дефектов, изменениями в операционном окружении, необходимостью покрытия новых требований и т.п.

Фаза сопровождения в жизненном цикле, обычно, начинается сразу после приемки/передачи продукта и действует в течение периода гарантии или технической поддержки. Но деятельность, связанная с сопровождением, начинается намного раньше.

Традиционно сложилось так, что вопросам сопровождения уделяется существенно меньше внимания, чем другим фазам жизненного цикла. В большой степени это связано с сокращением инвестиций организаций непосредственно в разработку программных систем, с целью увеличения сроков использования уже существующего и применяемого ПО.

Сопровождение программного обеспечения в SWEBOK определяется как вся совокупность деятельности, необходимой для обеспечения эффективной (с точки зрения затрат) поддержки программных систем. Эти работы выполняются как перед вводом системы в эксплуатацию, так и после этого. Предварительные работы включают планирование деятельности по сопровождению системы, а также организацию перехода к ее полнофункциональному использованию. Если новая система должна заменить старую систему, предназначенную для решения тех же задач, просто на новом уровне эффективности, стоимости использования, новых функциональных возможностей, в этом случае важно обеспечить плавный переход со старой системы на новую, максимально естественный для пользователей. С этим связано не только планирование, например, переноса информации, хранимой в соответствующих базах данных, но и обучение

пользователей, подготовка, настройка и проверка рабочей конфигурации, определение последовательности операций, организация и обучение службы поддержки (help-desk) и т.п.

Область знаний «Сопровождение программного обеспечения» связана с другими аспектами программной инженерии. Описание этой области знаний непосредственно пересекается со всеми другими дисциплинами. Область знаний «Сопровождение программного обеспечения» приведена на рисунке 3.1.

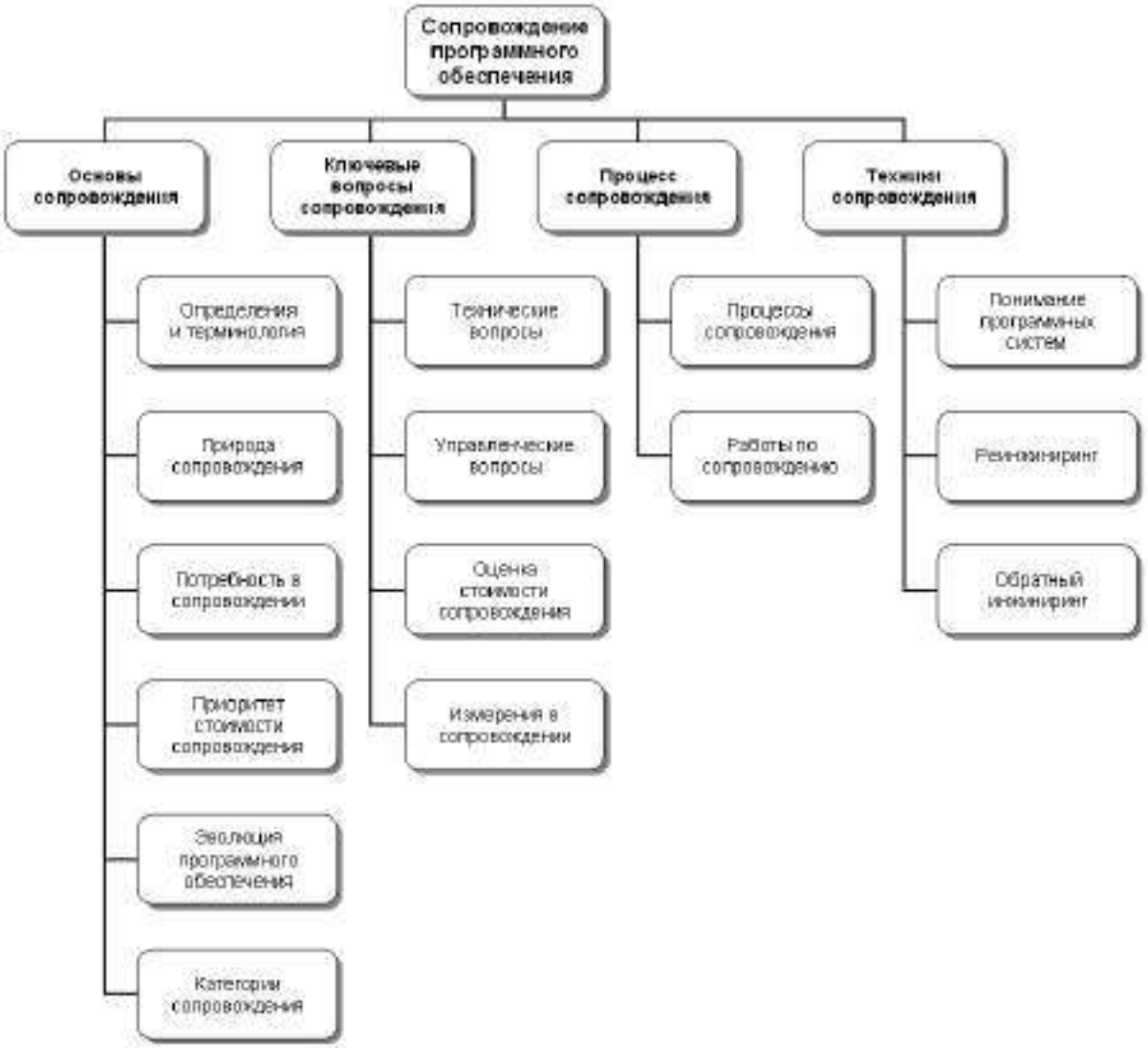


Рисунок 3.1 - Область знаний «Сопровождение программного обеспечения»

Сопровождение ПО включает четыре ключевых группы, назначение и содержательную часть которых необходимо рассмотреть подробнее.

3.1. Основы сопровождения программного обеспечения (Software Maintenance Fundamentals)

Группа «Основы сопровождения» включает концепции и терминологию, формирующие основы понимания роли и содержания работ по сопровождению программных систем. Темы данной группы предоставляют соответствующие определения и описывают, почему именно существует потребность в сопровождении. Категории сопровождения критически важны для понимания сути рассматриваемых вопросов.

3.1.1. Определения и терминология (Definitions and Terminology)

Сопровождение программного обеспечения определяется стандартом IEEE Standard for Software Maintenance (IEEE 1219) как модификация программного продукта после передачи в эксплуатацию для устранения сбоев, улучшения показателей производительности и/или других характеристик (атрибутов) продукта, или адаптации продукта для использования в модифицированном окружении.

Кроме этого, данный стандарт также касается вопросов подготовки к сопровождению до передачи системы в эксплуатацию, однако, структурно это сделано на уровне соответствующего информационного приложения, включенного в стандарт.

В свою очередь, стандарт жизненного цикла 12207 (IEEE, ISO/IEC, ГОСТ Р ИСО/МЭК) позиционирует сопровождение как один из главных процессов жизненного цикла. Этот стандарт описывает сопровождение как процесс модификации программного продукта в части его кода и документации для решения возникающих проблем (при эксплуатации) или реализации потребностей в улучшениях (тех или иных характеристик продукта). Задача состоит в модификации продукта при условии сохранения его целостности. Международный стандарт ISO/IEC 14764 (Standard for Software Engineering - Software Maintenance) определяет сопровождение программного обеспечения в тех же терминах, что и стандарт 12207, придавая особое значение работам по подготовке к деятельности по сопровождению до передачи системы в реальную эксплуатацию,

например, вопросам планирования регламентов и операций по сопровождению.

3.1.2. Природа сопровождения (Nature of Maintenance)

Сопровождение поддерживает функционирование программного продукта на протяжении всего операционного жизненного цикла, то есть периода его эксплуатации. В процессе сопровождения фиксируются и отслеживаются запросы на модификацию (также называемые “запросами на изменения” – change requests, в частности, в контексте конфигурационного управления), оценивается влияние предлагаемых изменений, модифицируется код и другие активы (артефакты) продукта, проводится необходимое тестирование и выпускается обновленная версия продукта. Кроме того, проводится обучение пользователей и обеспечивается их ежедневная поддержка при работе с текущей версией продукта. Сопровождение, с точки зрения операций отслеживания и контроля, обладает большим содержанием, чем разработка (в общем понимании). При этом объем и активность операций по контролю разработки в большой степени зависит от сложившихся практик, внутрикорпоративных регламентов и требований, а также применяемых методологий и концепции управления (в частности – проектного менеджмента). Соответственно, отслеживание и контроль – ключевые элементы деятельности по сопровождению программного обеспечения (как и других ИТ-активов предприятия).

Специалисты по сопровождению (персонал сопровождения) могут получать знания о программном продукте непосредственно от разработчиков. Взаимодействие с разработчиками и раннее привлечение этих специалистов помогает уменьшить усилия, необходимые для адекватного сопровождения программной системы. Передача знаний персоналу сопровождения, его обучение, должно начинаться не позднее начала опытной эксплуатации продукта. Иначе, усилия на одновременную поддержку прикладной системы и обучение соответствующих специалистов не только превысит реально допустимые нормы загрузки персонала (как группы или службы сопровождения и техподдержки, так и разработчиков системы), но и снизит эффективность поддержки пользователей на критически важном

этапе первоначального использования новой системы. Таким образом, начинает формироваться информационная инфраструктура службы технической поддержки и сопровождения у производителей программных продуктов.

3.1.3. Потребность в сопровождении (Need for Maintenance)

Сопровождение необходимо для обеспечения того, чтобы программный продукт на протяжении всего периода эксплуатации удовлетворяет требованиям пользователей. Деятельность по сопровождению применима для программного обеспечения, созданного с использованием любой модели жизненного цикла и методологии разработки. Изменения программной системы могут быть обусловлены как действиями по корректировке ее поведения или несвязанные с необходимостью корректировки (подразумевая уже не исправление ошибок, например, повышение производительности или расширение функциональности).

В общем случае, работы по сопровождению должны проводиться для решения следующих задач:

- устранение сбоев;
- улучшение дизайна;
- реализация расширений (функциональных возможностей);
- создание интерфейсов взаимодействия с другими (внешними) системами;
- адаптация для возможности работы на другой аппаратной платформе (или обновленной платформе), применения новых системных возможностей, функционирования в среде обновленной телекоммуникационной инфраструктуры и т.п.;
- миграции унаследованного (legacy) программного обеспечения;
- вывода программного обеспечения из эксплуатации.

Деятельность персонала сопровождения включает четыре ключевых аспекта:

- поддержка контроля (управляемости) программного обеспечения в течение всего цикла эксплуатации;
- поддержка модификаций программных систем;
- совершенствование существующих функций;
- предотвращение падения производительности программной системы до неприемлемого уровня.

3.1.4. Приоритет стоимости сопровождения (Majority of Maintenance Costs)

Работы по сопровождению потребляют если не большую, то значительную часть финансовых ресурсов жизненного цикла программного обеспечения. Общее понимание сопровождения подразумевает лишь устранение сбоев. Однако, исследования и опросы показывают, что более 80% усилий по сопровождению связаны не столько устранением сбоев, сколько с другими работами, не связанными с исправлением дефектов.

Существуют как технические, так и организационные факторы, оказывающие влияние на стоимость сопровождения, в целом:

- тип приложения;
- новизна программного обеспечения;
- наличие и квалификация персонала по сопровождению;
- длительность использования программной системы;
- характеристики и специфика аппаратной части (а также телекоммуникационной инфраструктуры);
- качество дизайна (например, модульность или масштабируемость), кода, документации и соответствующих работ по тестированию системы.

3.1.5. Эволюция программного обеспечения (Evolution of Software)

В 1969 году Леман впервые связал деятельность по сопровождению и вопросы эволюции программного обеспечения. Результаты более чем 20-летних исследований во главе с Леманом привели к формулированию ряда важных положений, ключевое из которых утверждает, что деятельность по сопровождению, по-сути, представляет собой эволюционную разработку программных систем. Принятию тех или иных решений в процессе сопровождения, помогает понимание того, что происходит с программной системой в процессе ее эксплуатации. Существующее (особенно, корпоративное) программное обеспечение никогда не бывает полностью завершенным и продолжает эволюционировать в течение всего срока эксплуатации. В процессе эволюционирования, программная

система становится все более сложной до тех пор, пока не предпринимаются специальные усилия (в том числе, в рамках специального проекта по модификации) по уменьшению его сложности.

3.1.6. Категории сопровождения (Categories of Maintenance)

Многие источники, в частности, стандарт IEEE 1216, определяют три категории работ по сопровождению: корректировка, адаптация и совершенствование. Такая классификация была обновлена в стандарте ISO/IEC 14764 Standard for Software Engineering - Software Maintenance введением четвертой составляющей. Таким образом, сегодня говорят о четырех категориях сопровождения:

• Корректирующее сопровождение (corrective maintenance): “реактивная” модификация программного продукта, выполняемая уже после передачи в эксплуатацию для устранения сбоев;

• Адаптирующее сопровождение (adaptive maintenance): модификация программного продукта на этапе эксплуатации для обеспечения продолжения его использования с заданной эффективностью (с точки зрения удовлетворения потребностей пользователей) в изменившемся или находящемся в процессе изменения окружении; в первую очередь, подразумевается изменение бизнес-окружения, порождающее новые требования к системе;

• Совершенствующее сопровождение (perfective maintenance): модификация программного продукта на этапе эксплуатации для повышения характеристик производительности и удобства сопровождения;

• Профилактическое сопровождение (preventive maintenance): модификация программного продукта на этапе эксплуатации для идентификации и предотвращения скрытых дефектов до того, когда они приведут к реальным сбоям.

3.2. Ключевые вопросы сопровождения программного обеспечения (Key Issues in Software Maintenance)

Для обеспечения эффективного сопровождения программных систем необходимо решать целый комплекс вопросов и проблем,

связанных с соответствующими работами. Необходимо понимать, что процесс сопровождения предъявляет уникальные технические и управленческие требования к персоналу, занимающемуся сопровождениям и, в первую очередь, специалистам-инженерам.

Попытка найти дефект в продукте, содержащем 500 тысяч строк кода, написанных другими инженерами – яркий пример сложностей, с которыми приходится сталкиваться инженерам по сопровождению. Другой пример, уже организационный, постоянная борьба за ресурсы с разработчиками. Одновременное планирование перспективной версии системы, реализация следующей версии и подготовка критических патчей для текущей версии – еще один пример проблем, с которыми приходится сталкиваться в процессе эксплуатации программного обеспечения.

3.2.1. Технические вопросы (Technical Issues)

К техническим вопросам относят: ограниченное понимание (Limited understanding); тестирование (Testing); анализ влияния (Impact analysis); возможность сопровождения (Maintainability).

Ограниченное понимание подразумевает как быстро инженер по сопровождению может понять где необходимо внести исправления или изменения в код системы, которую он не разрабатывал. Исследования показывают, что от 40 до 60 процентов усилий по сопровождению тратится на анализ и понимание сопровождаемого программного обеспечения. Формирование целостного взгляда о системе представляет большое значение для инженеров. Этот процесс более сложен в случае анализа текстового представления системы – ее исходного кода, особенно, когда процесс эволюции системы от сборки к сборке, от релиза к релизу, в нем никак не отмечен, не документирован и когда разработчики не могут объяснить историю и структуру изменений.

Тестирование. Стоимость повторения полного набора тестов для основных модулей системы может быть существенным как по времени, так и по стоимости. Для сопровождения системы особо значимым является выборочное регрессионное тестирование системы или его компонент для проверки того, что внесенные изменения не привели к непреднамеренному изменению поведения программного обеспечения.

Анализ влияния. Анализ влияния описывает как проводить (в частности, с точки зрения эффективности затрат) полный анализ возможных последствий и влияний изменений, вносимых в существующую систему. Персонал сопровождения должен обладать необходимыми знаниями о специфике системы ее содержании и структуре. Инженеры используют эти знания для выполнения работ по анализу влияния, идентифицируя все системы и программные продукты, на которые могут повлиять изменения, вносимые в обслуживаемую программную систему. При этом, должны быть определены риски, связанные с внесением обсуждаемых изменений.

Возможность сопровождения. Для уменьшения стоимости дальнейшего сопровождения, на протяжении всего процесса разработки необходимо специфицировать, оценивать и контролировать характеристики, влияющие на возможность сопровождения. Если такие работы проводятся регулярно, это облегчает дальнейшее сопровождение, повышая его сопровождаемость (в частности, как характеристику качества). Этого сложно добиться потому, что такого рода характеристики игнорируются при разработке. Разработчики заняты другими запланированными работами и также часто пренебрегают требованиями, предъявляемыми к сопровождаемости системы.

Одной из ключевых проблем сопровождения является отсутствие системной документации, мешающее формированию понимания системы и, как следствие, невозможности адекватного анализа влияния. Эта и другие проблемы могут быть решены при использовании систематического подхода к построению зрелых процессов, применению соответствующих техник и автоматизации необходимых задач по поддержке жизненного цикла с помощью специализированных инструментальных средств.

3.2.2. Управленческие вопросы (Management Issues)

В управленческих вопросах выделяют: согласование с организационными целями (Alignment with organizational objectives); проблемы кадрового обеспечения (Staffing); процесс (Process); организационные аспекты сопровождения (Organizational aspects of maintenance); аутсоурсинг (Outsourcing).

Согласование с организационными целями. Организационные цели описывают как продемонстрировать возврат инвестиций от деятельности по сопровождению программного обеспечения. Обычно, разработка ведется на проектной основе, с определенными временными и бюджетными ограничениями. Главный акцент делается на выпуске системы, отвечающей потребностям пользователей, в заданные сроки и в рамках бюджета. В отличие от этого, сопровождение системы преследует цели максимального продления срока эксплуатации программного обеспечения. Такой подход может основываться на необходимости обновления и расширения программного обеспечения, как отклика на изменяющиеся потребности пользователей. При этом, оценка возврата инвестиций становится более сложной и приводит к формированию точки зрения старшего менеджмента, что деятельность по сопровождению потребляет значительную часть ресурсов без явно выраженной и количественно определяемой отдачи для организации.

Проблемы кадрового обеспечения. Данная тема касается вопросов привлечения и удержания квалифицированного персонала по сопровождению. Работа по сопровождению не выглядит привлекательной, инженеры по поддержке воспринимаются как специалисты “второго класса” (в SWEBOK используется устойчивое выражение “second-class citizens”), что приводит к безусловному падению духа коллектива, отвечающего за поддержку систем. На более высоком уровне, для организаций и бизнесов - потребителей информационных технологий, эта задача связана с внутрикорпоративными департаментами автоматизации, в целом, которые воспринимаются только как центры затрат, а информационные технологии не рассматриваются как актив. В результате, такая позиция приводит к снижению эффективности работы подразделений автоматизации, а, следовательно, и падению качества информационного обеспечения бизнеса, что сказывается, в подавляющем большинстве случаев, и на бизнесе, как таковом.

Процесс. Процесс в общем случае, жизненный цикл, является набором работ (activities), методов, практик и, своего рода, трансформаций, которые используются людьми для разработки и сопровождения программных систем и ассоциированных с ними

продуктов. На уровне процесса, деятельность по сопровождению программного обеспечения имеет очень много общего с разработкой, например, в части конфигурационного управления, являющегося критически важной составляющей обоих видов деятельности. В то же время, сопровождение включает работы, не представленные в процессе разработки. Эта деятельность требует от менеджмента специального внимания.

Организационные аспекты сопровождения. В первую очередь, организационные вопросы подразумевают какая организация будет отвечать и/или какие функции необходимо выполнять для обеспечения деятельности по сопровождению.

Команда, разрабатывавшая программный продукт, далеко не всегда отвечает за его сопровождение. Это не только стандартное управленческое решение независимых поставщиков программного обеспечения, но, также встречается в организациях, использующих программные продукты в целях автоматизации своих бизнес-функций.

При решении вопроса, где (кем) будут осуществляться функции по сопровождению, может быть принято решение оставить их непосредственно тем, кто разрабатывал систему (как в терминах организации/компании, так и подразумевая непосредственно коллектив разработчиков), или передать другой команде или стороне (maintaner). Часто, выбор сопровождающей организации осуществляется исходя из тех соображений, которые выглядят обоснованными для обеспечения адекватной поддержки системы и возможности ее эволюционирования для удовлетворения меняющихся потребностей пользователей.

Соответствующие решения принимаются в каждом конкретном случае, с учетом его специфики (case-by-case). Но делегирование или назначение полномочий и ответственности по сопровождению должно быть произведено по отношению только к одной организации или лицу (менеджеру соответствующей команды поддержки). Все зависит от организационной структуры организации/компании, эксплуатирующей программное обеспечение.

Аутсоурсинг. Заимствованный термин “аутсоурсинг” уже прижился не только в среде ИТ-менеджеров, он стал частью современного бизнеса и управленческих практик. Суть его

заключается в передаче работ, в первую очередь, вспомогательных (непрофильных для организации) “на сторону”. Крупные корпорации передают в управление другим организациям целые портфели программных систем, а, иногда, и целиком всю ИТ-инфраструктуру. В то же время, существенно более часто, сопровождение передается другим организациям только для “второстепенных” программных систем, так как владельцы таких систем не желают терять контроль над ассоциированными с этими системами данными и/или функциональностью. Отмечается, что некоторые передают работы по сопровождению “в аутсоурсинг” только в тех случаях, если убеждены в стратегическом контроле над сопровождением. Но, это тема уже относится к общим вопросам управления и требует самостоятельного обсуждения, в контексте конкретной организации или бизнес-структуры.

При этом контроль сложно измерить. В свою очередь, перед аутсоурсером стоит серьезная проблема по определению содержания соответствующих работ, в том числе, для описания содержания соответствующего контракта. Отмечается, что около 50% сервисов, предоставляемых аутсоурсером, проводятся без соответствующего детального и однозначно интерпретируемого соглашения. Компании, занимающиеся аутсоурсингом, обычно затрачивают несколько месяцев на оценку программного обеспечения прежде, чем заключают соответствующий контракт. Еще один вопрос, требующий специального внимания, заключается в необходимости определения процесса и процедур передачи программного обеспечения на внешнее сопровождение.

3.2.3. Оценка стоимости сопровождения (Maintenance Cost Estimation)

Инженеры должны понимать разницу в различных категориях сопровождения. Это, в большой степени, необходимо для оценки соответствующих затрат. С точки зрения планирования, как составной части проектной и управленческой деятельности, оценка стоимости является важным аспектом деятельности по сопровождению программного обеспечения.

Оценка стоимости (Cost Estimation). Анализа влияния помогает в оценке стоимости работ по сопровождению. На эти затраты оказывает влияние множество технических и других

факторов. ISO/IEC 14764 (Standard for Software Engineering – Software Maintenance) определяет, что “существует два наиболее популярных метода оценки стоимости сопровождения – параметрическая модель и использование опыта”. Чаще всего, оба этих подхода комбинируются для повышения точности оценки.

Параметрические модели (Parametric models). SWEBOK приводит ряд источников, подробно рассматривающих вопросы оценки стоимости сопровождения и, в частности, параметрические модели. Для использования таких моделей используются данные предыдущих проектов. Наравне с историческими данными используется метод функциональных точек (стандарт IEEE 14143.1-00).

Опыт (Experience). Среди тех подходов, которые позволяют повысить точность оценок, полученных при использовании параметрических моделей – применение опыта (в форме экспертного мнения, например, при использовании техники оценки “Delphi”, название которой происходит от “делфийского оракула”), аналогий, а также структуры декомпозиции работ. Наилучшие результаты получаются в случае сочетания эмпирических методов с имеющимся опытом. Получаемые данные используются как результат программы измерения аспектов сопровождения.

3.2.4. Измерения в сопровождении программного обеспечения (Software Maintenance Measurement)

Формы и данные измерений в процессе сопровождения могут объединяться в единую корпоративную программу количественных оценок, проводимых в отношении программного обеспечения. Многие организации используют популярный и практичный подход для измерений, базирующийся на оценке количества проблем и статуса их решений (issue-driven measurement). Идеи этого подхода систематизированы в проекте Practical Software and Systems Measurement (PSM). Существуют общие (для всего жизненного цикла) метрики и, соответственно, их категории, в частности, определяемые Институтом Программной Инженерии университета Карнеги-Меллон (Software Engineering Institute, Carnegie-Mellon University – SEI

CMU): размер, усилия, расписание и качество. Применение этих метрик является хорошей отправной точкой для оценки работ со стороны организации, отвечающей за сопровождение. Более детальное обсуждение вопросов измерений в отношении продуктов и процессов представлено в области знаний “Процесс программной инженерии (Software Engineering Process). В свою очередь, вопросы организации программы измерений относятся к области знаний “Управление программной инженерией” (Software Engineering Management).

Специализированные метрики (Specific Measures). Существуют различные методы внутренней оценки продуктивности (benchmarking) персонала сопровождения для сравнения работы различных групп сопровождения. Организация, ведущая сопровождение, должна определить метрики, по которым будут оцениваться соответствующие работы. Стандарты IEEE 1219 (Standard for Software Maintenance) и ISO/IEC 9126-01 (Software Engineering – Product Quality – Part 1: Quality Model, 2001 г.) предлагают специализированные метрики, ориентированные именно на вопросы сопровождения и соответствующие программы.

Типичные метрики оценки работ по сопровождению, соответствующих распространенной классификации эксплуатационных характеристик программного обеспечения:

- анализируемость (Analyzability): оценка (в первую очередь, дополнительных) усилий или ресурсов, необходимых для диагностики недостатков или причин сбоев, а также для идентификации тех фрагментов программной системы, которые должны быть модифицированы;
- изменяемость (Changeability): оценка усилий, необходимых для проведения заданных модификаций;
- стабильность (Stability): оценка случаев непредусмотренного поведения системы, включая ситуации, обнаруженные в процессе тестирования;
- тестируемость (Testability): оценка усилий персонала сопровождения и пользователей по тестированию модифицированного программного обеспечения.

3.3. Процесс сопровождения (Maintenance Process)

Вопросы организации процесса сопровождения напрямую связаны с соответствующими стандартами и de facto практиками реализации такого процесса. Тема “Работы по сопровождению” (Maintenance Activities) различает вопросы сопровождения и разработки и показывает взаимосвязь с другими аспектами деятельности программной инженерии.

Одна из наиболее детально проработанных и распространенных процессных моделей, изначально созданных с ориентацией на программное обеспечение – CMMI (Capability Maturity Model Integration – интегрированная модель зрелости), разработанная в Институте программной инженерии уделяет специальное внимание процессам сопровождения. Существуют и другие, менее распространенные, но тем не менее развивающиеся модели.

Процессы сопровождения (Maintenance Processes). Процессы сопровождения описывают необходимые работы и детальные входы/выходы этих работ. Процесс сопровождения начинается по стандарту IEEE 1219 с момента передачи программной системы в эксплуатацию (post-delivery stage) и касается таких вопросов, как планирование деятельности по сопровождению (см. рисунок 3.2).

Рисунок 3.2 - Работы в процессе сопровождения по стандарту IEEE 1219

Стандарт ISO/IEC 14764 уточняет положения, связанные с процессом сопровождения, стандарта жизненного цикла 12207. Работы по сопровождению, описанные в этом стандарте аналогичны работам в IEEE 1219, за исключением того, что сгруппированы несколько иначе (см. рисунок 3.3).

Рисунок 3.3 - Процесс сопровождения по стандарту ISO/IEC 14764

Работы по сопровождению в стандарте 14764 разбиты на задачи:

- process Implementation – реализация процесса;
- problem and Modification Analysis – анализ проблем и модификаций;
- modification Implementation – проведение модификаций (реализация изменений);
- maintenance Review/Acceptance – оценка и принятие проведенных работ при сопровождении;
- migration – миграция (на модифицированную или новую версию программного обеспечения);
- software Retirement – вывод из эксплуатации (прекращение эксплуатации программного обеспечения).

В SWEBOK можно найти описание истории эволюции соответствующих процессных моделей упоминаемых стандартов ISO/IEC и IEEE. Кроме того, существует и общая (обобщенная) модель процессов сопровождения. Agile-методологии, активно развивающиеся в последние годы, предлагают “облегченные” (light или lightweight) процессы, в том числе, для организации деятельности по сопровождению, например, Extreme maintenance.

3.4. Техники сопровождения (Techniques for Maintenance)

Данная группа вводит некоторые общепринятые техники, используемые в процессе сопровождения программных систем: понимание программных систем (Program Comprehension); реинжиниринг (Reengineering); обратный инжиниринг (Reverse engineering).

Понимание программных систем (Program Comprehension). Для реализации изменений программисты тратят значительную часть времени на чтение и формирование понимания программного продукта. Средства работы с кодом являются ключевым инструментом для решения этой задачи. Четкая, однозначная и лаконичная документация обеспечивает адекватное понимание программных систем.

Реинжиниринг (Reengineering). Реинжиниринг определяется как детальная оценка (examination) и перестройка программного обеспечения для формирования понимания, воссоздания и дальнейшей реализации его функций в новой форме (например, с использованием новых технологий и платформ, при сохранении существующей расширением и облегчением возможностей добавлений новой функциональности. В индустрии существуют различные позиции в отношении реинжиниринга. Считается, что реинжиниринг является наиболее радикальной и затратной формой изменений программных систем, либо, что такой подход может применяться и для не столь кардинальных изменений (например, как смена платформы или архитектуры). Реинжиниринг, обычно, проводится не столько для улучшения возможностей сопровождения (maintainability), сколько для замены устаревшего программного обеспечения. Реинжиниринг

можно рассматривать как самостоятельный проект, включающий в себя формирование концепции, применение соответствующих инструментов и техник, анализ и приложения опыта проведения реинжиниринга, а также оценку рисков и преимуществ, связанных с такими работами. Реализация продукта в новом качестве при сохранении основной функциональности оригинального продукта, является неотъемлемой и определяющей частью реинжиниринга, в отличие от обратного инжиниринга, рассматриваемого ниже и являющегося важной составной частью реинжиниринга.

Обратный инжиниринг (Reverse engineering). «Обратный» инжиниринг, это процесс анализа программного обеспечения с целью идентификации программных компонент и связей между ними, а также формирования представления о программном обеспечении, с дальнейшей перестройкой в новой форме (уже, в процессе реинжиниринга).

Обратный инжиниринг является пассивным, предполагая отсутствие деятельности по изменению или созданию нового программного обеспечения. Обычно, в результате усилий по обратному инжинирингу создаются модели вызовов (call graphs) и потоков управления (control flow graphs) на основе исходного кода системы. Один из типов обратного инжиниринга – создание новой документации на существующую систему (redocumentation). Другой из распространенных типов – восстановление дизайна системы (design recovery). К вопросам обратного инжиниринга, как и к вопросам реинжиниринга, также относятся работы по рефакторингу. Рефакторинг – трансформация программного обеспечения, в процессе которой программная система реорганизуется (не переписываясь) с целью улучшения структуры, без изменения поведения. Сохранение «формы» (платформы, архитектурных и технологических решений) существующей программной системы позволяет рассматривать рефакторинг как один из вариантов обратного инжиниринга.

Соответствующая структура может быть организована следующим образом:

– формирование представления об эксплуатируемой/ сопровождаемой системе: включает восстановление, в первую очередь, бизнес- и функциональных требований к системе:

– восстановление детального дизайна системы: включает идентификацию всех компонентов системы и создание модели вызовов и других связей между компонентами;

– рефакторинг: как возможный процесс структурных изменений, вносимых в систему, в частности для улучшения возможностей по ее дальнейшему сопровождению (включая модификацию, связанную с расширением функциональности);

– переработка системы: создание нового релиза/версии системы в той же форме (например, с использованием той же технологической платформы), что и текущая (эксплуатируемая) версия;

– создание новой системы: рассматривает текущую версию и систему в целом, как устаревшую – legacy.

Программирование является бурно развивающейся областью, поэтому программист должен уметь быстро адаптироваться к текущему состоянию технологии и постоянно изучать новые технологии. Способность к самообучению — один из главных навыков, которым должен обладать программист. Данное пособие посвящено области программной инженерии, которая описывает промышленные технологии, методы, этапы и приемы создания программных продуктов.

Библиографический список

1. Информатика. Программная документация: учеб. пособие / А.А. Ренсков, А.С. Родионов, А.Ю. Чижов, И.П. Янченко; Гос. образовательное учреждение высшего профессионального образования "НВВКУС (Военный ин-т)"; - Новочеркасск: Изд-во НВВКУС, 2010. 253 с.
2. Кудинов Ю.И. Основы современной информатики [Электронный ресурс] : учебное пособие / Ю.И. Кудинов, Ф.Ф. Пащенко. – Электрон. Дан. – СПБ. : Лань, 2011. – 256 с. Режим доступа: http:e//lanbook.com/book/element.php?pl1_id= 68468-Загл. с экрана.
3. Кудинов Ю.И. Практикум по основам современной информатики [Электронный ресурс] : учебное пособие / Ю.И. Кудинов, Ф.Ф. Пащенко, А.Ю. Келина. – Электрон. Дан. – СПБ.: Лань, 2011. – 351 с. Режим доступа: http:e//lanbook.com/book/element.php?pl1_id= 68471-Загл. с экрана.
4. Крылов Е.В. Техника разработки программ: учебник для вузов : в 2 кн.: Кн. 2. / Е.В. Крылов, В.А. Острейковский, Н.Г. Типикин. – М.: Высш. шк. 2008. – 469 с.
5. Иванченко А. Н. Основы программирования: учеб. пособие / А. Н. Иванченко, П. А. Иванченко, А. А. Масленников ; Юж-Рос. гос. политехн. ун-т (НПИ) им. М.И. Платова. – Новочеркасск : ЮРГПУ (НПИ), 2014. - 108 с. – Режим доступа: http://lib/npi-tu.ru
6. Гринченков Д. В. Основы программирования в среде Windows на WinAPI: учеб. пособие / Д. В. Гринченков, П. В. Шлыков ; Юж-Рос. гос. политехн. ун-т (НПИ) им. М.И. Платова. – Новочеркасск : ЮРГПУ (НПИ), 2014. - 106 с. – Режим доступа: http://lib/npi-tu.ru
7. Иванченко А. Н. Функциональное и логическое программирование: учеб. пособие для вузов / А. Н. Иванченко, Д. В. Гринченков, С. И. Потоцкий; ЮРГТУ(НПИ). - 2-е изд., испр.. - Новочеркасск: Изд-во ЮРГТУ(НПИ), 2006. - 240 с.

Учебное издание

Гринченков Дмитрий Валерьевич
Дьяченко Владимир Борисович
Кущий Дарья Николаевна

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Учебное пособие

Редактор *Н.А. Юшко*

Подписано в печать 02.09.2016
Формат 60x841/16. Бумага офсетная. Печать цифровая.
Усл. печ. л. 5,35. Уч. изд. л. 5,5. Тираж 100 экз. Заказ _____.

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова.
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428. Г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ПАРАРАДИГМЫ ПРОГРАММИРОВАНИЯ И КОНЦЕПЦИИ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Методические указания к практическим занятиям и выполнению лабораторных и самостоятельных работ

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 004.43 (076.5)
ББК 32.973.2

Рецензент – кандидат технических наук, доцент кафедры «Программное обеспечение вычислительной техники» **Р.М. Синецкий**

Гринченков Д.В., Кущий Д.Н.

Парадигмы программирования и концепции языков программирования: методические указания к практическим занятиям и выполнению лабораторных и самостоятельных работ / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2017. – 20 с.

Издание содержит методические материалы по освоению навыков описания предметной области в виде фактов и правил на языке Visual Prolog и фреймов. Рассмотрены вопросы построения семантических сетей и организации параллельных вычислений. Приведены задачи для практических занятий, программа лабораторных работ и варианты заданий к ним, а также вопросы для проверки знаний.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника и 09.03.04 Программная инженерия.

УДК 004.43 (076.5)
ББК 32.973.2

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Описание предметной области в терминах языка логического программирования Prolog. Построение схемы логического вывода	4	Тест	20-25.02	[6, 7]
2	Представление знаний в виде фреймов	4	Тест	20-25.02	[6, 7]
3	Представление знаний в виде семантических сетей	4	Тест	20-25.03	[6, 7]
4	Парадигмы и языки параллельного программирования	4	Устный опрос	20-25.03	[6, 7]

Для студентов заочной формы обучения учебным планом предусмотрена дисциплина «Концепции языков программирование». Практические занятия проводятся по темам 1 и 2. На каждое занятие отводится 1 час.

Практическое занятие №1

Описание предметной области в терминах языка логического программирования Prolog. Построение схемы логического вывода

1. Прочитайте следующие факты и опишите предметную область на естественном языке. Определите арность используемых предикатов: девочка («Лиза»); читает («Лиза», «книга»); смотрит («папа», «телевизор», «хоккей»).
2. Запишите по правилам синтаксиса языка Пролог следующие утверждения: роза красная; хобби Ольги – чтение; Ивану 8 лет; Хельсинки – столица Финляндии.
3. Опишите факты по предложенной базе знаний «Золотой ключик»:
   3.1. Категория «взрослые»: шарманщик Папа Карло, столяр Джузеппе, хозяин театра Карабас-Барабас, продавец пиявок Дуремар.

3.2. Категория «животные»: лиса Алиса, собака Артемон, кот Базилио, черепаха Тортила.

3.3. Категория «куклы»: Буратино, Пьеро, Мальвина, Арлекин.

3.4. Категория «плохой»: Карабас-Барабас, Дуремар.

Укажите ответы на следующие вопросы:

– Перечислить взрослых персонажей.
– Кем был папа Карло?
– Какие из персонажей являются куклами?
– Как звали лису?

4. Дано описание предметной области по сказке «Теремок»:

живет («муха», «горюха»).
живет («комар», «пискун»).
живет («мышка», «погрызуха»).
живет («лягушка», «квакушка»).
живет («заюнок», «кривоног»).
живет («лиса», «краса»).
живет («волк», «хватыш»).
не_живет («медведь», «пригнетыш»).

Укажите ответы на следующие вопросы: живет («мышка», «погрызуха»); живет («волк», X); живет (X, «кривоног»); не_живет (M, P).

Сформулировать на Прологе следующие выражения: живет ли лягушка в теремке; какое прозвище у лисы; кто имеет прозвище горюха?

Какой следует задать вопрос, чтобы узнать обитателей теремка (без прозвищ)?

5. Запишите по правилам синтаксиса языка Пролог следующие предложения:

5.1. Студенты допускаются к экзамену, если они сдали зачеты и защитили курсовой проект.

5.2. Беру с собой зонт, если на улице пасмурно или идет дождь.

5.3. Лена и Лариса сокурсницы, если они погодки и учатся в университете.

5.4. Любой удачливый или старательный студент может сдать все экзамены.

Практическое занятие №2
Представление знаний в виде фреймов

Запишите по правилам синтаксиса языка Пролог основные операции над фреймами, представленными в виде структуры («имя фрейма», «список атрибутов»). При этом атрибуты также представляются в виде структуры («имя атрибута», «список значений»), а значение представляется структурой («тип значения», «собственно значение»). С использованием разработанных предикатов представления отношений описать не менее 10 объектов заданной предметной области. Описать правила, позволяющие получить представления обо всех атрибутах всех имеющихся объектов. Выполнить графическое представление описываемого фрейма.

1. Опишите с помощью фреймов ситуацию «Читатель пришел записываться в библиотеку».
2. Опишите с помощью фреймов ситуацию «Пользователь включил компьютер и запустил программу Microsoft Word».
3. Опишите с помощью фреймов процесс выполнения данного практического занятия.

Практическое занятие №3
Представление знаний в виде семантических сетей

1. Разработать семантическую модель представления знаний, содержащихся в следующих событиях:
   1.1. Директор завода «Салют» остановил 10.10.13 цех №4, чтобы заменить оборудование.
   1.2. Если станок закончил обработку, робот грузит кассету с деталями на робокар, который перевозит их на склад.
   1.3. Вася предполагает, что Маша любит мороженое.
2. Построить семантическую сеть для указанной предметной области с учетом приведенных понятий, используя для описания необходимые типы отношений:
   2.1. Предметная область – аэродинамика. Список понятий: самолет, ИЛ-76, крылья, двигатель, керосин, летать, законы аэродинамики, птица, орел, клюв, оперение.

2.2. Предметная область – птицы. Список понятий: птица, животное, канарейка, желтый, петь, крылья, клюв, кожа, дышать, летать.
2.3. Предметная область – млекопитающее. Список понятий: млекопитающее, животное, позвоночник, кошка, шерсть, медведь, рыба, вода, кит.
2.4. Предметная область – транспортное средство. Список понятий: автомобиль, водитель, пассажир, человек, Иванов, Audi, двигатель, стартер, бензин, топливо, октановое число, АИ-98.

3. Запишите по правилам синтаксиса языка Пролог основные виды отношений в семантических сетях, такие как «Является частью», «Принадлежит» и т.д. С использованием разработанных предикатов представления отношений описать не менее 10 объектов заданной предметной области. Описать правила, позволяющие получить представления обо всех объектах в ней и всех связях между ними. Выполнить графическое представление описываемой семантической сети.

3.1. Описать в виде семантической сети отношения основных персонажей сказки «Золотой ключик».
3.2. Описать в виде семантической сети ситуацию «Ученик 9-а класса Иванов И.И. пишет в классе сочинение по произведению А.С. Пушкина «Евгений Онегин»».

Практическое занятие №4
Парадигмы и языки параллельного программирования

1. Разработать и реализовать с помощью библиотеки MPI алгоритм параллельного умножения матрицы на вектор для топологии «кольцо» при условии, что количество строк матрицы и элементов вектора не делится нацело на количество компьютеров.
2. Разработать и реализовать с помощью библиотеки MPI алгоритм параллельного умножения матрицы на вектор для топологии «полный граф».
3. Вычислить сравнительные временные характеристики двух алгоритмов умножения матрицы на вектор для топологий «кольцо» и «полный граф». Обе программы запустить последовательно для матриц A [300][300], A [500][500], A [700][700], A [1000][1000] и соответствующих этим матрицам векторов.

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Знакомство с основами логического программирования. Структура программы на Prolog. Схема логического вывода	2	Защита отчета	20-25.02	[6, 7]
2	Изучения принципов проектирования и создания динамических баз данных в языке Prolog	6	Защита отчета	20-25.02	[6, 7]
3	Повторения и рекурсия в языке Haskell	4	Защита отчета	20-25.03	[6, 7]
4	Работа с файлами, строками и списками в языке Haskell	4	Защита отчета	20-25.03	[6, 7]

Для студентов заочной формы обучения учебным планом предусмотрена дисциплина «Концепции языков программирование». Выполняются лабораторные работы 1 и 4, на проверку каждой из которых отводится по 1 часу в период лабораторно-экзаменационной сессии.

Лабораторная работа №1

Знакомство с основами логического программирования. Структура программы на Prolog. Схема логического вывода

Цель работы: знакомство со средой Visual Prolog, изучение структуры и написание логической программы на языке Prolog.

Программа работы

Составить формальное описание предметной области, в которой имеется *N* объектов (не менее 20) и между ними заданы отношения: родитель, мужчина, женщина, супруги. Реализовать на языке Prolog соответствующую программу. Построить схему логического вывода.

Варианты заданий

1. Определить предикат двоюродный дедушка и найти всех двоюродных дедушек и двоюродных дедушек конкретного лица.
2. Определить предикат двоюродная бабушка и найти всех двоюродных бабушек и двоюродных бабушек конкретного лица
3. Определить предикат двоюродный брат и найти всех двоюродных братьев и двоюродных братьев конкретного лица.
4. Определить предикат двоюродная сестра и найти всех двоюродных сестер и двоюродных сестер конкретного лица.
5. Определить предикат племянник и найти всех племянников и племянников конкретного лица.
6. Определить предикат троюродный брат и найти всех троюродных братьев и троюродных братьев конкретного лица.
7. Определить предикат троюродная сестра и найти всех троюродных сестер и троюродных сестер конкретного лица.
8. Определить предикат внучатый племянник и найти всех внучатых племянников и внучатых племянников конкретного лица.
9. Определить предикат деверь и найти всех деверей и деверей конкретного лица.
10. Определить предикат сват и найти всех сватов и свата конкретного лица.
11. Определить предикат свекровь и найти всех свекровей и свекровь конкретного лица.
12. Определить предикат шурин и найти всех шуринов и шуринов конкретного лица.

Лабораторная работа №2
Изучения принципов проектирования и создания динамических баз данных в языке Prolog

Цель работы: написать программу, позволяющую осуществлять ввод, корректировку и поиск информации. База данных проекта должна храниться в отдельном файле, оформленном по правилам языка Prolog.

Программа работы

Реализовать в среде Visual Prolog Windows-приложение с развитым пользовательским интерфейсом, отвечающее принятым в языке стандартам работы с внешними базами данных.

Варианты заданий

1. Написать программу, моделирующую расписание занятий. Считать, что каждый день проводится по три пары. Должна быть реализована возможность получения следующей информации: по заданному номеру пары и дню недели получить название дисциплины и фамилию преподавателя; по фамилии преподавателя получить название проводимых им пар, время и место их проведения.
2. Написать программу, моделирующую каталог библиотеки. В программе должна быть реализована возможность получения следующей информации: по заданному автору получить название всех его книг, год их издания, издательство, тираж и количество экземпляров каждой из них в библиотеке; определить все книги, изданные в одном году, в одном издательстве.
3. Написать программу для составления топ-листа вышедших в прокат фильмов. В программе должны быть реализованы следующие запросы: какой фильм посетило наибольшее количество зрителей на прошлой неделе? Найти рекордсмена проката среди фильмов с наименьшим бюджетом.
4. Написать программу, моделирующую работу отдела кадров. В программе должна быть предусмотрена возможность получения ответа на запрос: какие специалисты и какой квалификации нужны; выдача всех специальностей, для которых существует возрастной ценз, и его значения; выдача всех специальностей с окладом, больше указанного.
5. Написать программу, моделирующую работу страховой компании. В программе должна быть предусмотрена возможность получения по фамилии клиента информации о дате страховки, номере страхового свидетельства, предмете страховки; выдачи сведений о клиенте и суммы выплат по страховым случаям.
6. Написать программу учета личных данных студентов. Личные данные о студентах. В программе должны быть реализованы следу-

ющие запросы: определить количество студентов на каждом курсе; определить суммарную стипендию группы А факультета В; вывести список групп факультета с указанием численности студентов в каждой группе; найти студентов, не достигших к моменту зачисления 18 лет.

7. Написать программу учета лекарств в аптеках города. В программе должны быть реализованы следующие запросы: определить, какие убытки понесет аптека А, если в течение месяца не реализует все лекарства, у которых кончается срок годности; определить, в каких аптеках дешевле всего «анальгин»; выбрать список лекарств, которые подходят для больного, страдающего болезнями А и В одновременно.

8. Написать программу, описывающую результат сдачи сессии студенческой группой. В программе должна быть предусмотрена возможность получения фамилий всех студентов, имеющих результаты без троек; фамилий всех задолжников с указанием предметов, по которым получены неудовлетворительные оценки.

9. Написать программу, моделирующую работу кредитного отдела банка. В программе должна быть предусмотрена возможность получения следующих данных: какая фирма и когда взяла кредит; названия фирм и суммы, которые следует вернуть в указанном месяце.

10. Написать программу, моделирующую работу справочного бюро автостанции. Должна быть реализована возможность получения следующей информации: по заданному маршруту определить все автобусы и время их отправления; по заданному интервалу времени определить число отходящих автобусов и маршруты их следования.

11. Написать программу, позволяющую вести учет фильмов в пункте проката видеофильмов. В программе должны быть реализованы следующие возможности: поиск необходимого фильма по любой заданной части информации: имя режиссера, название, актеры, страна производитель, киностудия, год выпуска и т.п. Учесть, что по каждому виду может быть задана часть информации.

12. Написать программу, позволяющую вести учет личной музыкальной коллекции. В программе должны быть реализованы следующие возможности: поиск исполнителей, которые сами пишут слова к

песням и музыку. Требуется найти какая песня одного и того же исполнителя записывалась на разных студиях.

13. Фирме принадлежит небольшая флотилия рыболовных катеров. Каждый катер имеет паспорт, куда занесены его название, тип, водоизмещение и дата постройки. Фирма регистрирует каждый выход на лов, записывая название катера, имена и адреса членов команды с указанием их должностей (капитан, боцман и т.д.), даты выхода и возвращения, а также вес пойманной рыбы отдельно по сортам. В программе должны быть реализованы следующие запросы: для заданной банки вывести список катеров, которые получили улов выше среднего; для указанного интервала дат вывести для каждого сорта рыбы список катеров с наибольшим уловом.

Лабораторная работа №3
Повторения и рекурсия в языке Haskell

Цель работы: познакомиться с рекурсией, как с алгоритмическим методом, и освоить способы построения рекурсивных процедур в языке Haskell.

Программа работы

Написать программу на языке Haskell, выполняющую рекурсивную процедуру в соответствии с заданием.

Варианты заданий

1. Вычислить сумму факториалов всех нечетных чисел от 1 до n.
2. Вычислить сумму факториалов всех четных чисел от 2 до n.
3. Напечатать числа, кратные трем, начиная с n до m, $m>n$.
4. Перевести заданное натуральное число в p-ричную систему счисления ($2 < p < 9$).
5. Найти все делители натурального числа n.
6. Найти сумму делителей натурального числа n.
7. Найти произведение наибольшего общего делителя и наименьшего общего кратного трех чисел.
8. Найти сумму первых n членов арифметической прогрессии.
9. Найти сумму первых n членов геометрической прогрессии.
10. Найти первые n членов ряда Фибоначчи.

11. Определить является ли заданное натуральное число простым.
12. Определить правила перевода числа из десятичной системы счисления в двоичную.

Лабораторная работа №4
Работа с файлами, строками и списками в языке Haskell

Цель работы: приобрести навыки работы со строками, файлами, списками, необходимыми для обработки естественно – языковых конструкций.

Программа работы

Реализовать на языке Haskell консольное приложение для обработки естественно-языковых конструкций. Задание включает освоение навыков работы с файловой системой.

Варианты заданий

1. Считать из файла слова (каждое слово записано на отдельной строке), определить число гласных и согласных букв в каждом слове и результаты записать в два списка. По окончании работы списки сохранить в новом файле.
2. Считать из файла некоторый текст, все слова текста разделить на два списка, в первый войдут слова, начинающиеся с гласной, во второй – с согласной. Полученные списки сохранить в некотором файле.
3. Считать из файла слова и разделить их на два списка по следующему правилу: слово попадает в первый список, если в нем больше гласных букв, и во второй, если больше согласных. Полученные списки сохранить в некотором файле.
4. Считать из файла текст и выполнить его синтаксический анализ, определив, сколько использовано гласных букв, результат записать в некоторый файл.
5. Считать слова из файла. Если число согласных букв в слове четное, то его записывают в один список, если нечетное, – в другой. Результаты сохранить в файле.

6. В файле записаны слова, некоторые из них повторяются. Требуется считывать слова из этого файла и формировать из них список, в котором повторов не будет. Результат записать в файл.
7. Из заданного списка переписать в новый список только те слова, у которых одинаковое число гласных и согласных букв.
8. Задан список слов, следует считывать слова и разбивать их на слоги, результаты разбиения каждого слова записывать в файл по формату: «слог–слог–...». Правило разбиения на слоги следующее: слог представляет из себя комбинацию «согласная–гласная» или «гласная–согласная–гласная».
9. Написать программу, которая будет обрабатывать арифметические выражения (выражение не содержит скобок и использует только операции +, –, *, /). Исходную информацию следует брать из файла, а результат заносить в список.
10. Задан список строк, каждая строка представляет собой комбинацию букв и цифр (например, ab25c539fh7). Надо для каждой строки найти сумму цифр, в нее входящих. Результаты записать в файл.
11. Задан список, состоящий из двух списков равной длины, первый из которых состоит из строк, второй – из целых чисел. Требуется построить список без вложений, в который войдут только те строки исходного, которые в списке чисел на соответствующих местах имеют числа, равные числу гласных символов в соответствующей строке.
12. Задан список списков чисел, требуется рассортировать его на несколько списков по следующему принципу: в один список попадают только четные числа, в другой – нечетные, в третий – вещественные. Объединить списки результатов в один список, и результат сохранить в файле.

2. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

Самостоятельная работа студентов предполагает конспектирование основных вопросов тем, указанных в нижеприведенных таблицах.

Очная форма обучения

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 95,2 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 1. Рекурсивные типы данных в языке Prolog.** Бинарное дерево, упорядоченное дерево, двоичные справочники. Примеры применения	10	[1-5]
2	**Тема 2. Среда Visual Prolog.** Создание приложений с графическим интерфейсом в среде Visual Prolog	15,2	[1-5]
3	**Тема 3. Разработка экспертных систем на языке Prolog.** Обзор компонент. Метаинтерпритаторы. Пример ответа на вопрос «Как?» и «Почему?» Выбор направления поиска с помощью эвристики. Вывод в условиях неопределенности	8	[1-5]
4	**Тема 4. Перспективы развития логического программирования.** Перспективы развития методов математической логики для решения задач спецификации и верификации программно-аппаратных средств, создания систем искусственного интеллекта и Семантического Web	10	[1-5]
5	**Тема 5. Функции высшего порядка в языке Haskell.** Понятие функции высшего порядка. Карринг, частичная параметризация и операторные секции. Композиция функций. Стандартные функции: takeWhile, dropWhile, foldr, map, filter. Особенности применения указанных функций	12	[1-5]
6	**Тема 6. Монады.** Понятие монады. Использование монад для реализации хранения состояния. Понятие и определение функции-парсера. Основные парсеры. Организация последовательного и выборочного применения парсеров. Организация повторного применения парсеров. Парсеры для лексем	16	[1-5]
7	**Тема 7. Функциональное и императивное программирование в пакетах прикладного математического ПО.** Scilab. FreeMat. GNU Ocnave. Работа с матрицами, управление потоком, линейная алгебра, интегрирование, дифференциальные уравнения.	12	[1-5]
8	**Тема 8. Мультипарадигмальный язык программирования Python.** Применение в задачах	12	[1-5]

№	Наименование тем	Кол-во часов	Литература
	прикладной математики и в web-программировании. Управление потоком, интроспекция. Передача аргументов. Встроенные функции. getattr(). Фильтрация списков. Логические «и» и «или». Лямбда-функции. Итераторы и генераторы. Библиотеки для прикладных математических задач: NumPy, SciPy.		

Заочная форма обучения

Для студентов заочной формы обучения учебным планом предусмотрена дисциплина «Концепции языков программирование».

СРС – темы и (или) разделы тем для самостоятельного изучения – 129,4 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 1. Основы логического программирование на языке Prolog.** Логика предикатов и Prolog. Структура программы. Факты, правил и цели. Объекты и отношения. Стандартные типы доменов. Предикаты. Свободные, связанные и анонимные переменные. Унификация. Арифметические операции, операции отношения. Повторение и рекурсия. Списки, работа со списками, стандартные приемы обработки списков. Организация ввода-вывода, стандартные предикаты для чтения и записи. Файлы, работа с файлами. Обработка строк и предикаты преобразования типа. Динамические базы данных. Рекурсивные типы данных. Бинарное дерево, упорядоченное дерево, двоичные справочники. Примеры применения	20	[1-5]
2	**Тема 2. Среда Visual Prolog.** Список тем для конспектирования совпадает с вопросами для очной формы обучения	10	[1-5]
3	**Тема 3. Разработка экспертных систем на языке Prolog.** Список тем для конспектирования совпадает с вопросами для очной формы обучения	10	[1-5]

№	Наименование тем	Кол-во часов	Литература
4	**Тема 4. Перспективы развития логического программирования.** Список тем для конспектирования совпадает с вопросами для очной формы обучения	10	[1-5]
5	**Тема 5. Основы функционального программирования на языке Haskell.** Ключевые отличия от императивного программирования. Язык программирования Haskell. Типы и структуры данных. Операции в Haskell. Функции. Лямбда-нотация. Рекурсия. Списки, работа со списками, базовые функции обработки списков. Обработка строк. Компиляторы и среды разработки.	20	[1-5]
6	**Тема 6. Функции высшего порядка в языке Haskell.** Список тем для конспектирования совпадает с вопросами по теме 5 для очной формы обучения	15	[1-5]
7	**Тема 7. Монады.** Список тем для конспектирования совпадает с вопросами по теме 6 для очной формы обучения	15	[1-5]
8	**Тема 8. Функциональное и императивное программирование в пакетах прикладного математического ПО.** Список тем для конспектирования совпадает с вопросами по теме 7 для очной формы обучения	9,4	
9	**Тема 9. Мультипарадигмальный язык программирования Python.** Список тем для конспектирования совпадает с вопросами по теме 8 для очной формы обучения	20	

Зачетные вопросы по курсу

1. Основная идея логического программирования. Область применения. История возникновения и развития.
2. Логические основы языка Prolog и понятие логической программы.
3. Понятие логической программы, факта и правила. Типы переменных в языке Prolog. Анонимные переменные. Примеры реализации.

4. Объявление фактов и правил. Режимы детерминизма фактов и правил. Потоки параметров.
5. Структура программы на Prolog и ее разделы. Основная терминология
6. Стандартные домены языка Prolog. Способы описания параметров предикатов.
7. Составные и альтернативные домены в Prolog. Способы описания и использования.
8. Основные механизмы машины вывода в языке Prolog. Дерево поиска цели.
9. Способы реализации повторяющихся действий в Prolog. Использование повторения. Отрицание и откат. Факт-переменная. Примеры реализации.
10. Способы реализации повторяющихся действий в Prolog. Использование повторения. Динамическое и статическое отсечение. Примеры реализации.
11. Способы реализации повторяющихся действий в Prolog. Метод повторения, определяемый пользователем. Обобщенное правило рекурсии. Примеры реализации.
12. Способы реализации повторяющихся действий в Prolog. Восходящая и нисходящая рекурсия. Примеры реализации.
13. Списки в Prolog. Описание, установление связи между различными списками, доступ к элементам списка.
14. Списки в Prolog. Предикат findall и конструкция [… || …].
15. Списки в Prolog. Сортировка списков. Принадлежность элемента списку.
16. Списки в Prolog. Слияние и разделение списков.
17. Обработка строк в Prolog. Стандартные предикаты работы со строками.
18. Организация ввода-вывода в Prolog. Работа с файлами.
19. Динамические базы данных в Prolog. Связь между реляционной моделью и моделью представления данных в базе данных Prolog. Стандартные предикаты для работы с динамическими базами данных.
20. Соответствие между функциональными и императивными программами. Императивный язык. Формальное описание семантики через интерпретатор императивного языка.

21. Функциональные эквиваленты императивных программ. Преобразование императивных программ в функциональные.
22. Функциональный язык Haskell. Значения и типы. Полиморфные типы. Типы, определяемые пользователем. Бинарные конструкторы данных. Рекурсивные типы. Синонимы типов. Встроенные типы.
23. Функциональное программирование. Понятие число функционального языка.
24. Функциональное программирование. Представление и интерпретация программ. Абстрактная и контрольная формы функциональной программы.
25. Функциональное программирование. Сопоставление с образцом. As-образцы. Универсальные образцы. Семантика сопоставления с образцом. Выражение case. Ленивые образцы.
26. Функциональное программирование. Лексическая видимость и вложенные формы. Let-выражения. Предложение where.
27. Функциональный язык Haskell. Понятие монады и его использование для ввода-вывода.
28. Функциональное программирование. Рекурсивные функции и лямбда-исчисление А.Черча.

Контактная внеаудиторная работа

Очная форма обучения:

СРС – групповые консультации с преподавателем в течение семестра – 0,8 ч.

Заочная форма обучения:

СРС – групповые консультации с преподавателем во время лабораторно-экзаменационной сессии – 0,6 ч.

СРС зач. – сдача зачета с оценкой – 0,25 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Рогозин О.В. Функциональное и рекурсивно-логическое программирование [Электронный ресурс]. – Электрон. дан. – М.: Евразийский открытый институт, 2009. – 139 с. – Режим доступа: http://www.knigafund.ru/books/185891 – Загл. с экрана.
2. Рублев В.С. Языки логического программирования [Электронный ресурс]. – Электрон. дан. – М.: Изд-во «Интернет-Университет

информационных технологий», 2008. – 115 с. – Режим доступа: http://www.knigafund.ru/books/177227 – Загл. с экрана.
3. Городняя Л. В. Основы функционального программирования [Электронный ресурс]. – Электрон. дан. – М.: Изд-во «Интернет-Университет Информационных Технологий», 2004. – 217 с. – Режим доступа: http://www.knigafund.ru/books/177760 – Загл. с экрана.
4. Сузи Р. А. Язык программирования Python [Электронный ресурс]. – Электрон. дан. – М.: Изд-во «Интернет-Университет Информационных Технологий», 2007. – 327 с. – Режим доступа: http://www.knigafund.ru/books/178190 – Загл. с экрана.
5. Лавлинский В. В., Коровина О. В. Технология программирования на современных языках программирования [Электронный ресурс]. – Электрон. дан. – Воронеж: Воронежская государственная лесотехническая академия, 2012. – 118 с. – Режим доступа: http://www.knigafund.ru/books/187182 – Загл. с экрана.
6. Шрайнер П.А. Основы программирования на языке Пролог [Электронный ресурс]. – Электрон. дан. – М.: Изд-во «Интернет-университет информационных технологий ИНТУИТ.ру», 2005. – 176 с. – Режим доступа: http://www.knigafund.ru/books/178287 – Загл. с экрана.
7. Мейлахс А.Л. Практикум по математическим основам информатики: Метод. указания. Ч.2.: Введение в математическую логику [Электронный ресурс]. – Электрон. дан. – М.: Горная книга, 2004. – 73 с. – Режим доступа: http://www.knigafund.ru/books/177682 – Загл. с экрана.

Учебно-методическое издание

Гринченков Дмитрий Валерьевич
Кущий Дарья Николаевна

Парадигмы программирования и концепции языков программирования

Методические указания к практическим занятиям и выполнению лабораторных и самостоятельных работ

Редактор *Н.А.Юшко*
Подписано в печать 27.07.2017.
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,16. Уч.-изд.л. 1,25. Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК

Методические указания
к практическим занятиям, лабораторным
работам, курсовой и самостоятельной
работе студентов бакалавриата
направления подготовки
«Математическое обеспечение и администрирование информационных систем»,
«Информатика и вычислительная техника»
и «Программная инженерия»
(очная и заочная формы обучения)

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 681.3.06 (076.5)

Рецензент

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Панфилов Александр Николаевич

Шайда А.Ю.

Программирование приложений с использованием библиотек: методические указания к практическим занятиям, лабораторным работам курсовой и самостоятельной работе студентов бакалавриата направления подготовки «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника» и «Программная инженерия» (очная и заочная формы обучения)/ Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 48с.

Приведены методические рекомендации к выполнению практических заданий, лабораторных работ и курсовой работы. Рассмотрены основные возможности и особенности стандартной библиотеки шаблонов языка C++, построение приложений с использованием контейнеров, алгоритмов и итераторов STL. Представлено решение задач с применением стандартной библиотеки шаблонов языка программирования C++. Предложены темы для самостоятельного изучения.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия (очная и заочная формы обучения).

УДК 681.3.06 (076.5)

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов*	Форма контроля	Литература
1	Последовательные контейнеры стандартной библиотеки шаблонов	6/2	Тест	[1-5]
2	Ассоциативные контейнеры стандартной библиотеки шаблонов	4/2	Тест	[1-5]
3	Итераторы стандартной библиотеки шаблонов	4/2	Тест	[1-5]
4	Алгоритмы стандартной библиотеки шаблонов	4/2	Тест	[1-5]

*Объем в часах для очной/заочной формы обучения.

Практическое занятие № 1

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ СТАНДАРТНОЙ БИБЛИОТЕКИ ШАБЛОНОВ

1. Что выдаст на консоль следующий фрагмент кода?

```
vector<int> v{2,3,4,5,6};
cout << v.capacity() << ' ' << v.size() << '\n';
v.push_back(7);
cout << v.capacity() << ' ' << v.size() << '\n';
v.shrink_to_fit();
cout << v.capacity() << ' ' << v.size() << '\n';
```

2. Что выдаст на консоль следующий фрагмент кода?

```
vector<int> v{2,3,4,5,6};
for (auto i = v.rbegin(); i < v.rend(); i++) cout << *i << ' ';
```

3. Что выдаст на консоль следующий фрагмент кода?

```
vector<int> v{2,3,4,5,6};
for (auto i = v.end(); i > v.begin();) cout << *(--i) << ' ';
```

4. Что выдаст на консоль следующий фрагмент кода?

```
list<pair<int,string>> v{{1,"one"}, {2,"two"}};
v.emplace(v.begin(), 3, "three");
for (auto x : v) cout << x.second << ' ';
```

5. Что выдаст на консоль следующий фрагмент кода?

```
list<pair<int,string>> v{{1,"one"}, {2,"two"}};
v.insert(v.begin(), make_pair(3, "three"));
for (auto x : v) cout << x.second << ' ';
```

6. Что выдаст на консоль следующий фрагмент кода?

```
list<int> v{1,2,3,4,5,6,7,8};
v.remove_if([](int k){return k%2 == 1;});
for (auto x : v) cout << x << ' ';
```

7. Что выдаст на консоль следующий фрагмент кода?

```
list<int> v{1,2,3,4,5,6,7,8};
v.sort([](int k1, int k2){return k2<k1;});
for (auto x : v) cout << x << ' ';
```

Практическое занятие № 2

АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ СТАНДАРТНОЙ БИБЛИОТЕКИ ШАБЛОНОВ

1. Что выдаст на консоль следующий фрагмент кода?

```
set<int,bool (*) (int,int)> h([](int k1, int k2){return k2<k1;});
h = {1,2,3,4,5};
for (auto x : h) cout << x << ' ';
```

2. Что выдаст на консоль следующий фрагмент кода?

```
struct S {bool operator()(int k1, int k2){return k2<k1;}};
set<int,S> d {1,2,3,4,5};
for (auto x : d) cout << x << ' ';
```

3. Что выдаст на консоль следующий фрагмент кода?

```
auto ff = [](int k1, int k2){return k2<k1;};
int z[]{3, 5, 7, 9};
set<int, bool (*) (int,int)> d(z, z+4, ff);
for (auto x : d) cout << x << ' ';
```

4. Что выдаст на консоль следующий фрагмент кода?

```
template<class T>
struct S {bool operator()(T k1, T k2) const {return k2<k1;}};
map<string, int, S<string>> M;
```

```
M = {{"red", 625}, {"orange", 590}, {"yellow", 565},
{"green", 500}};
for (auto x : M) cout << x.second << ' ';
```

5. Что выдаст на консоль следующий фрагмент кода?

```
template<class T>
struct S {bool operator()(T k1, T k2) const {return k2<k1;}};
map<string, int, S<string>> M;
M = {{"red", 625}, {"orange", 590}, {"yellow", 565},
{"green", 500}};
M.emplace("blue", 485); M["dark_blue"] = 440;
for (auto x : M) cout << x.second << ' ';
```

6. Что выдаст на консоль следующий фрагмент кода?

```
map<string, int> M;
M = {{"red", 625}, {"orange", 590}, {"yellow", 565},
{"green", 500}};
M.emplace("blue", 485); M["dark_blue"] = 440;
for (auto x : M) cout << x.second << ' ';
```

7. Что выдаст на консоль следующий фрагмент кода?

```
map<string, int> M;
M = {{"red", 625}, {"orange", 590}, {"yellow", 565},
 {"green", 500}};
M.emplace("blue", 485); M["dark_blue"] = 440;
auto it = M.find("green");
M.erase(++it); M.erase("red");
for (auto x : M) cout << x.second << ' ';
```

Практическое занятие № 3

ИТЕРАТОРЫ СТАНДАРТНОЙ БИБЛИОТЕКИ ШАБЛОНОВ

1. Что выдаст на консоль следующий фрагмент кода?

```
void PrintVal(int num) {  cout << num << " "; }
int main() {
  int init[] = { 1, 2, 3, 4, 5 };
      auto _f = for_each(init, init + 5, PrintVal);
        _f(100);
  return 0;
}
```

2. Что выдаст на консоль следующий фрагмент кода?

```
int main() {
  int init1[] = { 1, 2, 3, 4, 5 };
  int init2[] = { 6, 7, 8, 9, 10 };
  vector <int> v(10);
     merge(init1, init1 + 5, init2, init2 + 4, v.begin());
      copy(v.begin(), v.end(), ostream_iterator<int>(cout, ", "));
   return 0;
}
```

3. Что выдаст на консоль следующий фрагмент кода?

```
int main() {
    int init[] = { 1, 2, 3, 4, 5 };
   replace(init, init + 5, 0, 2);
      replace(init, init + 5, 1, 0);
        replace(init, init + 5, 2, 1);
  copy(init, init + 5, ostream_iterator<int>(cout, " "));
  return 0;
}
```

4. Что выдаст на консоль следующий фрагмент кода?

```
int main() {
   int init[] = { 1, 2, 3, 4, 5 };
      reverse(init, init + 5);
     copy(init, init + 5, ostream_iterator<int>(cout, " "));
    return 0;
}
```

5. Что выдаст на консоль следующий фрагмент кода?

```
int main() {
      const int init[] = { 1, 2, 3, 4, 5 };
      vector<int> v(5); vector<int>::iterator it;
      copy(init, init + 5, it = v.begin());
      cout << *(it + 4) << endl;
      cout << *(it += 3) << endl;
      cout << *(it -= 1) << endl;
      cout << *(it = it - 1) << endl;
      cout << *(--it) << endl;
      for (int i = 0; i < (v.end() - v.begin()); i++)
            cout << it[i] << " ";
}
```

Практическое занятие № 4

АЛГОРИТМЫ СТАНДАРТНОЙ БИБЛИОТЕКИ ШАБЛОНОВ

1. Что выдаст на консоль следующий фрагмент кода?

```
struct S {
   int k {1}; int operator()() {return k*=3;}
} fs;

int main () {
   list<int> f(8);
   generate(f.begin(), f.end(), fs);
   if ( all_of(f.begin(), f.end(), [](int i){return i%2;}) )
      cout << "Все элементы - нечетные числа.\n";
   if ( any_of(f.begin(), f.end(), [](int i){return i%3 == 0;}) )
      cout << "Некоторые элементы делятся на 3.\n";
   if ( none_of(f.begin(), f.end(), [](int i){return i>50;}) )
      cout << "Нет элементов больше 50.\n";
   return 0;
}
```

2. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
   list<int> f(8);
   iota(f.begin(), f.end(), 2);
   for_each(f.begin(), f.end(), [](int& k) {return k *= k;});
   if ( all_of(f.begin(), f.end(), [](int i){return i%2;}) )
      cout << "Все элементы - нечетные числа.\n";
   if ( any_of(f.begin(), f.end(), [](int i){return i%3 == 0;}) )
      cout << "Некоторые элементы делятся на 3.\n";
   if ( none_of(f.begin(), f.end(), [](int i){return i>50;}) )
      cout << "Нет элементов больше 50.\n";
   return 0;
}
```

3. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
   vector<int> v {3,7,11,5,6,4,9,8,1,10,0,12};
     auto bound = partition (v.begin(), v.end(), [](int i)
  { return (i%2)==1; });
   for (auto it = v.begin(); it != bound;) cout << ' ' << *(it++);
```

```
    return 0;
}
```

4. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
    vector<int> v {3,7,11,5,6,4,9,8,1,10,0,12};
    auto bound = partition (v.begin(), v.end(), [](int i)
{ return (i%2)==1; });
    for (auto it = bound; it != v.end();) cout << ' ' << *(it++);
    return 0;
}
```

5. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
    vector <int> v {16, 14, 9, 8, 7, 10, 3, 1, 4, 2};
    if(is_heap (v.begin(),v.end())) cout << "Это пирамида\n";
    else cout << "Это не пирамида\n";
    return 0;
}
```

6. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
    vector<pair<int,int>> v {{1,2},{4,3},{5,6},{8,7},{1,9},{7,2}};
        auto bound = partition (v.begin(), v.end(), [](pair<int,int> x)
    { return x.first < x.second; });
    for (auto it = bound; it != v.end();)
        cout << '(' << it->first << ',' << (it++)->second << ") ";
    return 0;
}
```

7. Что выдаст на консоль следующий фрагмент кода?

```
int main () {
    vector<pair<int,int>> v {{1,2},{4,3},{5,6},{8,7},{1,9},{7,2}};
        auto bound = partition (v.begin(), v.end(), [](pair<int,int> x)
     { return x.first < x.second; });
    for (auto it = v.begin(); it != bound;)
        cout << '(' << it->first << ',' << (it++)->second << ") ";
    return 0;
}
```

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Литера-тура
1	Последовательные контейнеры. Массивы	3/1	[1-5]
2	Последовательные контейнеры. Векторы и деки	4/1	[1-5]
3	Последовательные контейнеры. Списки	3/1	[1-5]
4	Ассоциативные контейнеры	4/1	[1-5]
5	Алгоритмы	4/2	[1-5]

*Объем в часах для очной/заочной формы обучения.

Лабораторная работа №1

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ. МАССИВЫ

Цель работы: развитие практических навыков написания программ с использованием массивов стандартной библиотеки шаблонов языка программирования C++.

Пояснения к работе: реализовать согласно вариантам задания, используя последовательный контейнер фиксированного размера *array*<>.

Задания к работе:

1.1. Задан массив из k символов. Преобразовать массив следующим образом: сначала должны стоять цифры, входящие в массив, а затем все остальные символы. Взаимное расположение символов в каждой группе не должно изменяться.

1.2. Задан массив из k символов. Преобразовать массив следующим образом: расположить символы в обратном порядке.

1.3. Задан массив из k чисел. Найти число, наиболее часто встречающееся в этом массиве. Отсортировать элементы массива по возрастанию.

1.4. Задан массив из k чисел. Написать программу, которая находит минимальный элемент целочисленного массива, исключает его и затем включает его в конец того же массива

1.5. Задан массив из k чисел. Найти числа, входящие в массив только один раз. Сдвинуть элементы массива циклически на n позиций влево.

1.6. Задан массив из k чисел. Написать программу, которая удаляет из целочисленного массива одинаковые подряд идущие элементы, оставляя их в массиве по одному

1.7. Задан массив из k чисел. Вычислить произведение нечетных элементов массива, расположенных на четных позициях. Сдвинуть элементы массива циклически на n позиций вправо.

1.8. Задан массив из k чисел. Преобразовать массив следующим образом: все отрицательные элементы массива перенести в начало, а все остальные – в конец, сохранив исходное взаимное расположение как среди отрицательных, так и среди положительных элементов.

1.9. Задан массив из k символов. Создать два новых массива: в первый перенести все цифры из исходного массива, во второй – все остальные символы.

1.10. Задан массив из k символов. Определить, симметричен ли он, т. е. читается ли он одинаково слева направо и справа налево.

1.11. Задано два массива. Найти наименьшие среди элементов первого массива, которые не входят во второй массив.

1.12. Задан массив из k чисел. Определить количество инверсий в массиве (т. е. таких пар элементов, в которых большее число находится слева от меньшего).

1.13. Задан массив из k символов. Определить количество различных элементов в массиве. Удалить из него повторные вхождения каждого символа.

1.14. Задан массив из k символов. Вычислить сумму S первых 20 отрицательных элементов массива, кратных 5. Удалить из массива элементы, значения которых равны по абсолютному значению S.

1.15. Задан массив из k символов латинского алфавита. Вывести на экран в алфавитном порядке все символы, которые входят в этот массив по одному разу.

1.16. Дан целочисленный массив размера N. Назовем серией группу подряд идущих одинаковых элементов, а длиной серии - количество этих элементов (длина серии может быть равна 1). Вывести массив, содержащий длины всех серий исходного массива.

1.17. Дан целочисленный массив размера N. Если он является перестановкой, то есть содержит все числа от 1 до N, то вывести 0, в противном случае вывести номер первого недопустимого элемента.

1.18. Дан массив ненулевых целых чисел размера N. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести 0, если нет, то вывести номер первого элемента, нарушающего закономерность.

1.19. Дан целочисленный массив A размера 10. Вывести номер первого и последнего из тех его элементов A[i], которые удовлетворяют двойному неравенству: A[1] < A[i] < A[10]. Если таких элементов нет, то вывести 0.

1.20. В последовательности действительных чисел a1, a2, ..., an есть только положительные и отрицательные элементы. Вычислить произведение отрицательных элементов P1 и произведение положительных элементов P2. Сравнить модуль P1 модулем P2, указать, какое из произведений по модулю больше.

1.21. Даны целые числа a1, a2, ..., an. Наименьший член этой после-довательности заменить целой частью среднего арифметического всех членов, остальные члены оставить без изменения. Если в последовательности несколько наименьших членов, то заменить последний по порядку.

1.22. Задан целочисленный массив размерности N. Есть ли среди элементов массива простые числа? Если да, то вывести номера этих элементов.

1.23. Дана последовательность из n различных целых чисел. Найти сумму ее членов, расположенных между максимальным и минимальным значениями (в сумму включить и оба этих числа).

1.24. Дан массив целых чисел. Найти в этом массиве минимальный элемент m и максимальный элемент M. Получить в порядке возрастания все целые числа из интервала (m; M), которые не входят в данный массив.

1.25. В одномерном массиве все отрицательные элементы переместить в начало массива, а остальные — в конец с сохранением порядка следования. Дополнительный массив заводить не разрешается.

Лабораторная работа №2

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ. ВЕКТОРЫ И ДЕКИ

Цель работы: развитие практических навыков написания программ с использованием векторов и деков стандартной библиотеки шаблонов языка программирования C++.

Пояснения к работе: реализовать согласно вариантам задания, используя последовательный контейнер *vector*<> или *deque* <>.

Задания к работе:

2.1. Задан вектор и дек из N целых чисел: первый – используя датчик случайных чисел, второй – с клавиатуры. Необходимо найти в каждом из них среднее значение, а также посчитать сумму четных и нечетных элементов.

2.2. Задан вектор из N целых чисел. Заменить все элементы, равные сумме двух соседних элементов (слева и справа), на произведение этих элементов. Найти дополнения этих чисел до максимального.

2.3. Задан вектор из N и дек из M целых чисел, упорядоченными по возрастанию. Объединить элементы этих контейнеров в один так, чтобы элементы оказались упорядоченными по убыванию. Определить сколько чисел не лежат в интервале [a, b] (a и b – заданные числа) и запомнить их порядковые номера.

2.4. Задан вектор из N целых чисел. Определить порядковый номер того элемента, значение которого наиболее отличается от заданного числа. Все встречающиеся последовательности элементов, упорядоченных по возрастанию, отсортировать по убыванию.

2.5. Задан вектор и дек из N целых чисел. Необходимо реализовать для первого сортировку методом Шелла, а для второго – сортировку пузырьковым методом. Объединить векторы в один, и полученный вектор отсортировать по возрастанию любым из методов.

2.6. Если в векторе N целых чисел есть хотя бы один член меньше 10, то все отрицательные числа заменить их квадратами,

в противном случае – домножить все члены на 2. Отсортировать полученный вектор по возрастанию.

2.7. Дек из N целых чисел преобразовать, расположив вначале отрицательные члены, а затем – неотрицательные. При этом порядок отрицательных чисел сохраняется прежним, а порядок неотрицательных чисел изменяется наоборот. Отсортировать полученный дек по убыванию.

2.8. Задан вектор из N целых чисел. Преобразовать эту последовательность, расположить вначале отрицательные члены, а затем – неотрицательные, с сохранением порядка чисел (без сортировки). Удалить из вектор элемент с номером К. Добавить после каждого четного элемента вектора элемент со значением 0.

2.9. Сформировать дек целых чисел, используя датчик случайных чисел. Удалить первый элемент, равный 0. Если такого элемента нет, удалить первый четный элемент. Добавить после каждого четного элемента элемент со значением M[I-1]+2.

2.10. Сформировать вектор целых чисел, используя датчик случайных чисел. Удалить элементы, индексы которых кратны 3. Добавить после элемента вектора с заданным индексом элемент со значением А.

2.11. Задан дек из N целых чисел вводом значений с клавиатуры. Удалить первый элемент с заданным значением. Сдвинуть контейнер циклически на К элементов вправо.

2.12. Сформировать вектор целых чисел, используя датчик случайных чисел. Поменять местами минимальный и максимальный элементы массива. Удалить из вектора все элементы, превышающие его среднее значение более, чем на 10%.

2.13. Сформировать дек целых чисел, используя датчик случайных чисел. Удалить 5 первых элементов дека. Добавить в конец дека 3 новых элемента. Отсортировать полученный дек по убыванию.

2.14. Сформировать дек целых чисел, используя датчик случайных чисел. Удалить из дека все элементы, совпадающие с его минимальным значением. Добавить в начало дека 3 элемента со значением, равным среднему арифметическому всех его элементов. Отсортировать полученный дек по возрастанию.

2.15. Сформировать вектор целых чисел, используя датчик случайных чисел. Перевернуть вектор и, если число элементов

нечетное, удалить его средний элемент. Добавить в конец вектора 3 элемента со значением M[I+10]-2.

2.16. Сформировать с помощью датчика случайных чисел вектор вещественных чисел. Количество элементов вектора и диапазон значений вводится с клавиатуры. Написать функцию, выводящую на экран элементы вектора – в столбик с указанием индекса или в строку через пробел. Формат вывода зависит от параметра, задаваемого с клавиатуры. Отсортировать по убыванию только элементы вектора, расположенные на чётных позициях.

2.17. Даны два неубывающих дека: первый – используя датчик случайных чисел, второй – с клавиатуры. Найти их соединение, то есть неубывающий дек, содержащий все их элементы, причем каждый элемент должен входить в результирующий дек столько раз, сколько он входит в общей сложности в исходные деки.

2.18. Задан вектор из N целых чисел. Найдите наибольший нечетный элемент. Далее трижды осуществите циклический сдвиг влево элементов, стоящих справа от найденного максимума, и один раз сдвиг элементов вправо, стоящих слева от найденного максимума.

2.19. Задан дек из N целых чисел. Найти минимальный по модулю элемент дека. Сумму модулей элементов дека, расположенных после первого элемента, равного нулю. Преобразовать дек таким образом, чтобы в первой его половине расположились элементы, стоящие в чётных позициях, а во второй половине – элементы стоявшие в нечётных позициях.

2.20. Задан вектор из N целых чисел. Найти два соседних элемента, сумма которых максимальна, и вывести эти элементы в порядке возрастания их индексов, а также сумма, которых минимальна, и вывести эти элементы в порядке убывания их индексов.

2.21. Задан вектор из N символов латинского алфавита. Вывести на экран в алфавитном порядке все символы, которые входят в этот вектор по одному разу. Удалить из вектора элемент с номером K. Добавить после каждого гласной буквы вектора элемент со значением A.

2.22. Задан дек из N вещественных чисел. Написать программу, которая записывает в новый вектор и затем удаляет из

исходного дека элемент, если среднее арифметическое его «соседей» меньше 5. Отсортировать полученные дек и вектор по возрастанию. Переписать все элементы вектора в дек добавляя элементы в голову.

2.23. Задан вектор из строк. Ввод слов производите до тех пор, пока не встретится END. Прочитав все слова, обработайте вектор и переведите символы каждого слова в верхний регистр. Отобразите преобразованные элементы по восемь слов на строку.

2.24. Задан вектор из N целых чисел. Отобразите сумму каждой пары соседних элементов. Измените программу так, чтобы она отображала сумму первого и последнего элементов, затем сумму второго и последнего и т.д.

2.25. Напишите программу, инициализирующую вектор значениями из массива целых чисел. Преобразовать вектор следующим образом: все отрицательные элементы вектора перенести в начало, а все остальные – в конец, сохранив исходное взаимное расположение как среди отрицательных, так и среди положительных элементов.

Лабораторная работа №3

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ. СПИСКИ

Цель работы: развитие практических навыков написания программ с использованием двунаправленных и однонаправленных списков стандартной библиотеки шаблонов языка программирования C++.

Пояснения к работе: реализовать согласно вариантам задания, используя последовательный контейнер *list* <> или *forward_list* <>.

Задания к работе:

Создать список с числами в диапазоне от –50 до +50. Отсортировать список и вывести его на экран. Найти элемент списка, равный номеру варианта, и вывести его порядковый номер либо сообщение о том, что такого элемента нет. Выполнить индивидуальное задание согласно вариантам:

3.1. Найти минимальный элемент двунаправленного списка и сделать его первым.

3.2. Преобразовать однонаправленный список в два списка. Первый должен содержать только положительные числа, а второй – отрицательные.

3.3. Создать два двунаправленных списка. Первый должен содержать положительные, а второй – отрицательные числа.

3.4. Создать новый однонаправленный список, содержащий только четные числа из первого списка.

3.5. Удалить из двунаправленного списка элементы, находящиеся между его максимальным и минимальным значениями.

3.6. Создать новый однонаправленный список, содержащий только те числа из первого списка, которые больше среднего значения всех элементов первого списка.

3.7. Переместить во второй двунаправленный список все элементы, находящиеся после элемента с максимальным значением.

3.8. Создать новый однонаправленный список, содержащий только положительные числа из первого списка.

3.9. Сместить элементы двунаправленного списка вперед на заданное число позиций.

3.10. Подсчитать, сколько элементов однонаправленного списка имеют значение, которое превышает среднее значение всех элементов списка.

3.11. Определить, сколько элементов однонаправленного списка находится до элемента с максимальным значением.

3.12. Удалить все отрицательные элементы двунаправленного списка.

3.13. Определить, сколько элементов однонаправленного списка находится до элемента с минимальным значением.

3.14. Переместить во второй двунаправленный список все элементы, находящиеся между максимальным и минимальным элементами. Остальные элементы должны остаться в первом списке.

3.15. Удалить по одному элементу справа и слева от минимального элемента двунаправленного списка. Если элементы справа или слева отсутствуют, то удалить сам минимальный элемент.

3.16. Создать однонаправленный список, порядок следования элементов в котором обратный относительно первого списка.

3.17. Удалить элементы, заключенные между максимальным и минимальным значениями двунаправленного списка.

3.18. Создать новый однонаправленный список, в который поместить каждый третий элемент первого списка.

3.19. Удалить из двунаправленного списка элементы с повторяющимися более одного раза значениями.

3.20. Создать новый однонаправленный список, содержащий только числа, большие среднего значения всех элементов первого списка.

3.21. Поменять местами первый и максимальный элементы, первый и последний элементы списка двунаправленного списка.

3.22. Преобразовать список в два однонаправленных списка. В первый поместить все четные, а во второй – нечетные числа.

3.23. Поменять местами элементы с максимальным и минимальным значениями. Сместить элементы двунаправленного списка назад на заданное число позиций.

3.24. Создать однонаправленный список, в который поместить элементы, лежащие во второй половите первого списка.

3.25. Определить, сколько элементов однонаправленного списка находится между его минимальным и максимальным элементами, имеют значение меньше среднего значения всех элементов списка.

Лабораторная работа №4
АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ

Цель работы: развитие практических навыков написания программ с использованием ассоциативных контейнеров стандартной библиотеки шаблонов языка программирования C++.

Пояснения к работе: реализовать согласно вариантам задания, используя соответствующий тип ассоциативного контейнера.

Задания к работе:

4.1.Создать отображение, инициализировать его десятью значениями (ключ – целочисленный, значение – вещественное) и вывести на экран. Найти элемент с заданным ключом и добавить его в заданную позицию. Найти элемент с заданным ключом и

удалить его из контейнера. Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера.

4.2.Создать множество, инициализировать его десятью целочисленными значениями и с помощью итераторов вывести на экран. Найти минимальный элемент и добавить его в конец множества. Найти элемент с заданным ключом и удалить его из множества. К каждому элементу добавить сумму минимального и максимального элементов множества.

4.3.Создать пустое отображение, заполнить его данными (ключ – фамилия абонента, значение – номер телефона) и вывести на экран. Выполнить поиск в словаре существующего и несуществующего элемента и вывести результат на экран. Выполнить редактирование номера телефона найденного элемента.

4.4.Создать пустое множество вещественных чисел с дубликатами, заполнить его данными и вывести на экран. Найти элемент с заданным ключом и добавить его на заданную позицию. Найти элемент с заданным ключом и удалить его из мультимножества. Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента мультимножества.

4.5.Создать два пустых отображения, заполнить его данными (ключ – фамилия абонента, значение – номер телефона) и вывести на экран. Выполнить обмен словарей и вывести их на экран. Очистить словари.

4.6.Создать пустое множество вещественных чисел, заполнить данными и вывести на экран. Найти минимальный элемент и добавить его на заданную позицию. Найти элементы большие среднего арифметического и удалить их из множества. Каждый элемент домножить на максимальный элемент контейнера.

4.7. Создать пустое мультиотображение (ключ – фамилия абонента, значение – номер телефона). С помощью метода *insert()* заполнить мультиотображение данными, так, чтобы оно содержало дубликаты с одинаковыми номерами. Выполнить поиск в словаре информации с заданными ключами и выведите результаты поиска на экран (ключ с дубликатами, без дубликатов и отсутствующий ключ).

4.8. Создать множество вещественных чисел с дубликатами, инициализировать его десятью значениями и с помощью итераторов вывести на экран. Найти минимальный элемент и добавить его в конец. Найти элемент с заданным ключом и удалить его из мультимножества. К каждому элементу добавить сумму минимального и максимального элементов множества.

4.9. Создать пустое мультиотображение, заполнить его данными (ключ – улица, значение – номер дома) и вывести на экран. Добавить новый элемент в начало контейнера с помощью функции *emplace_hint()*. Выполнить удаление записей из отображения по заданному ключу. Все четные номера заменить нечетными.

4.10. Создать множество вещественных чисел, инициализировать его десятью значениями и с помощью итераторов вывести на экран. Найти максимальный элемент и добавить его в конец множества. Найти элемент с заданным ключом и удалить его из множества. К каждому элементу добавить среднее арифметическое элементов множества.

4.11.Создать пустое отображение, заполнить его данными (ключ – целочисленный, значение – вещественное) и вывести на экран. Найти максимальный элемент и добавить его в конец контейнера. Найти элемент с заданным ключом и удалить его из контейнера. К каждому элементу добавить среднее арифметическое элементов контейнера.

4.12.Создать пустое множество целых чисел с дубликатами, заполнить его данными и вывести на экран. Найти среднее арифметическое и добавить его в начало контейнера. Найти элемент с заданным ключом и удалить его из мультимножества. Из каждого элемента вычесть минимальный элемент контейнера.

4.13. Создать мультиотображение, инициализировать его десятью значениями (ключ – целочисленный, значение – вещественное) и вывести на экран. Найти максимальный элемент и добавить его в конец контейнера. Найти минимальный элемент и удалить его из контейнера. К каждому элементу добавить среднее арифметическое элементов контейнера.

4.14.Создать множество, инициализировать его десятью целочисленными значениями и с помощью итераторов вывести на экран. Найти максимальный элемент и добавить его в конец множества. Найти элемент из заданного диапазона и удалить его

из множества. К каждому элементу добавить сумму минимального и максимального элементов множества.

4.15. Создать пустое мультиотображение, заполнить его данными (ключ – улица, значение – номер дома) и вывести на экран. Найти минимальный элемент и добавить его на заданную позицию. Найти элементы большее среднего арифметического и удалить их. Каждый элемент домножить на максимальный элемент.

4.16. Создать множество вещественных чисел с дубликатами, инициализировать его десятью значениями и с помощью итераторов вывести на экран. Найти минимальный элемент и добавить его в конец. Найти элемент с заданным ключом и удалить его из мультимножества. К каждому элементу добавить сумму минимального и максимального элементов множества.

4.17.Создать пустое множество целых чисел с дубликатами, заполнить его данными и вывести на экран. Найти среднее арифметическое и добавить его в начало контейнера. Найти элемент с заданным ключом и удалить его из мультимножества. Из каждого элемента вычесть минимальный элемент контейнера.

4.18.Создать пустое множество вещественных чисел, заполнить его данными и вывести на экран. Найти минимальный элемент и добавить его на заданную позицию. Найти элементы большие среднего арифметического и удалить их. Каждый элемент домножить на максимальный элемент контейнера.

4.19.Создать множество, инициализировать его десятью целочисленными значениями и с помощью итераторов вывести на экран. Найти минимальный элемент и добавить его в конец множества. Найти элемент с заданным ключом и удалить его из множества. К каждому элементу добавить сумму минимального и максимального элементов множества.

4.20. Создать пустое мультиотображение, заполнить его данными (ключ – улица, значение – номер дома) и вывести на экран. Добавить новый элемент в начало контейнера с помощью функции *emplace_hint()*. Выполнить удаление записей из отображения по заданному ключу. Все четные номера домов заменить на нечетные.

4.21.Создать пустое множество вещественных чисел с дубликатами, заполнить его данными и вывести на экран. Найти

элемент с заданным ключом и добавить его на заданную позицию. Найти элемент с заданным ключом и удалить его из мультимножества. Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента мультимножества.

4.22. Создать пустое мультиотображение (ключ – фамилия абонента, значение – номер телефона). С помощью метода *insert()* заполнить мультиотображение данными, так, чтобы оно содержало дубликаты с одинаковыми номерами. Выполнить поиск в словаре информации с заданными ключами и выведите результаты поиска на экран (ключ с дубликатами, без дубликатов и отсутствующий ключ).

4.23. Создать мультиотображение, инициализировать его десятью значениями (ключ – целочисленный, значение – вещественное) и вывести на экран. Найти максимальный элемент и добавить его в конец контейнера. Найти минимальный элемент и удалить его из контейнера. К каждому элементу добавить среднее арифметическое элементов контейнера.

4.24.Создать пустое отображение, заполнить его данными (ключ – целочисленный, значение – вещественное) и вывести на экран. Найти максимальный элемент и добавить его в конец контейнера. Найти элемент с заданным ключом и удалить его из контейнера. К каждому элементу добавить среднее арифметическое элементов контейнера

4.25.Создать пустое отображение, заполнить данными (ключ – фамилия абонента, значение – номер телефона) и вывести на экран. Выполнить поиск в словаре существующего и несуществующего элемента и вывести результат поиска на экран. Выполнить редактирование номера телефона найденного элемента.

Лабораторная работа №5
АЛГОРИТМЫ

Цель работы: развитие практических навыков написания программ с использованием алгоритмов стандартной библиотеки шаблонов языка программирования C++.

Пояснения к работе: реализовать задания согласно вариантам, применяя только средства стандартной библиотеки языка C++ (потоковый ввод-вывод, объявления стандартных алгоритмов из файла <algorithm> и объявления стандартных функциональных объектов из файла <functional>).

5.1. Напишите законченную программу, в которой создайте и инициализируйте два целочисленных вектора размерами соответственно 4 и 2 элемента. С помощью метода *for_search()* выведите значения их элементов на экран.

5.2. Напишите законченную программу, в которой создайте и инициализируйте целочисленный вектор. Выведите значения его элементов на экран. С помощью метода *find()* выполните поиск в векторе элемента с заданным значением и выведите результаты поиска на экран. С помощью метода count() подсчитайте в векторе количество элементов с заданным значением и выведите результаты поиска на экран.

5.3. Напишите законченную программу, в которой создайте и инициализируйте целочисленный вектор. Выведите значения его элементов на экран. С помощью метода *find-if()* выполните поиск в векторе первого элемента с нечетным значением и выведите результаты поиска на экран.

5.4. Напишите законченную программу, в которой создайте и инициализируйте целочисленный вектор. Выведите значения его элементов на экран. С помощью метода *adjacent_find-if()* выполните поиск в векторе первой пары элементов со значениями "меньше-больше" и выведите результаты поиска на экран.

5.5. Напишите законченную программу, в которой создайте и инициализируйте два целочисленных вектора *v, v1* размерами соответственно 4 и 2 элемента. Выведите значения их элементов на экран. С помощью метода *find-end()* выполните поиск последнего вхождения *v1* в *v* и выведите результаты поиска на экран, в том числе напечатайте значение найденного элемента. С помощью метода *find-first_of()* выполните поиск первого вхождения в *v* элемента из *v1* и выведите результаты поиска на экран

5.6. Напишите законченную программу, в которой создайте и инициализируйте два целочисленных вектора *v, v1* размерами соответственно 4 и 2 элемента. Выведите значения их элементов на экран. Проверьте *v, v1* на несовпадение и совпадение соответ-

ственно с помощью методов *mismatch()* и *equal()* и выведите результаты проверки на экран. С помощью метода *search()* выполните проверку вхождения в *v1* в *v* и выведите результаты проверки на экран

5.7. Напишите законченную программу, в которой создайте два целочисленных массива *a[5] и b[4]*.Первый массив инициализируйте значениями 1, 2, 3, 4, 5. С помощью алгоритма *copy()* скопируйте последние четыре элемента массива *a* в массив *b*. Выведите значения элементов *b* на экран. С помощью алгоритма *copy()* скопируйте последние четыре элемента массива *a* в его начало и выведите значения элементов *a* на экран.. С помощью алгоритма *copy_backward()* скопируйте первые три элемента массива *b* справа-налево в этот же массив, начиная копирование в четвертый элемент этого же массива. Выведите значения элементов *b* на экран. С помощью алгоритма *copy()* выведите на экран значения элементов массива *a*. С этой целью в качестве третьего аргумента в вызове алгоритма *copy()* используйте потоковый итератор *ostream_iterator< int >(cout, " ")*.

5.8. Напишите законченную программу, в которой создайте целочисленный массив *a[5]*. С помощью алгоритма *fill()* заполните массив единичными значениями и выведите значения элементов массива на экран. С помощью алгоритма *fill_n()* заполните третий и четвертый элементы массива нулевыми значениями и выведите значения элементов массива на экран.

5.9. Напишите законченную программу, в которой создайте целочисленный массив *a[5]*. С помощью алгоритма *generate()* заполните массив значениями, возвращаемыми функцией и выведите значения элементов массива на экран.

5.10. Напишите законченную программу, в которой создайте целочисленные массивы *a[5]={1,2,3,4.5}* и *b[5]={11,12,13,14,15}*. С помощью алгоритма*iter_swap()*поменяйте местами первый элемент *a* и пятый элемент *b* и выведите значения элементов массивов на экран. С помощью этого же алгоритма поменяйте местами первый и последний элементы массива *a* и выведите значения элементов массива *a* на экран. С помощью алгоритма *swap_ranges()*поменяйте местами первые два элемента массива *a* с четвертым и пятым эле-

ментами массива *b* и выведите значения элементов массивов на экран. С помощью этого же алгоритма поменяйте местами первые два элемента массива *a* с его четвертым и пятым элементами и выведите значения элементов массива *a* на экран.

5.11. Напишите законченную программу, в которой создайте целочисленный массив *a[5]={1,2,3,4.5}*. С помощью алгоритма *random_shuffle()*перетасуйте элементы случайным образом и выведите значения элементов массива на экран. С помощью этого же алгоритма перетасуйте часть массива – первые три элемента - и выведите значения элементов массива *a* на экран.

5.12. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два целочисленных вектора *a* – пустой и *b* размером 10 элементов с нулевыми значениями. Занесите в вектор *a* последовательность значений 0, 1, 2, 3, 4, 0, 1, 2, 3, 4. Выведите значения элементов созданных векторов на экран. С помощью алгоритма *remove()* удалите из *a* элементы со значением 2 и выведите значения элементов вектора *a* на экран. С помощью алгоритма *remove_copy()* скопируйте *a* в *b* с удалением в копии элементов со значением 3 и выведите значения элементов обоих векторов на экран. С помощью алгоритма *remove_if()*удалите из *a* элементы со значениями, превышающими единицу. С этой целью в качестве третьего аргумента в вызове алгоритма используйте функцию-предикат Выведите значения элементов вектора *a* на экран.

5.13. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два целочисленных вектора *v1* – пустой и*v1* размером 10 элементов с нулевыми значениями. Занесите в вектор*v1* последовательность значений 10, 20, 30, 40, 50, 60, 70, 80, 90. Выведите значения элементов созданных векторов на экран. С помощью алгоритма *replace_copy_if()*скопируйте*v1* в*v2*с заменой в копии значений больших 30 и меньших 70 на значение 5. С этой целью в качестве третьего аргумента в вызове алгоритма используйте функцию-предикат. Выведите значения элементов обоих векторов на экран.

5.14. Напишите законченную программу, в которой создайте два целочисленных массива *a* и *b* одинакового размера и инициализируйте их. Выведите значения элементов созданных массивов на экран. С помощью алгоритма *transform()*выполните попарную

обработку элементов этих массивов в соответствии с выражением $a_i = a_i^2 - b_i^2$. С этой целью в качестве пятого аргумента в вызове алгоритма используйте функциональный объект. Выведите значения элементов массива *a* на экран. С помощью алгоритма *transform()* выполните инвертирование значений элементов вектора *b*. С этой целью в качестве четвертого аргумента в вызове алгоритма используйте предикат *negate< int >*. Выведите значения элементов массива *b* на экран.

5.15. Напишите законченную программу, в которой создайте три целочисленных массива *a, b* и *c* одинакового размера. Инициализируйте массивы *a* и *c* так, чтобы они содержали одинаковые элементы. Выведите значения элементов массивов *a* и *c* на экран. С помощью алгоритма *unique()* удалите в последовательностях элементов массива *a* повторяющиеся элементы и выведите на экран значения элементов этого массива. С помощью алгоритма *unique_copy()* выполните копирование массива *c* в массив *b* с удалением в последовательностях элементов массива *b* повторяющихся элементов и выведите на экран значения элементов массива *b*.

5.16. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v, v1* и *v2*. Инициализируйте первые два вектора случайными значениями, а вектор *v2* так, чтобы он содержал две отсортированных последовательности. Выведите значения элементов этих векторов на экран. С помощью алгоритмов *sort()* и *stable_sort()*выполните соответственно обычную и устойчивую сортировку векторов *v, v1* и выведите на экран значения элементов этих векторов. С помощью алгоритма *binary_search()* выполните поиск в векторе *v* заданного значения и выведите результаты поиска на экран. С помощью алгоритма *implace_merge()* выполните слияние двух отсортированных последовательностей в векторе *v2* и выведите значения элементов этого вектора экран.

5.17. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v, v1* одинакового размера. Инициализируйте оба вектора отсортированными и одинаковыми значениями. Выведи-

те значения элементов этих векторов на экран. С помощью алгоритма *lexicogpaphical_compare()* сравните вектора *v, v1* между собой и выведите на экран результаты сравнения. Измените значение одного из элементов вектора *v1*. С помощью того же алгоритма сравните вектора *v, v1* между собой с использованием предиката *greater<...>* и выведите на экран результаты сравнения. С помощью алгоритмов *lower_bound()* и *upper_bound()* определите соответственно значения элементов вектора *v* после которого и перед которым можно вставить элемент с заданным значением, не нарушая его упорядоченности, и выведите значения элементов вектора *v* на экран.

5.18. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v1, v2* размером 3. Инициализируйте оба вектора так, чтобы они были отсортированными. С помощью подходящего конструктора создайте вещественный вектор *v3*размером 6. Выведите значения элементов векторов *v1, v2* на экран. С помощью алгоритмов *max_element()* и *min_element()* определите и выведите на экран соответственно максимальное и минимальное значения элементов вектора *v1*. С помощью алгоритма *merge()* слейте отсортированные векторы *v1, v2* в вектор *v3* и выведите на экран значения элементов вектора *v3*. С помощью алгоритмов *next_permutation()* и *prev_permutation()* сгенерируйте и выведите на экран перестановки элементов в лексикографическом порядке для векторов *v1, v2* соответственно.

Применяйте только средства стандартной библиотеки языка C++ (потоковый ввод-вывод и объявления стандартных алгоритмов из файла <*algorithm*>).

5.19. Напишите законченную программу, в которой создайте целочисленные массив *a* со значениями элементов 2, 5, 7, 9, массив *b* со значениями элементов 1, 5, 9 и массив *tmp* из семи элементов. Выведите значения элементов массивов *a, b* на экран. С помощью алгоритма *set_intersection()*создайте отсортированное пересечение массивов *a, b,* поместите его в массив *tmp* и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *set_union()* создайте отсортированное объединение массивов *a, b,* поместите его в массив *tmp* и выведите на экран значения элементов массива *tmp*. С помощью алгорит-

ма *set_difference()* скопируйте в *tmp* из массивов *a, b* значения элементов, входящих только в массив *a* и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *sym_difference()* скопируйте в *tmp* из массивов *a, b* значения элементов, входящих только в один из массивов и выведите на экран значения элементов массива *tmp*.С помощью алгоритма *includes()* проверьте включение массива *b* в массив *a* и выведите на экран результаты проверки.

5.20. Напишите законченную программу, в которой с помощью подходящего конструктора создайте целочисленные вектор *Number* со значениями элементов 4, 10, 70, 10, 30, 69, 96, 100 и выведите значения элементов вектора на экран. С помощью алгоритма *make_heap()* преобразуйте *Number* в пирамиду и выведите на экран значения элементов *Number*. С помощью алгоритма *sort_heap()*преобразуйте пирамиду в отсортированную последовательность и выведите на экран значения элементов *Number*. С помощью метода *push_back()* и алгоритма *push_heap()* добавьте элемент в *Number* и выведите на экран значения элементов *Number*. С помощью алгоритма *make_heap()*преобразуйте *Number* в пирамиду и выведите на экран значения элементов *Number*. Извлеките и выведите на экран корневой элемент пирамиды. Затем с помощью алгоритма *pop_heap()*восстановите пирамиду и выведите на экран значения элементов *Number*.

5.21. Напишите законченную программу, в которой создайте три целочисленных массива *a, b* и *c* одинакового размера. Инициализируйте массивы *a* и *c* так, чтобы они содержали одинаковые элементы. Выведите значения элементов массивов *a* и *c* на экран. С помощью алгоритма *unique()*удалите в последовательностях элементов массива *a* повторяющиеся элементы и выведите на экран значения элементов этого массива. С помощью алгоритма *unique_copy()* выполните копирование массива *c* в массив *b* с удалением в последовательностях элементов массива *b* повторяющихся элементов и выведите на экран значения элементов массива *b*.

5.22. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v, v1* и *v2*. Инициализируйте первые два вектора

случайными значениями, а вектор *v2* так, чтобы он содержал две отсортированных последовательности. Выведите значения элементов этих векторов на экран. С помощью алгоритмов *sort()* и *stable_sort()*выполните соответственно обычную и устойчивую сортировку векторов *v, v1* и выведите на экран значения элементов этих векторов. С помощью алгоритма *binary_search()* выполните поиск в векторе *v* заданного значения и выведите результаты поиска на экран. С помощью алгоритма *implace_merge()* выполните слияние двух отсортированных последовательностей в векторе *v2* и выведите значения элементов этого вектора экран.

5.23. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v, v1* одинакового размера. Инициализируйте оба вектора отсортированными и одинаковыми значениями. Выведите значения элементов этих векторов на экран. С помощью алгоритма *lexicogpaphical_compare()* сравните вектора *v, v1* между собой и выведите на экран результаты сравнения. Измените значение одного из элементов вектора *v1*. С помощью того же алгоритма сравните вектора *v, v1* между собой с использованием предиката *greater<...>* и выведите на экран результаты сравнения. С помощью алгоритмов *lower_bound()* и *upper_bound()* определите соответственно значения элементов вектора *v* после которого и перед которым можно вставить элемент с заданным значением, не нарушая его упорядоченности, и выведите значения элементов вектора *v* на экран.

5.24. Напишите законченную программу, в которой с помощью подходящих конструкторов создайте два вектора вещественного типа *v1, v2* размером 3. Инициализируйте оба вектора так, чтобы они были отсортированными. С помощью подходящего конструктора создайте вещественный вектор *v3*размером 6. Выведите значения элементов векторов *v1, v2* на экран. С помощью алгоритмов *max_element()* и *min_element()* определите и выведите на экран соответственно максимальное и минимальное значения элементов вектора *v1*. С помощью алгоритма *merge()* слейте отсортированные векторы *v1, v2* в вектор *v3* и выведите на экран значения элементов вектора *v3*. С помощью алгоритмов *next_permutation()* и *prev_permutation()* сгенерируйте

и выведите на экран перестановки элементов в лексикографическом порядке для векторов *v1, v2* соответственно.

5.25. Напишите законченную программу, в которой создайте целочисленные массив *a* со значениями элементов 2, 5, 7, 9, массив *b* со значениями элементов 1, 5, 9 и массив *tmp* из семи элементов. Выведите значения элементов массивов *a, b* на экран. С помощью алгоритма *set_intersection()* создайте отсортированное пересечение массивов *a, b,* поместите его в массив *tmp* и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *set_union()* создайте отсортированное объединение массивов *a, b,* поместите его в массив *tmp* и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *set_difference()* скопируйте в *tmp* из массивов *a, b* значения элементов, входящих только в массив *a* и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *sym_difference()* скопируйте в *tmp* из массивов *a, b* значения элементов, входящих только в один из массивов и выведите на экран значения элементов массива *tmp*. С помощью алгоритма *includes()* проверьте включение массива *b* в массив *a* и выведите на экран результаты проверки.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 123,1/184,1ч.*

№	Наименование тем	Кол-во часов*	Лит-ра
1	Тема 1. Ассоциативные контейнеры	0/40	[1-5]
2	Тема 2. Итераторы	0/20	[1-5]
3	Тема 3. Алгоритмы	0/20	[1-5]
4	Тема 4. Библиотеки объектов и классов	20/20	[1-5]
5	Тема 5. Удаленные библиотеки	20/20	[1-5]
6	Тема 6. Библиотеки генерации кода	20/0	[1-5]
7	Тема 7. Динамически подключаемые библиотеки	20/20	[1-5]
8	Тема 8. Кроссплатформенная библиотека (CLX)	20/20	[1-5]
9	Тема 9. DLL hijacking	23,1/24,1	[1-5]

*Объем в часах для очной/заочной формы обучения.

Задания для самостоятельного изучения и конспектирования

Тема 1 «Ассоциативные контейнеры». Рассмотреть следующие вопросы:

- создание, копирование и удаление ассоциативных контейнеров;
- управление размером ассоциативных контейнеров;
- операторы сравнения ассоциативных контейнеров;
- прямой доступ к элементам ассоциативных контейнеров.

Тема 2 «Итераторы». Рассмотреть следующие вопросы:

- основные понятия и определения;
- вспомогательные функции для работы с итераторами.

Тема 3 «Алгоритмы». Рассмотреть следующие вопросы:

- классификация библиотечных функций;
- функции разбиения;
- функции сортировки;
- функции двоичного поиска.

Тема 4 «Библиотеки объектов и классов». Рассмотреть следующие вопросы:

- основные понятия и определения объектно-ориентированного программирования;
- библиотеки объектов;
- библиотеки классов;

Тема 5 «Удаленные библиотеки». Рассмотреть следующие вопросы:

- удаленный вызов процедур;
- распределенная архитектура;
- клиент-серверные системы;
- серверы приложений.

Тема 6 «Библиотеки генерации кода». Рассмотреть следующие вопросы:

- кроссплатформенные языки программирования;
- байт-код;

Тема 7 «Динамически подключаемые библиотеки». Рассмотреть следующие вопросы:

- DLL;
- управление памятью;
- импорт библиотек;
- явная загрузка во время выполнения;
- отложенная загрузка.

Тема 8 «Кроссплатформенная библиотека (CLX)». Рассмотреть следующие вопросы:

- основные понятия и определения;
- использование библиотеки CLX.

Тема 9 «DLL hijacking». Рассмотреть следующие вопросы:

- основные понятия и определения;
- история обнаружения уязвимости.

СРС экз. – Самостоятельная работа по подготовке к экзамену в период экзаменационной сессии – 35,65 ч/8,65 ч.*

Экзаменационные вопросы по курсу

1. Библиотеки предназначены для:

а) эффективного использования памяти компьютера
б) облегчения распространения программного кода
в) повышения быстродействия программ
г) повторного использования программного кода

2. Установите соответствие между парами понятий:

1) код из библиотеки связывается с программой после ее запуска на выполнение	A) статически компонуемая библиотека (SLL)
2) во время сборки (компоновки) в программу вставляются куски объектного кода из библиотеки	B) библиотека динамической компоновки (DLL)
3) код из библиотеки связывается с программой как часть процесса ее запуска на выполнение	

3. К созданию STL имеют отношение:

а) Гради Буч б) Александр Степанов в) Ивар Якобсон
г) Борис Петров д) Дэвид Мюссер е) Джон Бэкус
ж) Менг Ли

4. Наиболее важная концепция, положенная в основу STL:

а) объектно-ориентированное программирование
б) обобщенное программирование

5. STL является частью стандартной библиотеки C++ и содержит определение следующих видов шаблонов:

а) итераторы б) ввод-вывод в) алгоритмы
г) регулярные выражения д) контейнеры

6. Контейнеры делятся на следующие категории:

а) последовательные б) параллельные в) ассоциативные
г) прямого доступа д) неупорядоченные ассоциативные
е) контейнерные адаптеры

7. Установите соответствие между парами понятий:

1)	array	A)	последовательный контейнер
2)	deque	B)	ассоциативный контейнер
3)	map		
4)	vector		
5)	set		
6)	list		

8. Какие шаблонные классы являются последовательными контейнерами?

а) array б) complex в) deque г) forward_list д) list е) map ж) queue з) set и) stack к) unordered_map л) unordered_set м) vector

9. Установите соответствие между парами понятий:

1)	list	A)	поддерживает итераторы произвольного доступа
2)	deque	B)	поддерживает двунаправленные итераторы
3)	vector	C)	вставка и удаление в произвольном месте занимают линейное время

D) вставка и удаление в произвольном месте требуют постоянного времени
E) вставка и удаление в конце последовательности требуют постоянного времени
F) вставка и удаление в начале и в конце последовательности требуют постоянного времени

10. Установите соответствие между фрагментами кода и способами создания последовательных контейнеров:

1) deque<int> d1{2,3,4,5};	A) Создание пустого контейнера с помощью конструктора по умолчанию
2) list<int> a{2,3,4,5}; list<int> v(a);	B) Создание контейнера заданного размера, заполненного значениями по умолчанию
3) vector<int> v;	C) Создание контейнера заданного размера, заполненного постоянными значениями
4) list<int> s(5);	D) Создание контейнера, заполненного диапазоном значений
5) deque<int> d1{2,3,4,5}; deque<int> d2(move(d1));	E) Создание контейнера путем копирования контейнера того же типа
6) int z[]{3, 5, 7, 9}; list x(z, z+3);	F) Создание контейнера путем перемещения контейнера того же типа
7) deque<int> d(7,3);	G) Создание контейнера с помощью списка инициализации

11. Установите соответствие между фрагментами кода и способами присваивания последовательных контейнеров:

1) deque<int> d1{2,3,4,5}, d2; d2 = d1;	A) Контейнеру присваивается список
2) list<int> a{2,3,4,5}, b; b = move(a);	B) Контейнеру присваивается последовательноть одинаковых значений заданной длины
3) vector<int> v;	C) Контейнеру

v = {2,3,4,5};

4) deque<int> d1{2,3,4,5}, d2;
d2.assign(d1.begin(),d1.begin()+3);

5) vector<int> v;
v.assign(5,100);

6) list g;
g.assign({2,3,4,5});

присваивается другой контейнер

D) Контейнер получает значения из другого контейнера путем перемещения

E) Контейнер получает диапазон значений из другого контейнера

12. Какие шаблонные классы являются ассоциативными контейнерами?

а) array б) complex в) deque г) forward_list д) list е) map ж) queue

з) set и) stack к) vector л) multiset м) multimap

13. Какие объекты из нижеперечисленных имеет смысл использовать в качестве сравнивающих объектов для ассоциативных контейнеров?

а) [](int k1, int k2){return k2<k1;}

б) [](int k){return k%2 == 1;}

в) [](int k1, int k2){return k2+k1;}

г) struct S {bool operator()(int k1, int k2){return k2<k1;}} f;

д) struct S {bool operator()(int k1){return k1<10;}} f;

е) bool f1(int k1, int k2){return k2<k1;}

ж) int f2(int k1, int k2){return k2-k1;}

14. Установите соответствие между фрагментами кода и способами создания ассоциативных контейнеров:

1) struct S {bool operator()(int k1, int k2){return k2<k1;}};
set<int,S> d {1,2,3,4,5};

2) auto ff = [](int k1, int k2){return k2<k1;};
int z[]{3, 5, 7, 9};

A) Создание пустого контейнера с помощью конструктора по умолчанию

B) Создание пустого контейнера с помощью конструктора со сравнивающим объектом

set<int, bool (*) (int,int)> d(z, z+4, ff);

3) set<int> v;

4) struct S {bool operator()(int k1, int k2){return k2<k1;}} f;
set<int,S> q1{1,2,3,4,5};
set<int,S> q2(move(q1));

5) set<int,bool (*) (int,int)> h([](int k1, int k2){return k2<k1;});
h = {1,2,3,4,5};
set<int, bool (*) (int,int)> w(h);

6) set<int,bool (*) (int,int)> h([](int k1, int k2){return k2<k1;});

7) struct S {bool operator()(int k1, int k2){return k2<k1;}};
set<int,S> d;

C) Создание контейнера, заполненного диапазоном значений

D) Создание контейнера путем копирования контейнера того же типа

E) Создание контейнера путем перемещения контейнера того же типа

F) Создание контейнера с помощью списка инициализации

15. Аргументами шаблонных функций, реализующих стандартные алгоритмы STL, могут быть:

а) объекты любых контейнерных классов б) итераторы в) функции г) функторы д) лямбда-выражения

16. Интервалы какого вида могут задаваться в качестве аргументов шаблонных функций STL?

а) открытые б) полуоткрытые в) закрытые

17. Является ли стандартная библиотека шаблонов (STL) подмножеством стандартной библиотеки C++?

нет

да

18. Из каких основных компонентов состоит ядро STL?

1) строки 2) числа 3) потоки 4) регулярные выражения 5) контейнеры 6) алгоритмы 7)итераторы

19. Какие последовательные контейнерами поддерживают произвольный доступ?

а) однонаправленный список
б) двунаправленный список
в) вектор
г) массив
д) дек

20. Какие последовательные контейнерами поддерживают обратные итераторы?

а) последовательный список
б) дек
в) массив
г) вектор
д) список

4. КУРСОВАЯ РАБОТА

Учебным планом предусмотрено выполнение курсовой работы – 34,5 ч.

Цель работы: закрепление и углубление знаний, полученных студентами при изучении указанной дисциплины, развитие практических навыков при выборе представления исходных данных, использовании стандартной библиотеки шаблонов языка программирования С++ при написании программ, тестировании и отладки программы, оформлении сопроводительной документации.

Пояснения к работе: курсовая работа выполняется индивидуально каждым студентом в соответствии с выданным вариантом. Обязательным является использование в курсовой работе стандартной библиотеки шаблонов языка программирования С++ и пользовательских классов. Курсовая работа выполняется в интегрированной среде разработки Dev-C++.

В процессе работы студент должен:

1. Выполнить анализ предметной области.
2. Разработать пользовательские классы.
3. Разработать пользовательский интерфейс для ввода и получения информации.
4. Предусмотреть обработку исключительных ситуаций, возникающих во время работы программы.
5. Провести отладку и тестирование программы.
6. Оформить для нее документацию.

При защите курсовой работы необходимо предоставить:

1. Пояснительную записку, включающую:
   - титульный лист;
   - лист задания на работу;
   - оглавление;
   - введение;
   - теоретическую часть;
   - практическую часть с описанием хода работы с необходимыми пояснениями;
   - заключение;
   - список литературы;
   - приложение с листингом программы.
2. Исходные тексты программы в электронном виде.
3. Тексты программ должны быть оформлены в соответствии с принятыми стандартами C++, и содержать комментарии разработчика для всех методов и классов.

Если перечисленные выше требования не выполнены – работа не принимается.

Задания к работе:

1. Разработать автоматизированную информационную систему железнодорожного вокзала. Информационная система должна содержать сведения об отправлении поездов дальнего следования. Система должна обеспечивать возможность выбора с помощью меню и выполнения следующих функций:
   - первоначальный ввод данных в информационную систему;
   - корректировка введенных данных;
   - удаление введенных данных;

- вывод сведений по всем поездам;
- вывод сведений по поезду с запрошенным номером;
- вывод сведений по тем поездам, которые следуют до запрошенной станции назначения;
- формирование информационного табло со списком поездов и временем отправления.

2. Разработать автоматизированную информационную систему составления расписания в учебном заведении. Данная система должна составлять расписание работы для конкретного преподавателя на неделю, с учетом того, что один и тот же преподаватель может читать несколько разных предметов в разных группах. Также система должна осуществлять поиск по следующим критериям:
   - ФИО преподавателя;
   - номер аудитории;
   - название предмета;
   - номер группы.
3. Разработать программу – ежедневник, который должна содержать список ежедневных дел и отслеживать своевременность их выполнения. Ежедневник должен поддерживать следующие операции:
   - создание пустого ежедневника;
   - добавление нового элемента в список дел;
   - редактирование элемента ежедневника;
   - вывод списка ежедневника, упорядоченного по сроку выполнения;
   - вывод списка просроченных дел;
   - получение информации об элементе, который должен выполняться в данный момент;
   - исключение выполненного элемента из списка дел;
   - очистка списка дел.
4. Разработать программу – журнал салона видео проката. Должны быть реализованы следующие функции:
   - добавление фильма в каталог фильмов проката;
   - добавление новых абонентов;

- занесение фильма в каталог фильмов, которые абонент получил в салоне.
- поиск по фамилии фильмов, которые взял абонент, сколько заплатил, на какой срок взял фильм, какие во время не вернул.
- поиск по наименованию информации о текущем статусе фильма (в салоне/у абонента и т.д.).

5. Разработать программу – телефонный справочник. Каждая запись справочника должна содержать следующую информацию:
   - номер телефона;
   - фамилия, имя, отчество;
   - адрес;
   - примечания.

   Разрабатываемая программа должна поддерживать следующие функции:
   - просмотр информации справочника;
   - редактирование уже существующей информации;
   - добавление новой информации;
   - сортировка необходимой информации по различным полям.
   - поиск информации по различным полям.
6. Разработать программу для проведения электронного экзамена. Преподаватель готовит список тестовых заданий (в закрытой и открытой формах) к экзамену, указывая для каждого из них правильный ответ. Студенты сдают экзамен, отвечая на N вопросов, случайным образом выбранных системой и получают оценку. Студент после регистрации должен выбрать предмет и запустить тест. Если экзамен уже сдан по предмету, то его в этом списке не будет.
7. Разработать программу для формирования программы концерта по заявкам. Разрабатываемая программа должна поддерживать следующие функции:
   - добавление новых песен;
   - редактирование существующей информации о песнях;
   - удаление песен;
   - просмотр списка песен;

– поиск песен по различным полям.

Пользователи регистрируются в системе и выбирают песни из предложенного списка. Каждый пользователь может выбрать любое количество песен. Когда время подачи заявок оканчивается, формируется программа концерта, включающая песни, набравшие наибольшее число заявок (количество песен определяется заранее).

8. Разработать программу автоматизирующую учет компьютеров и программного обеспечения в организации. Должны быть реализованы следующие функции:
   – добавление информации в журнал;
   – редактирование существующей информации;
   – удаление информации из журнала;
   – сортировка по технике, по заводскому номеру, по инвентарному номеру;
   – поиск информации по различным полям.
9. Разработать программу, хранящую данные о студентах и их успеваемости по каждой дисциплине. Должны быть реализованы следующие функции:
   – добавление, редактирование, удаление информации о студентах;
   – добавление, редактирование, удаление информации о дисциплинах;
   – поиск информации по фамилии студента, с выводом всех его оценок по дисциплине;
   – поиск информации по дисциплине с выводом количества студентов, изучающих данный предмет;
   – поиск информации по оценкам.
10. Разработать автоматизированную информационную систему аэропорта. Информационная система должна содержать информацию о полетах, их расписании и самолетах. Система должна обеспечивать возможность выбора с помощью меню и выполнения следующих функций:
   – первоначальный ввод данных в информационную систему;
   – корректировка введенных данных;
   – удаление введенных данных;

- вывод сведений по всем самолетам и их типам;
- вывод сведений по самолету с запрошенным номером рейса;
- вывод сведений по тем самолетам, которые следуют до запрошенного аэропорта.
- формирование информационного табло со списком рейсов и временем отправления.

11. Разработать программу, хранящую данные о книжном складе. Программа должна хранить и обрабатывать информацию о книгах на книжных складах, а также учитывать продажи книг со складов. Должны быть реализованы следующие функции:
    - добавление, редактирование, удаление информации о книгах;
    - поиск книг по различным полям (наименованию, автору, цене и т.д.);
    - сортировка книг по различным полям (наименованию, автору, цене и т.д.);
    - расчет отпускных цен на книги в зависимости от указанного глобального параметра;
    - учет количества проданных книг;
    - подсчет выручки (по всем книгам, по автору, по жанру и т.д.).
12. Разработать программу, хранящую штатное расписание предприятия. Должны быть реализованы следующие функции:
    - добавление, редактирование, удаление работников;
    - поиск работников по различным полям (ФИО, подразделение, отдел, должность, разряд и т.д.);
    - сортировка работников по различным полям;
    - формирование следующих отчетов: вакантные места, расписание по категориям, расписание по подразделениям, пенсионный список.
13. Разработать программу учета услуг Водоканала. Должны быть реализованы следующие функции:
    - добавление, редактирование, удаление данных о потребителях;

- добавление, редактирование, удаление данных об услугах;
- поиск потребителей по различным полям (ФИО, адрес и т.д.);
- поиск услуг по различным полям;
- формирование отчета по оказанным услугам потребителям за указанный промежуток времени.

14. Разработать программу, позволяющую выполнять перевод слов с английского языка на русский и наоборот. Должны быть реализованы следующие функции:
    - добавление, редактирование, удаление слов в словаре;
    - вывод слов русского и английского словарей с возможностью выбора типа сортировки;
    - перевод введенного слова с выводом возможных вариантов перевода. Слово, которое не найдено в словаре, выводиться в неизменном виде.
15. Разработать программу, автоматизирующую работу библиотеки. Должны быть реализованы следующие функции:
    - регистрация новой книги;
    - редактирование информации о книге;
    - удаление книги из общего списка;
    - регистрация читателя;
    - редактирование личных данных читателя;
    - удаление читателя из списка;
    - операции поиска и отображения книги по фамилии автора, с указанием общего количества экземпляров и количества выданных экземпляров;
    - выдача общего списка выданных книг;
    - выдача списка выданных книг по фамилии читателя.
16. Разработать программу хранящую данные об автомобилях и их владельцах. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных об автомобилях;
    - добавление, удаление, редактирование данных о владельцах;

- поиск автомобилей по различным полям (VIN, государственный номер, марка) с выводом списка владельцев с сортировкой по различным полям;
- вывод списка автомобилей по ФИО владельца с сортировкой по различным полям;

17. Разработать программу хранящую данные о нарушениях правил дорожного движения. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о штрафах;
    - добавление, удаление, редактирование данных о нарушителях;
    - вывод таблицы штрафов с возможностью выбора типа сортировки;
    - поиск штрафов по гос. номеру автомобиля;
    - поиск штрафов по номеру водительского удостоверения с указанием общей суммы задолженности и статуса штрафа (оплачен\не оплачен);
18. Разработать программу учета входящей и исходящей корреспонденции. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о входящих\исходящих документах;
    - добавление, удаление, редактирование данных об исполнителях;
    - вывод данных о входящих\исходящих документах с возможностью выбора типа сортировки;
    - поиск и отображение уже исполненной корреспонденции, только текущей и корреспонденции, у которой просрочен срок исполнения;
    - поиск корреспонденции по данным исполнителя;
19. Разработать автоматизированную информационную систему автовокзала. Информационная система должна обеспечивать хранение информации о номере автобуса, пункте отправления и назначения, времени отправления и стоимости билета и т.д, а также обеспечивать возможность выбора с помощью меню и выполнения следующих функций:

- первоначальный ввод данных, корректировка введенных данных, удаление введенных данных;
- вывод сведений по всем рейсам, по рейсам с запрошенным номером, по тем рейсам, которые следуют до запрошенной станции назначения;
- формирование информационного табло со списком рейсов, временем отправления с возможностью выбора типа сортировки;
- ведение информации об общем количестве мест, о проданных на конкретный рейс билетах, выводе общей суммы проданных билетов;

20. Разработать программу хранящую данные транспортного агентства. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных об автомобилях агентства;
    - добавление, удаление, редактирование данных о клиентах агентства;
    - вывод таблицы автомобилей с возможностью выбора типа сортировки, с указанием подробной истории аренды;
    - поиск автомобилей по гос. номеру автомобиля, марке, количеству посадочных мест, с указанием статуса автомобиля (свободен\в аренде), срока, на который автомобиль арендован, фамилии арендатора;
    - поиск по атрибутам клиента информации об арендованных автомобилях;
21. Разработать программу хранящую данные жильцов домов. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о домах;
    - добавление, удаление, редактирование данных о жильцах;
    - вывод таблицы домов с возможностью выбора типа сортировки, с указанием общего количества квартир и жильцов;
    - поиск всех квартир по данным жильца;

- поиск квартиры с максимальным числом жильцов в доме, по всем домам;
- поиск самого юного и самого пожилого жильца в доме, по всем домам;
- поиск студентов, пенсионеров в доме, по всем домам;

22. Разработать программу хранящую данные о животных. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о животных (название, вид, численность и т.д.);
    - добавление, удаление, редактирование данных странах обитания животных;
    - вывод таблицы животных с возможностью выбора типа сортировки;
    - вывод таблицы животных с возможностью выбора типа сортировки, численность которых меньше N;
    - вывод стран обитания с возможностью выбора типа сортировки, с выводом списка животных;
    - поиск животных по названию, стране обитания, виду;
23. Разработать программу налогового учета. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о налоговом органе;
    - ввод данных о сданных налоговых отчетов по выбранному налоговому органу;
    - поиск налогового органа - по наименованию, региону, коду;
    - вывод в порядке расположения всех отчетных данных по каждому налоговому органу (за каждый месяц);
    - формирование итоговой таблицы по количеству сданных отчетных данных налоговых органов. Данные выводятся в порядке возрастания. Необходимо задать период формирования отчетных данных
24. Разработать программу автоматизирующую работу фитнес-клуба. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных о клиентах клуба;

- ведение справочника клубных карт, с указанием скидки на услуги по карте и накопленных бонусов;
- регистрация посещений клиентов, формирование журнала посещений за любой интервал времени по любому залу и клиенту;
- вывод всех клиентов, по типу клубной карты с возможностью выбора типа сортировки;
- формирование финансовой статистики.

25. Разработать программу учета продаж и гарантийного обслуживания автомобилей. Должны быть реализованы следующие функции:
    - добавление, удаление, редактирование данных об автомобилях;
    - добавление, удаление, редактирование данных о владельцах;
    - учет продаж;
    - поиск проданных автомобилей за определенный период с возможностью выбора типа сортировки;
    - поиск проданных автомобилей по марке с возможностью выбора типа сортировки;
    - ведение истории обращений по автомобилю;
    - поиск автомобилей по типу обращения с возможностью выбора типа сортировки;

5. КОНТАКТНАЯ ВНЕАУДИТОРНАЯ РАБОТА

СРС – групповые консультации с преподавателем в течение семестра – 0,9 ч.

Групповая консультация перед экзаменом – 2 ч.

Проведение консультаций и защита курсовой работы – 1,5 ч.

СРС экз. – сдача экзамена – 0,35 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1.Липман С. Язык программирования C++. Полное руководство [Электронный ресурс] : / Липман С., Лажойе Ж. - Электрон. дан. - М. : ДМК Пресс, 2006. - 1104 с. - Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=1216 - Загл. с экрана.

2.Страуструп Б. Дизайн и эволюция C++ [Электронный ресурс] : . - Электрон. дан. - М. : ДМК Пресс, 2007. - 445 с. - Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=1222 - Загл. с экрана.

Дополнительная учебная литература

3.Кетков Ю. Л. Введение в языки программирования C и C++ [Электронный ресурс]: Ю. Л. Кетков – М.:Национальный Открытый Университет «ИНТУИТ», 2008. - 252 с. - Режим доступа: http://www.knigafund.ru/books/176103

4.Страуструп Б. Язык программирования C++ для профессионалов [Электронный ресурс]: Б. Страуструп – М.:Национальный Открытый Университет «ИНТУИТ», 2006. - 568 с. - Режим доступа: http://www.knigafund.ru/books/177881.

5.Керниган Б. В., Ричи Д. М. Язык программирования C [Электронный ресурс]: Б. В. Керниган, Д. М. Ричи – М.:Национальный Открытый Университет «ИНТУИТ», 2006. - 272 с. - Режим доступа: http://www.knigafund.ru/books/176845.

Учебно-методическое издание

Шайда Алексей Юрьевич

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК

Методические указания
к практическим занятиям, лабораторным
работам, курсовой и самостоятельной
работе студентов бакалавриата
направления подготовки
«Математическое обеспечение и администрирование информационных систем»,
«Информатика и вычислительная техника»
и «Программная инженерия»
(очная и заочная формы обучения)

Редактор *Н.А.Юшко*
Подписано в печать 27.12.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 2,79. Уч.-изд.л. 3,0 . Тираж 50 экз. Заказ.

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ПРОГРАММИРОВАНИЕ

Методические указания
к практическим занятиям, лабораторным работам
и самостоятельной работе студентов
бакалавриата направления подготовки
«Математическое обеспечение и администрирование информационных систем»,
«Информатика и вычислительная техника»
и **«Программная инженерия»**

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 51:512(076.5)
ББК 22.1 я73

Рецензент

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Панфилов Александр Николаевич

Масленников А.А.

Программирование: методические указания к практическим занятиям, лабораторным работам и самостоятельной работе студентов бакалавриата направления подготовки «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника» и «Программная инженерия» / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 24с.

Приведены методические рекомендации к выполнению практических заданий и лабораторных работ. Рассмотрены основные возможности и особенности платформы Microsoft .NET Framework, различные типы приложений на основе технологий Windows Forms, WPF, WWF, построение приложений на основе сервис-ориентированной архитектуры. Представлено решение задач с применением объектно-ориентированного языка программирования C#. Предложены темы для самостоятельного изучения платформы Microsoft .NET Framework, включающие вопросы функционирования платформы .NET, основных ее компонент, вопросы работы с СУБД Microsoft SQL Server, веб-сервисами, облачными технологиями.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 51:512(076.5)

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Изучение компилятора языка C# платформы .Net Framework	2	Тест	1-5.03	[6]
2	Разбор простого примера на C# платформы .Net Framework	2	Тест	16-18.03	[6]
3	Примеры использования операторов цикла и условных операторов .Net Framework	2	Тест	30.03-01.04	[6]
4	Работа с методами классов .Net Framework	2	Тест	05.04-07.04	[6]
5	Работа с массива .Net Framework	2	Тест	13-15.04	[6]
6	Примеры запросов LINQ to SQL	2	Тест	27-29.04	[6]
7	Пример веб-сайта на технологии ASP.NET	2	Тест	03-05.05	[6]
8	Приложения на технологии WPF	2	Тест	10-12.05	[6]
9	Пример приложения MVVM	2	Тест	24-26.05	[6]

Практическое занятие № 1

ИЗУЧЕНИЕ КОМПИЛЯТОРА ЯЗЫКА C# ПЛАТФОРМЫ .NET FRAMEWORK

1. Создать консольное приложение:

```
using System;
namespace HelloWorldApplication {
 class HelloWorld {
 static void Main(string[] args) {
 /*первая программа C# */
Console.WriteLine("Hello World");
Console.ReadKey(); }
}
}
```

2. Запустить программу .NET Reflector, указать адрес скомпилированного exe файла, открыть и рассмотреть полученные MSIL инструкции.

Практическое занятие № 2

РАЗБОР ПРОСТОГО ПРИМЕРА НА C# ПЛАТФОРМЫ .NET FRAMEWORK

1. Скомпилировать приложение, выделить операторы ввода, операторы вывода:

```
using System;
namespace RectangleApplication
{   class Rectangle
  {
    // member variables
    double length; double width;
    public void Acceptdetails()
    {
      length = 4.5; width = 3.5;
    }
    public double GetArea()
    {
      return length * width;
```

```
    }
public void Display()
    {
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);
      Console.WriteLine("Area: {0}", GetArea());
    }
  }
  class ExecuteRectangle
  {
    static void Main(string[] args)
    {
      Rectangle r = new Rectangle();
      r.Acceptdetails();
      r.Display();
      Console.ReadLine();
    }
  } }
```

2. Изменить программу таким образом, чтобы перед запуском программа запрашивала наименование учетной записи пользователя.

Практическое занятие № 3

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ОПЕРАТОРОВ ЦИКЛА И УСЛОВНЫХ ОПЕРАТОРОВ .NET FRAMEWORK

1. Скомпилировать приложение, использующее бесконечный оператор цикла, добавить выход из оператора при количество обращений равным 1000:

```
//пример оператора
for (;;)
{
 //}
```

2. Реализовать оператор цикла без условия выхода из цикла, как в примере:

```
for (int laborNum = 1; laborNum <= 12; )
```

```
{ bool success = HerculesLabors(laborNum);
    if (success == true)  laborNum++;
 }
```

3. Реализовать оператор цикла с несколькими параметрами, например:

```
for (int i = 0, j = 9; i < 10; i++, j--)
{ nums[i, j] = 1; }
```

Практическое занятие № 4

РАБОТА С МЕТОДАМИ КЛАССОВ .NET FRAMEWORK

1. Реализовать простой класс, содержащий один метод, обеспечить ввод и вывод данных на консоль. Пример класса:

```
class NumberManipulator
{
  public int FindMax(int num1, int num2)
  {  int result;
    if (num1 > num2)
      result = num1;
    else
      result = num2;
    return result;
  }
}
NumberManipulator NM=new NumberManipulator();
var Res= NM.FindMax(10,20);
```

2. Выполнить тестирование перекрытия переменной в классе:

```
class NumberManipulator
{
  public int num1;
  public int ChangeNum1(int num1)
 { um1= num1+100;// меняется только локальная переменная
/*свойство класса NumberManipulator остается без изменений*/
    return result;   } }
```

```
NumberManipulator NM=new NumberManipulator();
var Res= NM. ChangeNum1(10);
```

3. Выполнить обращение к текущему объекту внутри метода в соответствии с примером:

```
class NumberManipulator
{ public int num1;
  public int ChangeNum1(int num1)
  {  this.num1= num1+100;// меняется
     // свойство класса NumberManipulator
      return result;
  }
}
NumberManipulator NM=new NumberManipulator();
var Res= NM. ChangeNum1(10);
```

Практическое занятие № 5

РАБОТА С МАССИВА .NET FRAMEWORK

1. Реализовать одномерный массив на основе System.Array, обеспечить заполнение данных из консоли и вывод на консоль.
2. Реализовать двумерный массив на основе System.Array, обеспечить заполнение данных из консоли и вывод на консоль.
3. Реализовать заполнение списка строк на основе List<T>, обеспечить заполнение данных из консоли и вывод на консоль.

Практическое занятие № 6

ПРИМЕРЫ ЗАПРОСОВ LINQ TO SQL

1. Используя Management Studio создать базу данных, реализовать одну таблицу, описывающую сотрудников какой-либо организации.
2. Выполнить поиск сотрудников по фамилии на основе лямбда выражений LinqToSql, оператора Where
3. Выполнить выборку только фамилий сотрудников на основе лямбда выражений LinqToSql, оператора Select

Практическое занятие № 7

ПРИМЕР ВЕБ-САЙТА НА ТЕХНОЛОГИИ ASP.NET

1. В Microsoft Visual Studio создать проект веб-сайта ASP.NET, создать одну мастер страницу, унаследовать дочернюю страницу, на странице должно быть размещено сообщение «Hello world».

2. Добавить веб-сервис к созданному приложению ASP.NET, веб-сервис должен содержать метод, возвращающий текущую дату, запустить проект веб-сервиса, убедиться в существовании страницы описания веб-сервиса

Практическое занятие № 8

ПРИЛОЖЕНИЯ НА ТЕХНОЛОГИИ WPF

1. Создать приложение WPF, содержащее три кнопки, по нажатии на которые кнопки должны исчезать с анимацией изменения формы.

2. Создать приложение WPF, позволяющее отображать анимацию положения солнца в текущий момент времени.

Практическое занятие № 9

ПРИМЕР ПРИЛОЖЕНИЯ MVVM

1. Разработать приложение Windows Forms, содержащее заполнение данных о сотруднике (ФИО, дата рождения).
2. Разработать такое же приложение на основе технологии MVVM и технологии WPF, сравнить и описать преимущество разных реализаций задачи.

2. ЛАБОРАТОРНЫЕ РАБОТЫ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

№	Наименование тем	Кол-во часов	Сроки контроля	Литература
1	Создание библиотеки классов и консольного приложения	4	8-10.03	[1-6]
2	Создание оконного приложения Windows Forms	4	22-24.03	[1-6]
3	Создание веб-сервиса, с использованием технологии веб-приложений ASP.NET и web-services (*.asmx)	4	02.04-04.04	[1-6]
4	Создание веб-приложения ASP.NET Forms	4	08.04-10.04	[1-6]
5	Создание веб-сервиса на технологии Windows communication Foundation (WCF)	4	16-20.04	[1-6]
6	Создание объектов классов, реализованных в лабораторной работе 1 с помощью механизмов Windows Workflow Foundation (WWF)	4	28-29.04	[1-6]
7	Реализация функционала лабораторной работы № 2 с использованием технологии XAML(Extensible Application Markup Language) и WPF (Windows Presentation Foundation)	4	01-05.05	[1-6]
8	Реализация работы с очередью обмена сообщениями на основе технологии Service Bus Queue Azure Cloud	4	07-09.05	[1-6]

Лабораторная работа №1

СОЗДАНИЕ БИБЛИОТЕКИ КЛАССОВ И КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Задание: Реализовать библиотеку классов. Создать классы, в соответствии с вариантом задания. Классы должен содержать как минимум три свойства и три метода. Между классами должны быть обеспечены связи (агрегация, наследование и т.д.), должен быть выбран базовый класс и зависимые (дочерние). Реализовать консольное приложение для создания и заполнения полей объектов библиотеки классов.

Варианты заданий к лабораторной работе №1

1. Студент, преподаватель, персона, заведующий кафедрой.
2. Рабочий, кадры, инженер, администрация.
3. Деталь, механизм, изделие, узел.
4. Организация, страховая компания, нефтегазовая компания, завод.
5. Журнал, книга, печатное издание, учебник.
6. Тест, экзамен, выпускной экзамен, испытание.
7. Место, область, город, мегаполис.
8. Игрушка, продукт, товар, молочный продукт.
9. Квитанция, накладная, документ, счет.
10. Автомобиль, поезд, транспортное средство, экспресс.
11. Двигатель, двигатель внутреннего сгорания, дизель, реактивный двигатель.
12. Республика, монархия, королевство, государство.
13. Млекопитающее, парнокопытное, птица, животное.
14. Корабль, пароход, парусник, корвет.
15. Частный дом, многоквартирный дом, строение, офис.

Лабораторная работа №2

СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ WINDOWS FORMS

Задание: Реализовать оконное приложение (Windows Forms).

Оконное приложение должно позволять создавать объекты, отображать список созданных объектов в табличной форме, выполнять поиск. Реализовать сериализацию и десериализацию классов в приложении. Интерфейс приложения должен быть дополнен кнопками для сохранения и загрузки классов из XML файлов.

Операции сохранения и загрузки XML файлов должны быть выполнены с использованием методов async и await.

Варианты заданий к лабораторной работе №2

В соответствии с вариантом лабораторной работы № 1.

Лабораторная работа №3

СОЗДАНИЕ ВЕБ-СЕРВИСА, С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ ВЕБ-ПРИЛОЖЕНИЙ ASP.NET И WEB-SERVICES (*.ASMX)

Задание: Создать веб-сервис, используя технологию веб-приложений ASP.NET и web-services (*.asmx). Реализовать метод для запроса объектов с сервера (которые должны храниться в базе данных, для каждого класса, который задан в варианте). Реализовать методы для создания объектов каждого из классов заданного варианта на сервере (в метод передаются параметры, создание объекта происходит на сервере, объект записывается в базу данных). Клиентское приложение должно обращаться к веб-сервису, разработанному студентом с любым другим вариантом лабораторной работы.

Ссылка на веб-сервис должна быть добавлена в приложение,

созданное в предыдущей лабораторной работе. Из оконного приложения запрос должен отправляться к веб-сервису для получения списка объектов и создания новых.

Запросы к серверу и их обработка должны использовать асинхронные варианты методов веб-сервиса (они генерируются автоматически для каждого соответствующего метода).

Варианты заданий к лабораторной работе №3

В соответствии с вариантом лабораторной работы № 1.

Лабораторная работа №4
СОЗДАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ASP.NET FORMS

Задание: К созданному веб-приложению необходимо добавить веб-страницу *.aspx (страница ASP.NET). Затем необходимо в серверной части кода реализовать асинхронный метод и вызывать его с использованием технологий AJAX и PageMethods.

Реализованный на странице асинхронный метод должен возвращать объект одного из классов, после чего данный объект должен быть отображен на странице.

Варианты заданий к лабораторной работе №4

В соответствии с вариантом лабораторной работы № 1.

Лабораторная работа №5
СОЗДАНИЕ ВЕБ-СЕРВИСА НА ТЕХНОЛОГИИ WINDOWS COMMUNICATION FOUNDATION (WCF)

Задание: Реализовать веб-сервис и клиентское приложение (по аналогии с лабораторной работой № 3). Веб-сервис должен быть реализован с использованием технологии Windows communication Foundation (WCF). Обращение должно производиться к веб-сервису, разработанному студентом с любым другим вариантом задания.

Варианты заданий к лабораторной работе №5

В соответствии с вариантом лабораторной работы № 3.

Лабораторная работа №6
СОЗДАНИЕ ОБЪЕКТОВ КЛАССОВ, РЕАЛИЗОВАННЫХ В ЛАБОРАТОРНОЙ РАБОТЕ 1 С ПОМОЩЬЮ МЕХАНИЗМОВ WINDOWS WORKFLOW FOUNDATION (WWF)

Задание: Реализовать создание объектов классов, реализованных в лабораторной работе 1 с помощью механизмов Windows Workflow Foundation (WWF). Так же в завершении процессов объекты должны сохраняться в файл (сериализация).

Варианты заданий к лабораторной работе №6

В соответствии с вариантом лабораторной работы № 1.

Лабораторная работа №7
РЕАЛИЗАЦИЯ ФУНКЦИОНАЛА ЛАБОРАТОРНОЙ РАБОТЫ № 2 С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ XAML(EXTENSIBLE APPLICATION MARKUP LANGUAGE) И WPF (WINDOWS PRESENTATION FOUNDATION)

Задание: Реализовать функционал лабораторной работы № 2 с использованием технологии XAML(Extensible Application Markup Language) и WPF (Windows Presentation Foundation).

Варианты заданий к лабораторной работе №7

В соответствии с вариантом лабораторной работы № 2.

Лабораторная работа №8
РЕАЛИЗАЦИЯ РАБОТЫ С ОЧЕРЕДЬЮ ОБМЕНА СООБЩЕНИЯМИ НА ОСНОВЕ ТЕХНОЛОГИИ SERVICE BUS QUEUE AZURE CLOUD

Задание: Реализовать очередь обмена сообщениями на основе технологии Service Bus Queue Azure Cloud. Реализовать клиентские приложения для демонстрации обмена сообщениями. В сообщениях должна передаваться информация о составе объектов (одного из классов) между клиентскими приложениями. После передачи объекта от одного клиента другому – объект должен быть сериализован.

Варианты заданий к лабораторной работе №8

В соответствии с вариантом лабораторной работы № 1.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование –87,1ч.

№	Наименование тем	Кол-во часов	Литература
1	Тема 1. Обзор нововведений платформы Microsoft .NET Framework. Подготовка примеров с использованием новых языковых конструкций	10	[1-6]
2	Тема 2. Обзор языка MSIL, изучение конструкций данного языка для простого примера программы на языке C#	10	[1-6]
3	Тема 3. Закрепление навыков работы с операторами цикла и условными операторами C#	10	[1-6]
4	Тема 4. Конструкторы класса. Перегрузка методов класса. Виртуальные методы. Наследование. Закрытые от наследования классы.	10	[1-6]
5	Тема 5. Рефлексия. Доступ к свойствам объекта. Зубчатые массивы.	10	[1-6]
6	Тема 6. Изучение основ Microsoft SQL Server. Знакомство с дизайнером LINQ to SQL. Знакомство с технологией LINQ To Objects	10	[1-6]
7	Тема 7. Изучение правил формирования CSS. Обзор Javascript, AJAX и JQuery.	10	[1-6]
8	Тема 8. Разработка примеров для сравнения технологий разработки программного обеспечения (MVVM, MVP и MVC)	9	[1-6]
9	Тема 9. Регистрация в Windows Azure Cloud, настройка учетной записи. Настройка ресурсов.	8,1	[1-6]

Задания для самостоятельного изучения и конспектирования

Тема 1 «Обзор нововведений платформы Microsoft .NET Framework. Подготовка примеров с использованием новых языковых конструкций». Рассмотреть следующие вопросы:

- Компилируемые и интерпретируемые языки программирования
- Сравнение технологий JAVA и Microsoft .NET Framework
- Понятие CLR, IL

Тема 2 «Обзор языка MSIL, изучение конструкций данного языка для простого примера программы на языке C#». Рассмотреть следующие вопросы:

- Описание CTP
- Основные компоненты MSIL
- Общая система типов, как механизм генерации языков программирования для платформы Microsoft .NET Framework

Тема 3 «Закрепление навыков работы с операторами цикла и условными операторами C#». Рассмотреть следующие вопросы:

- Бесконечный оператор цикла, варианты реализации
- Ключевые слова для управления ходом выполнения оператора цикла
- Оператор foreach и интерфейс IEnumerable

Тема 4 «Конструкторы класса. Перегрузка ме-тодов класса. Виртуальные методы. Наследование. Закрытые от наследования классы». Рассмотреть следующие вопросы:

- Конструкторы класса, конструктор копирования
- Последовательность вызова конструкторов класса
- Поля только для чтения и взаимодействие с конструкторами
- Наследование и виртуальные методы, переопределение и перекрытие методов

Тема 5 «Рефлексия. Доступ к свойствам объекта. Зубчатые массивы». Рассмотреть следующие вопросы:

- Методы доступа к перечню полей класса в процессе работы программы
- Получение перечня полей объекта класса и их типов
- Получение информации о типе объекта

Тема 6 «Изучение основ Microsoft SQL Server. Знакомство с дизайнером LINQ to SQL. Знакомство с технологией LINQ To Objects». Рассмотреть следующие вопросы:

- Основные характеристики СУБД Microsoft SQL Server
- Обзор среды Management Studio for Microsoft SQL Server
- Особенности языка TSQL
- Технология доступа к данным LINQ, обзор LINQ to SQL, LINQ to Objects

Тема 7 «Изучение правил формирования CSS. Обзор Javascript, AJAX и JQuery». Рассмотреть следующие вопросы:

- Назначение версии каскадных таблиц стилей CSS
- Нововведения в последней версии CSS
- Назначение правил по идентификаторам, классам, а так же элементам типа
- Нововведения в HTML 5
- Основные элементы языка Javascript
- Библиотека JQuery для манипулирования HMTL и стилями
- Технология асинхронных запросов к серверу AJAX

Тема 8 «Разработка примеров для сравнения технологий разработки программного обеспечения (MVVM, MVP и MVC)». Рассмотреть следующие вопросы:

- Основная концепция технологии MVVM
- Основная концепция технологии MVP
- Основная концепция технологии MVC

Тема 9 «Регистрация в Windows Azure Cloud, настройка учетной записи. Настройка ресурсов». Рассмотреть следующие вопросы:

- Технологии облачных сервисов
- Сервисы технологии Windows Azure Cloud

СРС экз. – самостоятельная работа по подготовке к экзамену в период экзаменационной сессии – 53,65 ч.

Экзаменационные вопросы по курсу

1. Определение и назначение общеязыковой исполняющей среды (CLR) .NET Framework?
2. В чем состоит отличие компилируемых от транслируемых языков?
3. Определение и назначение общей системы типов (CTS) .NET Framework?
4. Определение и назначение общеязыковой спецификации (CLS) .NET Framework?
5. Для чего используется MSIL?
6. В чем состоит отличие в реализации платформы .NET Framework 4.5.1 от остальных ранее реализованных версий платформы?
7. Какие типы приложений поддерживает платформа .NET Framework?
8. В чем состоит отличие процедурных языков от языков объектно-ориентированного программирования?
9. Что такое «Абстракция»? Привести пример.
10. Что такое «Инкапсуляция»? Привести пример.
11. Что такое «Полиморфизм»? Привести пример.
12. Назначение наследования, особенности наследования в языке C#. Привести пример.
13. Привести пример описания класса на языке C#, дать определение полям, свойствам и методам класса.
14. Что такое тип «var»? Особенности работы с типом «var».
15. Перечислить основные методы, определенные для класса string. Привести примеры.

16. Что такое приведение типов? Какие виды приведения типов существуют? Привести примеры.

17. Перегрузка методов в языке C#. Примеры

18. Конструкторы в языке C#, перегрузка конструкторов. Примеры.

19. Какие в C# существуют функции конвертирования значений из строковых в числовые? Привести примеры.

20. Что такое «поля только для чтения»? Привести примеры.

21. Какие условные операторы в C# существуют? Привести примеры.

22. Какие операторы выбора существуют в языке C#? Привести примеры.

23. Какие операторы цикла существуют в языке C#? Привести примеры.

24. Какой интерфейс должен наследовать класс, чтобы мог быть использовать для перечисления в операторе «foreach»? Привести примеры.

25. Можно ли использовать ключевое слово «return» в методах, не возвращающих значений? Привести примеры.

26. В чем отличие статических методов класса от нестатических? Привести примеры.

27. Что такое рекурсия? Привести примеры рекурсивных функций.

28. Для чего используется ключевое слово «this»?

29. Что такое «необязательный параметр функции»? Привести примеры.

30. Привести пример именованных параметров функции.

31. Для чего используется модификатор «params»?

32. Что такое «метод расширения»? Привести пример.

33. Для чего нужны асинхронные методы? Блокировки в асинхронных методах.

34. Какие способы определения и инициализации массивов существуют в C#? Привести пример определения одномерного массива.

35. Способы перечисления элементов массива. Примеры использования массивов с оператором «for» и «foreach».

36. Какие реализации технологии LINQ существуют? Коротко перечислить основные особенности реализаций.

37. Привести примеры запросов с поиском по полям класса (оператор «Where») в LINQ to Objects.

38. Привести примеры запросов с использованием оператора «Select» в LINQ to Objects.

39. Что такое «XML»? Основные особенности и элементы.

40. Что такое сериализация? Привести пример.

41. Что такое десериализация? Привести пример.

42. Как можно контролировать процесс сериализации и десериализации?

43. Перечислить основные особенности и назначение платформы ASP.NET.

44. Какие виды приложений поддерживаются платформой ASP.NET?

45. Основные элементы веб-приложения, построенного на платформе ASP.NET.

46. Что такое «веб-сервис», для чего используется?

47. По какому протоколу работают «веб-сервисы»? Особенности протокола.

48. Привести пример создания веб-сервиса и методов веб-сервиса.

49. Обращение к веб-сервису. Создание прокси-класса для работы с веб-сервисом в Visual Studio.

50. Для чего предназначена технология Windows Communication Foundation?

51. В чем отличие сервисов WCF от веб-сервисов «.asmx»?

52. Привести пример создания WCF сервиса.

53. Привести пример создания клиентского приложения для работы с WCF сервисом.

54. Для чего необходим HTML, какой стандарт используется в настоящее время?

55. Что такое тэг HTML? Основные тэги HTML.

56. Что такое CSS и для чего используется?

57. Серверные тэги ASP.NET веб-приложения, использование HTML тэгов в ASP.NET приложении. Привести примеры.

58. Что такое AJAX?

59. Для чего используется технология «PageMethods»?

60. Привести примеры использования асинхронных запросов в ASP.NET.

61. Для чего используется технология WWF?
62. Основные понятия и элементы технологии WWF.
63. Для чего используется WPF?
64. Дать основные характеристики языка XAML. Привести примеры.
65. Какие средства создания WPF приложений существуют?
66. Какие типы приложений на технологии WPF существуют?
67. Особенность модели MVC. Привести пример.
68. Особенность модели MVP. Привести пример.
69. Особенность модели MVVM. Привести пример.
70. Для чего применяются облачные технологии, в чем состоит преимущество подобных технологий?
71. Какие сервисы предоставляет Windows Azure Cloud?
72. Очереди обмена сообщениями. Привести пример размещения сообщения в очереди.
73. Очереди обмена сообщениями. Привести пример считывания сообщения из очереди.
74. Что такое «Рефлексия»? Можно ли получить динамически информацию о составе полей класса? Привести пример.
75. Атрибуты полей класса, для чего используются.
76. Перечисления, пример определения перечислений в языке C#.
77. Формирование описаний для перечислений в языке C#. Привести пример.
78. Частичные классы. Привести пример.
79. Частичные методы. Привести пример.
80. Свойства класса, методы get и set, назначение и примеры использования.
81. Свойства только для чтения. Привести пример.
82. Делегаты, дать определение, описать назначение. Привести пример.
83. События. Для чего используются. Привести пример определения события.
84. Обработчики события. Привести пример.
85. Можно ли задавать последовательность выполнения обработчиков событий?
86. Анонимные функции, для чего используются, привести пример.

87. Функции запроса очистки памяти сборщиком мусора.

88. Классы для выполнения запросов по HTPP протоколу (WebClient). Привести пример обращения к веб-ресурсу и получению ответа.

89. Для чего необходимы интерфейсы? Привести пример.

90. Какие классы ответственны за шифрование в языке C#? Привести пример шифрования и дешифрования строки символов.

91. Статические классы, для чего используются, особенности, привести пример.

92. Модификаторы доступа, перечислить, привести примеры.

93. Можно ли использовать язык C# для разработки мобильных приложений? Под какие платформы и с помощью каких библиотек?

94. LINQ to XML, особенности технологии. Привести пример поиска тэгов в XML документе.

95. Привести пример записи объектов в таблицу базы данных с использованием LINQ to SQL.

96. Привести пример удаления объектов таблицы базы данных с использованием LINQ to SQL.

97. Привести пример обновления объектов таблицы базы данных с использованием LINQ to SQL.

98. Какие языки реализованы для платформы .NET Framework?

99. Под какие операционные системы существуют реализации платформы .NET Framework?

100. Привести пример использования метода int.TryParse.

Контактная внеаудиторная работа

СРС – групповые консультации с преподавателем в течение семестра – 0,9 ч.

Групповая консультация перед экзаменом – 2 ч.

СРС экз. – сдача экзамена – 0,35 ч.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Иванченко А.Н., Масленников А.А., Шайда А.Ю. Программирование на платформе .NET: учебное пособие по курсу «Программирование» [Электронный ре-

сурс]: А.Н. Иванченко, А.А. Масленников, А.Ю. Шайда – Юж.-Рос. гос. политехн. ун-т (НПИ) – Новочеркасск: ЮРГПУ(НПИ), 2014. - 88 с. - Режим доступа: http://lib.npi-tu.ru/

Дополнительная учебная литература

2. Баженова И. Ю., Сухомлин В. А. Введение в программирование [Электронный ре-сурс]: И. Ю. Баженова, В. А. Сухомлин– М.:Национальный Открытый Университет «ИНТУИТ», 2007. - 327 с. - Режим доступа: http://www.knigafund.ru/books/178416.

3. Биллинг В. А. Основы программирования на C# [Электронный ресурс]: В. А. Биллинг – М.:Национальный Открытый Университет «ИНТУИТ», 2006. - 485 с. - Режим доступа: http://www.knigafund.ru/books/178648.

4. Макаров А.В. Common Intermediate Language и системное программирование в Microsoft.NET: Учебное пособие [Электронный ресурс]: А. В. Макаров, С. Ю. Скоробогатов, А.М. Чеповский – М.:Интернет-Университет Информационных технологий, 2006. - 328 с. - Режим доступа: http://www.knigafund.ru/books/176948.

5. Павлова Е.А. Технологии разработки современных информационных систем на плат-форме Microsoft.NET: Учебное пособие [Электронный ресурс]: Е. А. Павлова – М.:Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2009. - 112 с.: ил., табл. – (Серия «Основы информационных технологий») - Режим доступа: http://www.knigafund.ru/books/178359.

6. Сафонов В. О. Платформа облачных вычислений Microsoft Windows Azure [Элек-тронный ресурс]: В. О. Сафонов – М.:Национальный Открытый Университет «ИНТУИТ», 2011. - 293 с. - Режим доступа: http://www.knigafund.ru/books/177115.

Учебно-методическое издание

Масленников Алексей Александрович

Программирование

Методические указания
к практическим занятиям, лабораторным работам
и самостоятельной работе студентов бакалавриата
направления подготовки «Математическое обеспечение
и администрирование информационных систем»,
«Информатика и вычислительная техника»
и «Программная инженерия»

Редактор *Н.А.Юшко*
Подписано в печать 21.12.2016
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,39. Уч.-изд.л. 1,5. Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

РАЗРАБОТКА ПРОЕКТОВ В ОБЛАСТИ КОМПЬЮТЕРНЫХ НАУК

Методические указания к практическим занятиям, лабораторным работам, самостоятельной работе и курсовому проектированию

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 004.4

Рецензент – кандидат технических наук, доцент, доцент кафедры «Информационные и измерительные системы и технологии» **Скоба Андрей Николаевич**

Мохов В.А.

Разработка проектов в области компьютерных наук: методические указания к практическим занятиям, лабораторным работам, самостоятельной работе и курсовому проектированию / В.А. Мохов; Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2017. – 30 с.

Приведено описание практических занятий, программ лабораторных работ, тем для самостоятельной работы, а также содержание курсовой работы. Даны вопросы для самоконтроля.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 004.4

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

1.1. Очная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля	Литература
1	Генерация идей для проектов: мозговой штурм	2	Защита отчета	15-20.03	7 [7]
2	Организация групповой работы над проектом: формирование коллектива и выбор ролей	2	Защита отчета	15-20.03	7 [7]
3	Коллективный анализ идеи: разработка диаграммы прецедентов	2	Защита отчета	15-20.04	7 [7]
4	Разработка перечня функциональных требований	2	Защита отчета	15-20.04	7 [7]
5	Временное планирование проекта: определение списка работ и квалификационная оценка затрат	2	Защита отчета	15-20.04	7 [7]
6	Формирование перечня ответственных за процессы	2	Защита отчета	15-20.05	7 [7]
7	Основы правильного выступления при защите проекта	2	Защита отчета	15-20.05	7 [7]
8	Организация защиты проекта	2	Защита отчета	15-20.05	7 [7]

Практическое занятие № 1

Генерация идей для проектов: мозговой штурм

1. Сформировать группу участников (4÷10 человек). Группа «штурмующих» по возможности состоит из представителей, разного пола, темперамента и склада мышления. Именно разнородный состав участников позволяет формировать решения, существенно отличающиеся от привычных.

2. Выбрать/сформулировать проблему, которую необходимо решить. Проблема обычно записывается на доске или представляется в виде плаката.

3. Проведение собственно штурма. Цель: получить как можно больше идей. На этом этапе вводится основное правило – приветствуются любые, самые безумные, явно ошибочные идеи, критика категорически запрещена. Наоборот, любую высказанную мысль нужно поощрять. Такая поддержка должна стимулировать творческий процесс.

Ведущий/модератор руководит процедурой, подбадривает, эмоционально заводит участников.

Важно четко перед началом этого этапа определиться с регламентом. Обычно на штурм идей отводится 15÷30 минут. Участники должны знать, что время ограниченно, и им необходимо выдать как можно больше идей в сжатые сроки. Это активизирует их, заставит выложиться в отведенное время.

4. Выполнение анализа. Из N-го количества идей полученного на этапе №3 (например, 15÷75 идей, в зависимости от количества участников штурма), необходимо выбрать самые «многообещающие», перспективные идеи, и развить их. Критика, наконец, разрешена, но в каждой идее необходимо попытаться найти конструктивное начало.

Практическое занятие № 2

Организация групповой работы над проектом: формирование коллектива и выбор ролей

1. Внимательно прочитайте приведенные утверждения и напротив каждого дать свой ответ:

– два очка – если данное утверждение полностью соответствует вашим деловым качествам;

– одно очко – если утверждение соответствует лишь частично;

– ноль очков – если данное утверждение не характеризует ваши деловые качества.

№	Утверждение	Кол-во баллов
1.	Я делаю оригинальные предложения	
2.	Я нахожу промахи и оплошности, которые другим не удается заметить	
3.	У меня есть способность организовывать других людей	
4.	Я выдвигаю идеи, которые могут иметь более широкое применение, чем в непосредственно решаемой задаче	
5.	Мое продуманное суждение потребует времени, но оно обычно довольно точно	
6.	Я люблю оказывать сильное влияние на решения коллектива	
7.	На меня можно положиться, чтобы увидеть работу выполненной	
8.	Я могу работать с людьми, которые имеют большое разнообразие личностных качеств и мировоззрений	
9.	Я часто замечаю, что мое воображение рассеивается при совместной работе в коллективе	
10.	Я могу работать с любыми людьми, если они только могут внести нечто стоящее в коллективную работу	

11.	Множество разнообразных контактов с другими людьми важно для моего стиля работы	
12.	Мои чувства редко мешают моим суждениям	
13.	Я пытаюсь выразить свои взгляды в ходе собраний коллектива	
14.	Сталкиваясь с трудностями, я борюсь, чтобы выполнить работу качественно и достичь целей.	
15.	Я умею хорошо детализировать	
16.	Я заинтересован в том, чтобы помогать своим товарищам в их проблемах	
17.	Я критически анализирую идеи других людей, как их достоинства, так и недостатки	
18.	Я часто отмечаю у себя новые подходы к давним проблемам	
19.	Я могу координировать и продуктивно использовать способности и таланты своих товарищей	
20.	Я заинтересован в завершении и совершенствовании всех своих начинаний	
21.	Я получаю особое удовольствие от изучения идей и технических приемов, которые являются для меня новыми	
22.	Мне трудно полностью отдаться работе, цели которой не определены четко и ясно	
23.	Я не очень заинтересован в лучшем познании людей	
24.	Мне кажется, что иногда стоит рискнуть и навлечь на себя некоторую временную непопулярность, чтобы преуспеть в распространении своих взглядов в коллективе	
25.	У меня творческий подход к решению проблем	
26.	Я больше заинтересован в практическом осуществлении имеющихся идей, чем в выдвижении новых	
27.	Мне кажется, что мои личные умения способствуют достижению согласия в коллективе	
28.	Обычно я могу найти аргумент, чтобы отклонить неразумные предложения	
29.	Я знаю, чьи специальные взгляды и умения особенно нужны для решения стоящих перед коллективом проблем	
30.	Я больше исполнитель, чем инициатор	
31.	Я чувствую себя в своей стихии, когда работа требует высокой степени концентрации и внимания	
32.	Я охотно подчеркиваю собственную точку зрения на собраниях коллектива	
33.	Я хорошо уживаюсь с другими людьми и упорно работаю ради коллектива	
34.	В большинстве ситуаций я имею независимый и новаторский взгляд	
35.	Я сохраняю устойчивый систематический подход, каким бы ни было давление	
36.	У меня довольно напористый характер, но я реагирую на нужды других людей	
37.	Я счастлив быть лидером в ситуациях, требующих действий	
38.	Я внимательно обдумываю новые идеи в их развитии	
39.	Я люблю тщательно взвешивать и оценивать все предложения своих товарищей, прежде чем сделать выбор	
40.	Быть занятым решением деловой проблемы доставляет мне истинное удовольствие	
41.	Я могу определить, какие идеи и технические приемы могут быть использованы в новых ситуациях	

42.	Я сильно реагирую, когда собрание, похоже, теряет из виду главную цель	
43.	Мне доставляет удовольствие сопоставлять различные точки зрения	
44.	Я всегда готов поддерживать хорошие предложения	
45.	У меня развито естественное чувство срочности при выполнении любой работы	
46.	Я придаю особое значение следованию интересным идеям (людям)	
47.	Я предпочитаю тщательно анализировать возникающие проблемы	
48.	Мне кажется, что у меня есть талант планирования конкретных шагов решения проблемы	
49.	Я думаю, что могу рекламировать и популяризировать идеи, которые интересуют меня	
50.	Я могу увидеть систему там, где другие обычно видят несвязанные предметы	
51.	Я готов волевым путем огласить мою личную точку зрения, если это необходимо	
52.	Я вижу все стороны проблемы и предлагаю решения, приемлемые для всех	
53.	Я внимательно слежу за участками, где может потребоваться моя помощь	
54.	Я люблю выявлять критические различия между альтернативами	
55.	Я работаю наилучшим образом, когда цели и задачи ясно определены	
56.	Обычно я внимательно слежу за участками, где могут возникнуть трудности	

2. Поставьте число баллов в соответствующие строки, а затем подсчитайте сумму по приведенным ниже группам утверждений. Количественная выраженность очков показывает вашу потенциальную возможность исполнять ту или иную социальную роль при совместной работе в коллективе. Если вы получили примерно одинаковое число баллов по нескольким позициям, это означает. Что в зависимости от ситуации вы можете играть в команде разные роли (многие роли не являются взаимоисключающими).

Номера утверждений	**Количество баллов**	**Социальная роль**
3, 10, 19, 27, 36, 43, 52		«Лидер»
6, 13, 24, 32, 37, 42, 51		«Реализатор»
1, 9, 18, 25, 34, 41, 50		«Генератор идей»
5, 12, 17, 28, 39, 47, 54		«Объективный критик»
7, 14, 22, 26, 35, 48, 55		«Организатор»
4, 11, 21, 29, 38, 46, 49		«Снабженец»
8, 16, 23, 30, 33, 44, 53		«Душа коллектива»
2, 15, 20, 31, 40, 45, 56		«Отделочник»

3. Разделите аудиторию на группы по 5÷7 человек так, чтобы в каждой группе был «полный комплект» социальных ролей.

Практическое занятие № 3

Коллективный анализ идеи: разработка диаграммы прецедентов

1. Идентифицируйте участников (акторов), окружающих проектируемую систему/подсистему. Для этого выделите и проведите анализ групп, заинтересованных в выполнении системой своих функций.

2. Сгруппируйте (организуйте) похожих участников с помощью отношений обобщения/специализации.

3. Сформируйте и разместите на диаграмме прецеденты (варианты использования) системы. Прецедент представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое актором.

4. Разместите участников на диаграмме и определите их связи с прецедентами системы.

При выполнении пп. 1÷4 необходимо соблюдать ряд правил:

– показывать на диаграмме прецедентов действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования;

– моделировать связи между действующими лицами нельзя (действующие лица находятся вне сферы действия системы и поэтому связи между ними разработчику системы не подконтрольны);

– соединять стрелкой непосредственно два варианта использования для определения порядка их выполнения нельзя (диаграмма прецедентов для этого не предназначена);

– каждый вариант использования должен быть инициирован действующим лицом (исключением являются связи использования и расширения).

Практическое занятие № 4

Разработка перечня функциональных требований

1. Сформулируйте бизнес-требования (business requirements) к проектируемой системе. Эти требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. На практике часто бизнес-требования записываются в форме документа об образе и границах проекта, который еще иногда назы-

вают уставом проекта (project charter) или документом рыночных требований (market requirements document).

2. Составьте перечень требований пользователей (user requirements). Эти требования пользователей описывают цели и задачи, которые пользователям даст система. К отличным способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие – отклик». Таким образом, в этом документе указано, что клиенты смогут делать с помощью системы.

3. Определите функциональные требования (functional requirements). Они определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Иногда они называются требованиями поведения (behavioral requirements), они содержат положения с традиционным «должен» или «должна» (например: «Система должна по электронной почте отправлять пользователю подтверждение о заказе»).

Функциональные требования документируются в спецификации требований к ПО, где описывается так полно, как необходимо, ожидаемое поведение системы.

Функциональные требования. Это перечень сервисов, которые должна выполнять система, причем должно быть указано, как система реагирует на те или иные входные данные, как она ведет себя в определенных ситуациях и т.д. В некоторых случаях указывается, что система не должна делать.

4. Подготовьте перечень системных требований (system requirements). Данные требования – это высокоуровневые требования к продукту, которые содержат многие подсистемы. Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди – часть системы, поэтому определенные функции системы могут распространяться и на людей.

5. Разработайте бизнес-правила (business rules), которые включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные ВИ, или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества,

которые реализуются в функциональности. Следовательно, вы можете отследить происхождение конкретных функциональных требований вплоть до соответствующих им бизнес-правил.

6. В заключении сформулируйте нефункциональные требования, описывающие цели и атрибуты качества. Атрибуты качества (quality attributes) представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся: легкость и простота использования, легкость перемещения, целостность, эффективность и устойчивость к сбоям, внешние взаимодействия между системой и внешним миром, а также ограничения дизайна и реализации. Ограничения (constraints) касаются выбора возможности разработки внешнего вида и структуры продукта.

Практическое занятие № 5

Временное планирование проекта: определение списка работ и квалификационная оценка затрат

1. На основании перечня функциональных требований из практической работы № 4 определите список работ, необходимых для реализации проекта. Для этого определение и задокументируйте работы, запланированные к выполнению. Этот список будет основой для составления смет, планирования сроков выполнения и контроля проектных работ.

Степень детализации списка работ зависит от цели детализации. Детализация операций для разработки иерархического расписания крупного проекта будет существенно отличаться от степени детализации при разработке расписания выполнения малого проекта. Степень детализации также зависит от количества контрольных событий, которые планируется отразить в расписании проекта.

Состав операций может определяться последовательно, методом набегающей волны. Этот метод применяется в крупных или долгосрочных проектах, когда имеется неопределенность относительно выполнения некоторых работ. При использовании метода набегающей волны пакеты работ, расположенные в отдаленном будущем, планируются только на высоком уровне, в то время как пакеты работ, расположенные ближе по оси времени, планируются детально. Этот метод рекомендуется применять при создании детальных планов на стадии разработки и производства.

Исходной информацией для процесса определения списка работ являются:

– методология внедрения информационной системы;

– контракт;

– описание содержания проекта;

– иерархическая структура работ;

– словарь иерархической структуры работ.

Для определения списка работ используют следующие инструменты и методы:

– декомпозиция;

– шаблоны;

– планирование методом набегающей волны;

– экспертная оценка.

Процесс определения списка работ завершается формированием списка операций и уточненным списком контрольных событий.

Список операций - перечень работ, запланированных для выполнения. В список операций входят идентификатор операции и описание содержания работ по каждой операции, подробное настолько, чтобы члены команды проекта понимали, какие работы необходимо провести.

Список контрольных событий - перечень основных событий, которые должны быть включены в расписание для мониторинга хода выполнения и управления проектом, с указанием, является ли контрольное событие обязательным (например, необходимым согласно контракту) или необязательным (например, основывающимся на исторической информации).

Пример списка работ представлен в таблице ниже:

№пп	**Наименование пакета работ**	**Наименование операций**
1	Обследование	• Формирование и согласование плана проведения интервью • Подготовка и рассылка опросных листов для интервью. Проведение интервью для описания бизнес-процессов
2	Описание бизнес-процессов	• Описание бизнес-процессов по функциональной области «*Финансы»* • Описание бизнес-процессов по функциональной области «*Логистика»* • Описание бизнес-процессов по функциональной области «*Персонал»*
3	Разработка системы	• Разработка решений по функциональной архитектуре. Подготовка функционального дизайна расширений. Настройка системы

		• Техническое проектирование расширений. Разработка расширений • Техническое проектирование программ конвертации данных • Разработка программ конвертации данных. Планирование тестирования приложения и интеграционного тестирования
4	Тестирование системы	• Разработка сценариев тестирования • Подготовка тестовых данных • Проведение тестирования по функциональным областям «*Финансы», «Логистика», «Персонал»* • Проведение интеграционного тестирования • Проведение тестирования конвертации данных

2. В соответствии со списком спланированных работ, подготовленных на предыдущем этапе, выполните квалификационную оценку затрат. Решение такой задачи предполагает выполнение временной оценки выполнения операций специалистами проектной команды в соответствии с их квалификацией. Важным моментом является оценка возможности и обеспечение временного распараллеливания ряда запланированных работ по проекту.

Процесс оценки и его результат в значительной степени зависят от точности описания содержания, качества доступной информации, от стадии проекта. Оценка проекта начинается на предпроектной стадии (до заключения контракта) и выполняется в течение всего времени выполнения проекта.

Практическое занятие № 6

Формирование перечня ответственных за процессы

Из состава команды выполнения проекта сформируйте список лиц, ответственных за операции из списка работ, определённого в лабораторной работе № 5. При этом важно понимать, что количество запланированных операций должно быть достаточным для того, чтобы ответственный за пакет работ мог отслеживать ход исполнения и осуществлять координацию работ. Число операций не должно быть слишком большим, затрудняющим оценку общего состояния проекта (например, команда может принять решение и ограничить количество операций проекта – не более 30, при этом любая операция должна иметь продолжительность не более 20 дней и не менее 10 дней и т.д.).

Практическое занятие № 7

Основы правильного выступления при защите проекта

1. Спланируйте структуру и содержательную часть выступления для защиты Вашего проекта. Подготовьте и отрепетируйте текст выступления.

2. Защита проекта является публичной презентацией результатов Вашей деятельности в течении 5÷7 минут. Примерная структура доклада по проекту включает:

– краткое представление;

– название проекта, его цель и концепция;

– структура проекта и его основное содержание;

– преимущества Вашего проекта;

– заключение.

Очерёдность приводимых аргументов влияет на их убедительность. Наиболее убедителен порядок аргументов: сильные, средние, самый сильный.

Практическое занятие № 8

Организация защиты проекта

1. К докладу на защиту проекта подготовьте презентацию на 10÷12 слайдов.

2. Подготовьтесь и выполните защиту проекта.

Презентация и защита индивидуальных проектов проводится на открытом заседании студенческой группы в рамках часов, отведённых на учебную дисциплину, на последнем аудиторном занятии. Для проведения процедуры защиты создаётся комиссия, в состав которой входит преподаватель-предметник, а также в неё могут входить другие преподаватели кафедры, администрация Университета и иные квалифицированные работники.

Комиссия оценивает уровень проектной деятельности конкретного обучающегося, дает оценку выполненной работы по проекту. Процедура защиты состоит в выступлении обучающегося с демонстрацией презентации, после которой следуют ответы на вопросы комиссии.

Проектная деятельность оценивается по двум критериям: критерию оценки содержания проекта и критерию оценки защиты проекта.

1.2. Заочная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Генерация идей для проектов: мозговой штурм	2	Защита отчета	В период лабораторно-экзаменационной сессии
2	Организация групповой работы над проектом: формирование коллектива и выбор ролей	2	Защита отчета	

Практическое занятие № 1

Соответствует практическому занятию № 1 из раздела 1.1.

Практическое занятие № 2

Соответствует практическому занятию № 2 из раздела 1.1.

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

2.1. Очная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Программа достижения цели: диаграмма Исикавы	2	Защита отчета	15-20.03
2	Технология разработки простых онтологических моделей	2	Защита отчета	15-20.03
3	Онтологическое моделирование в редакторе Protege	4	Защита отчета	15-20.04
4	Разработка диаграммы Ганта	4	Защита отчета	15-20.04
5	Процессное моделирование разработки	4	Защита отчета	15-20.05

Лабораторная работа №1

Программа достижения цели: диаграмма Исикавы

Цель работы: практическое освоение технологии разработки программ достижения цели с использованием технологии диаграмм Исикавы.

Программа работы

1. Нарисуйте диаграмму Исикавы («рыбью кость»), описывающую программу достижения цели Вашего проекта. Начинайте справа, строя основные «кости» (категории) по направлению влево. В качестве программного средства для исполнения диаграммы возможно использовать MS Visio, бесплатную версию XMind или иной доступный программный продукт/сервис.

2. Напишите цель вашего проекта в «голове рыбьей кости» (нужна только та часть, которая касается результата постановки проблемы).

3. Определите основные категории «рыбьей кости», которые относятся к данному результату.

4. Распределите основные категории в нисходящем порядке, начиная с той категории, которая имеет наибольшую вероятность того, что она вызвала потенциальную корневую причину, например рис. 1.1.

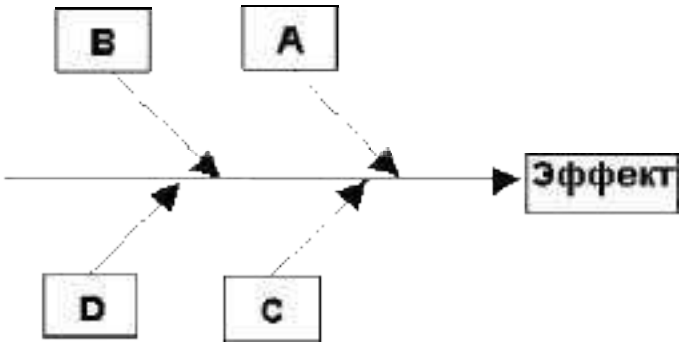


Рис. 1.1. Пример диаграммы «рыбья кость»

На диаграмме «рыбья кость» с четырьмя основными категориями или «костями» порядок приоритета будет таким: A, C, B, D. Следовательно, A – это основная «кость», расположенная в наибольшей близости к «голове рыбы», за ней следуют C, B, D.

Такой порядок приоритета особенно полезен в дальнейшем, когда будет вновь просматриваться «рыбья кость». Если человек, просматривающий диаграмму, может проследить логическую цепочку в построении «рыбьей кости», то он может проследить процесс мышления команды и предложить лучший диагностический совет команде.

5. После того, как «рыбья кость» составлена, начните с основной категории, которую разработчик (команда) определил в качестве наиболее вероятной, которая вызвала корневую причину (категория, находящейся в наибольшей близости к «голове ры-

бы»). Начните задавать вопрос «почему». Почему это происходит? Почему такое состояние существует?

Обязательно проследите логику вашей диаграммы в обоих направлениях как показано на рисунке ниже (а1 вызвано посредством а2, которое, в свою очередь, вызвано посредством а3. В обратном порядке а3 вызвало а2, которое, в свою очередь, вызвало а1.) Очень часто невозможно понять логику диаграммы, не проследив ее в обратном направлении (рис. 1.2).

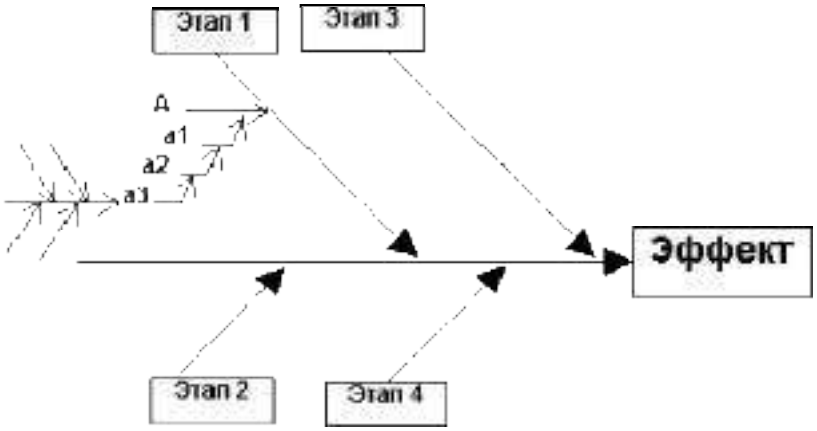


Рис. 1.2. Логика построения диаграммы

Далее просмотрите каждую «подкость», чтобы обнаружить дополнительные причины; т.е. перейдите к а2 и задайте вопрос «Почему происходит а2?». Затем задайте вопрос «Почему происходит а1?» и продолжайте процесс запрашивания, продвигаясь к основной «кости».

6. Прежде чем переходить к этапу 7, завершите анализ всей диаграммы «рыбья кость».

7. Определите наиболее вероятные корневые причины и обведите последний элемент в цепочке.

8. Удостоверьтесь с помощью данных в наиболее вероятной корневой причине. Команды должны собрать данные, чтобы удостовериться, что это, действительно, корневая причина «результата» причины. Если потенциальная причина содержит в себе множество сложных подпричин, то разбейте вашу диаграмму на ряд отдельных диаграмм.

Варианты заданий

Варианты тем для разработки проектов приведены ниже:

Вариант	Тема
1	Информационная система ЖКХ
2	Информационная система организации перевозок
3	Информационная система планирования заказов
4	Информационная система управления заказами
5	Информационная система малого предприятия
6	Программно-технический комплекс GPS-мониторинга
7	Программно-технический комплекс рассылки SMS-сообщений
8	Программно-технический комплекс обмера габаритов помещений
9	Программно-технический комплекс «Умный дом»
10	Программно-технический комплекс аэровидеосъёмки

Темы для разработки студенческих проектов являются сквозными для выполнения всего комплекса лабораторных и практических работ, а, возможно, и курсового проектирования.

При этом тематика студенческих проектов может быть расширена инициативными темами, согласованными в рабочем порядке с преподавателем-предметником.

Лабораторная работа № 2

Технология разработки простых онтологических моделей

Цель работы: изучить и освоить базовые понятия онтологического подхода.

Программа работы

1. Для выбранной предметной области, в рамках которой выполняется студенческий проект, выделить 25÷30 понятий.
2. Дать определения этим понятиям (составить словарь).
3. На множестве понятий ввести отношения и функции интерпретации для построения онтологии по предметной области.
4. Осуществить поиск информации по разработанной предметной онтологии.
5. В отчет по лабораторной работе включить обзор по методам и средствам онтологического проектирования и возможностям инструментальных средств, а также разработанный онтологический каркас (графическое представление онтологии для выбранной предметной области и словарь понятий, полученный в результате проектирования).

6. Подготовить отчет для защиты лабораторной работы.

Варианты заданий

Варианты тем для разработки онтологических каркасов выбираются в соответствии с определённой тематикой студенческого проекта из таблицы заданий к лабораторной работе № 1 или инициативной темой, согласованной в рабочем порядке с преподавателем-предметником

Лабораторная работа № 3

Онтологическое моделирование в редакторе Protege

Цель работы: освоить основные приёмы работы с инструментальными средствами онтологического проектирования.

Программа работы

1. На базе разработанного в предыдущей лабораторной работе онтологического каркаса разработайте онтологическую модель. Для этого используйте язык и инструментальные средства онтологического проектирования (редактор онтологий Protégé).
2. С помощью одного из доступных в Protege ризонеров (систем логического вывода) сгенерируйте RDF файл с восстановленными связями.
3. Протестируйте построенную сеть, используя базовые SparQL запросы.

Варианты заданий

Вариант задания для онтологического моделирования выбирается в соответствии с определённой ранее тематикой и результатами выполнения лабораторной работы № 2.

Лабораторная работа № 4

Разработка диаграммы Ганта

Цель работы: освоение методики и инструментов планирования и иллюстрации графика работ разрабатываемого проекта.

Программа работы

1. На основе сквозного индивидуального задания разработайте (уясните) план выполнения работ по проекту. Для этого соберите необходимую информацию обо всех шагах или процессах, составляющих проект.

2. Рассчитайте сроки. Глядя на собранную информацию, определите, как долго будет длиться каждый шаг или процесс в проекте. Добавьте это на общую диаграмму и установите длину соответствующих полосок, которые представляют собой фазы или процессы проекта.

3. Сдвиньте сроки результатов работ. Помня об общих временных рамках для каждого шага или процесса, расположите их в общей диаграмме Ганта так, чтобы уложиться в конечные сроки для каждого из элементов. Это может помочь немного разнести во времени процессы, чтобы обеспечить более плавные переходы или избавиться от дат с большим количеством привязанных к ним окончаний работ, что может запросто запутать менеджера проекта.

4. Разместите все полоски на диаграмме. Диаграмма Ганта собирает вместе все части, относящиеся к конкретной фиксированной дате завершения работы. В рамках сроков всего проекта для тех, кто смотрит на диаграмму Ганта, сроки по отдельным процессам становятся более ясными.

5. Оцените зависимости или взаимоотношения между фазами или процессами проекта.

6. Вручную, или используя различные средства автоматизации (MS Excel, MS Visio, OO Calc, html-редакторы, языки программирования и т.д.), постройте диаграмму Ганта для своего проекта.

7. При необходимости воспользуйтесь примерами построения диаграммы Ганта с помощью Excel (например, http://it.kgsu.ru/MSExcel/excel119.html).

Варианты заданий

Вариант задания для формирования диаграммы Ганта выбирается в соответствии с определённой ранее тематикой и результатами выполнения практической работы №5 «Временное планирование проекта: определение списка работ и квалификационная оценка затрат».

Лабораторная работа № 5

Процессное моделирование разработки

Цель работы: построение модели проектируемой информационной системы с помощью методологии процессного моделирования.

Программа работы

1. Ознакомьтесь с методологией процессного (функционального) моделирования работ, диаграмм потоков данных для проектирования информационных систем (IDEF0), например, здесь http://informatic.ugatu.ac.ru/gefest/arm/mag2011/labwork/lab_2.pdf.
2. Изучите возможности представления методологии IDEF0 в бесплатных пакетах Ramus Educational (http://ramussoftware.com) или StarUML (http://staruml.io/).
3. Проанализируйте предметную область, в которой выполняется ваш проект, уточнив и дополнив ее, руководствуясь собственным опытом, консультациями и другими источниками.
4. Выполните анализ задач и функций проектируемой системы в целом и функции каждой из подсистемы.
5. Постройте процессную модель (набор процессных диаграмм для всей информационной системы в целом и для отдельных подсистем, отражающих соответствующие логику и взаимоотношения подсистем).
6. Оформите итоговый отчёт.

Варианты заданий

Вариант задания для процессного моделирования диаграммы выбирается в соответствии с тематикой, определённой ранее.

2.2. Заочная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Программа достижения цели: диаграмма Исикавы	2	Защита отчета	В период лабораторно-экзаменационной сессии
2	Процессное моделирование Разработки	2	Защита отчета	

Лабораторная работа № 1

Соответствует лабораторной работе № 1 из раздела 2.1.

Лабораторная работа № 2

Соответствует лабораторной работе № 5 из раздела 2.1.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

3.1. Очная форма обучения

СРС по дисциплине «Разработка проектов в области компьютерных наук», включает следующие разделы:

– темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирования;

– самостоятельную работу по выполнению курсовой работы;

– самостоятельную подготовку к экзаменам.

№	Наименование тем	Литература
1	**Тема 1. Системная инженерия** Определения системной инженерии. Профессия системного инженера. Отличия системной инженерии от других дисциплин. Актуальные технологии командной разработки проектов. Современные аспекты использования иностранных языков при реализации ИТ-проектов	7 [1-6]
2	**Тема 2. Формализмы проектирования** Терминология и онтология. Математические формализмы. Моделеориентированность. Научность системной инженерии. Ситуационная инженерия методов. Методологическая действительность: дисциплины, практики, методы. Семь основных альф инженерного проекта	7 [1-6]
3	**Тема 3. Системный подход** Стейкхолдеры. Системы систем. Воплощение системы: компоненты, модули, размещения. Структура системы: разбиения. Обозначения систем. Практики изготовления (производства)	7 [1-6]
4	**Тема 4. Формирование команды проекта** Основные принципы формирования команды проекта. Основы генерации идей и поиска приверженцев. Роли исполнителей в проекте. Распределение ролей в проекте и инструктаж	7 [1-6]
5	**Тема 5. Анализ идеи** Определение круга потенциальных пользователей. Сбор требований по ролям. Диаграммы прецедентов. Разработка списка функциональных требований. Онтологическое моделирование	7 [1-6]
6	**Тема 6. Временное планирование проекта** Определение списка работ проекта. Квалифицированная оценка затрат. Диаграммы временного планирования (диаграммы Ганта)	7 [1-6]

7	**Тема 7. Проектирование системы** Процессные диаграммы в нотация IDEF0. Распределение ответственности за процессы. Разработка процессной модели проекта, формирование выгрузки, подготовка пакета проектной документации	7 [1-6]
8	**Тема 8. Представление и защита проекта** Подготовка презентации. Подготовка выступления. Публичная защита проекта. Способы оценки проектов	7 [1-6]

Выполнение курсовой работы

Учебным планом предусмотрено выполнение курсовой работы.

Курсовая работа является завершающей стадией изучения дисциплины, требует от студента практического применения знаний, умений и навыков по разработке проектов в области компьютерных наук, служит проверкой готовности студента к самостоятельному выполнению программных проектов и основой для реализации выпускной квалификационной работы бакалавра.

Курсовая работа выполняется в часы самостоятельной работы. Оформляется курсовая работа в виде пояснительной записки установленного образца объемом 25÷30 листов.

Содержание курсовой работы:

1. Проведение системного анализа предметной области (согласно заданию).
2. Разработка проектной программы достижения цели на основе диаграммы Исикавы.
3. Выполнение детального анализа идеи проекта с реализацией её описания в виде диаграммы прецедентов.
4. Разработка перечня функциональных требований к разработке и его итоговое представление в табличной форме.
5. Онтологическое моделирование планируемой разработки с визуализацией модели в виде таксономии и формализацией средствами редактора Protege.
6. Выполнение временного планирования проекта на основе диаграмм Ганта с определением списка работ и квалификационной оценкой затрат.
7. Процессное моделирование разработки, формирование выгрузки, подготовка пакета проектной документации.
8. Разработка перечня ответственных за процессы.

9. Оформление пояснительной записки и презентации для защиты проекта.
10. Публичная защита проекта.

Пояснительная записка к курсовой работе содержит следующие разделы:

1. Задание.
2. Введение.
3. Краткое описание анализа предметной области.
5. Описание этапов подготовки проект.
6. Графическое представление и описание результатов работы.
7. Список используемой литературы.
8. Материалы приложений.

Презентация для защиты проекта должна содержать сведения о разработчике, тему проекта, его контактные сведения и графическое представление результатов выполнения работы.

При выполнении курсовой работы студенты:

1) осуществляют выбор предметной области для реализации проекта в области компьютерных наук (самостоятельно или из перечня, рекомендованного в методических указаниях [7];
2) составляют и утверждают техническое задание на курсовую работу;
3) исследуют предметную область для реализации проекта;
4) разрабатывают диаграмму Исикавы;
5) разрабатывают диаграмму прецедентов;
6) составляют перечень функциональных требований;
7) разрабатывают онтологическую модель;
8) разрабатывают диаграмму Ганта;
9) разрабатывают процессную модель и пакет проектной документации к проекту;
10) составляют перечень ответственных за процессы;
11) оформляют пояснительную записку и презентацию;
12) защищают проект.

3.2. Заочная форма обучения

СРС по дисциплине «Разработка проектов в области компьютерных наук», включает следующие разделы:

– темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование;

– самостоятельная работа по выполнению курсовой работы;

– подготовка к зачёту в период лабораторно-экзаменационной сессии.

№	Наименование тем	Литература
1	**Тема 1. Модели организации команд** Административная модель. Модель хаоса. Открытая архитектура	7 [1-6]
2	**Тема 2. Общение в команде** Люди стремятся быть понятыми. Конфликты. Достижение компромисса и консенсуса. Принятие решений. Как добиться консенсуса? Мотивация и демотивация. Антипаттерны некомпетентности. Антипаттерны мнительности. Последствия применения антипаттернов. Корпоративная политика (наведение мостов)	7 [1-6]
3	**Тема 3. Планирование и контроль** Зачем надо планировать? План – ничто. Планирование – все! Что надо планировать? Как проверять и оценивать? Метрики проекта. Как и когда начинать планировать? Структурная декомпозиция работ. Создание СДР. Критерии СДР	7 [1-6]
4	**Тема 4. Системная инженерия** Соответствует теме № 1 из раздела 3.1	7 [1-6]
5	**Тема 5. Формализмы проектирования** Соответствует теме № 2 из раздела 3.1	7 [1-6]
6	**Тема 6. Системный подход** Соответствует теме № 3 из раздела 3.1	7 [1-6]
7	**Тема 7. Формирование команды проекта** Соответствует теме № 4 из раздела 3.1	7 [1-6]
8	**Тема 8. Анализ идеи** Соответствует теме № 5 из раздела 3.1	7 [1-6]
9	**Тема 9. Временное планирование проекта** Соответствует теме № 6 из раздела 3.1	7 [1-6]
10	**Тема 10. Проектирование системы** Соответствует теме № 7 из раздела 3.1	7 [1-6]
11	**Тема 11. Представление и защита проекта** Соответствует теме № 8 из раздела 3.1	7 [1-6]

Выполнение курсовой работы

Содержание и требования к выполнению курсовой работы определяется содержанием одноимённого подраздела из раздела 3.1.

Вопросы для оценивания знаний

1. Понятие управления и проекта.
2. История управления проектами.
3. Категории управления проектами.
4. Треугольник ограничений проекта.
5. Что не является проектами?
6. Что должен знать менеджер проекта?
7. Ролевая модель команды.
8. Организация командной работы.
9. Способы командной генерации идей.
10. Целеполагание и диаграммы Исискавы.
11. Инструментальные средства для организации командной работы.
12. Командные роли.
13. Команда заказчика.
14. Команда исполнителя.
15. Взаимодействие команд.
16. Структура «идеальной» команды проекта.
17. Модели организации команд: административная модель.
18. Модели организации команд: модель хаоса.
19. Модели организации команд: открытая архитектура.
20. Общение в команде: конфликты.
21. Общение в команде: достижение компромисса и консенсуса.
22. Общение в команде: принятие решений.
23. Общение в команде: как добиться консенсуса?
24. Мотивация и демотивация.
25. Антипаттерны некомпетентности.
26. Антипаттерны мнительности.
27. Последствия применения антипаттернов.
28. Корпоративная политика (наведение мостов).
29. Зачем надо планировать проекты?
30. Что надо планировать в проекте?
31. Метрики проекта.

32. Как и когда начинать планировать?
33. Сбор пользовательских требований.
34. Способы определения потенциальных пользователей.
35. Диаграммы прецедентов.
36. Разработка функциональных требований.
37. Онтологии.
38. Структурная декомпозиция работ.
39. Оценка временных затрат для работ.
40. Временное планирование на основе диаграмм Ганта.
41. Проектирование систем: процессное моделирование.
42. Нотация IDEF0.
43. Распределение ответственных за процессы.
44. Подготовка презентаций.
45. Подготовка презентаций.
46. Основы публичной защиты проектов.
47. Способы оценки качества проектов.

Вопросы для оценивания умений

1. Организация и проведение мозгового штурма по заданной тематике.
2. Позиционирование и выполнение роли в командном проекте.
3. Составление диаграммы Исикавы.
4. Составление диаграммы Ганта.
5. Формирование диаграммы прецедентов.
6. Разработка мини-онтологии в заданной предметной области.
7. Разработка процессной диаграммы.
8. Планирование публичного выступления.

Вопросы для оценивания навыков

1. Выявите основные риски на каждой фазе ЖЦ информационного проекта. Определите последствия, к которым может привести реализация рисковых событий, и предложите мероприятия для устранения рисков или минимизации их последствий.
2. Создайте план фазы разработки и фазы эксплуатации с разбивкой на задачи.

3. Компания получила заказ на разработку информационного проекта. Обоснуйте состав команды проекта и изобразите схему управления проектом.
4. Компания получила заказ на разработку информационного проекта. Была создана команда в составе 5 участников: менеджер, аналитик, разработчик, дизайнер, тестировщик. Создайте план проекта и назначьте ответственных на задачи.
5. Обоснуйте необходимый набор инструментальных средств для разработки информационного проекта и составьте план их развертывания.
6. Разработайте функциональные обязанности для аналитика ИТ проекта.
7. Компания хочет запустить проект по созданию локальной сети.
8. Разработайте требования заказчика для проекта.
9. Разработайте функциональные обязанности для разработчика ИТ-проекта.
10. Разработайте структуру команды проекта по внедрению системы взаимодействия с клиентами.

Контактная внеаудиторная работа

СРС данного вида по дисциплине «Разработка проектов в области компьютерных наук», включает следующие разделы:

– подготовку к зачёту в период лабораторно-экзаменационной сессии;

– групповые консультации с преподавателем в течение семестра;

– консультации и защиту курсовой работы;

– групповые консультация перед экзаменом;

– сдачу экзамена.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вдовин В.М. Теория систем и системный анализ: учебник. [Электронный ресурс] / В.М. Вдовин, Л.Е. Суркова, В.А. Валентинов. – Электрон. дан. – М. : Дашков и К, 2014. – 644 с. – Режим доступа: http://e.lanbook.com/book/56310 – Загл. с экрана.
2. Косяков А. Системная инженерия. Принципы и практика [Электронный ресурс]: учеб. пособие / Косяков А., Свит У. – Электрон. дан. – М. : ДМК Пресс, 2014. – 624 с. – Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=66484 – Загл. с экрана.
3. Вылегжанина А.О. Информационно-технологическое и программное обеспечение управления проектом [Электронный ресурс] : учеб. пособие / Вылегжанина А.О. – Электрон. дан. – М. ; Берлин : Директ-Медиа, 2015. – 429 с. – Режим доступа: http://www.knigafund.ru/books/184027 – Загл. с экрана.

Дополнительная учебная литература

4. Волкова В.Н. Системный анализ информационных комплексов. [Электронный ресурс] : учеб. пособие – Электрон. дан. – СПб. : Лань, 2016. – 336 с. – Режим доступа: http://e.lanbook.com/book/75506 – Загл. с экрана.
5. Дремина М.А. Проектный подход к разработке и внедрению систем менеджмента качества. [Электронный ресурс] / М.А. Дремина, В.А. Копнов, А.А. Станкин. – Электрон. дан. – СПб. : Лань, 2015. – 304 с. Режим доступа: http://e.lanbook.com/book/60653 – Загл. с экрана.

6. Гаврилова Т.А. Инженерия знаний. Модели и методы [Электронный ресурс]: учебник / Т.А. Гаврилова, Д.В. Кудрявцев, Д.И. Муромцев. – Электрон. дан. – СПб.: Лань, 2016. – 324 с. – Режим доступа: http://e.lanbook.com/book/81565 – Загл. с экрана.

Методические указания к лабораторным и практическим занятиям

7. Мохов В.А. Методические указания к самостоятельной работе, лабораторным занятиям, практическим занятиям и курсовому проектированию по курсу «Разработка проектов в области компьютерных наук» / Юж.-Рос. гос. политехн. ун-т (НПИ). – Новочеркасск : ЮРГПУ (НПИ), 2016. – 20 с. (электронный носитель).

Учебно-методическое издание

Мохов Василий Александрович

Разработка проектов в области компьютерных наук

Методические указания
к практическим занятиям, лабораторным работам,
самостоятельной работе и курсовому проектированию

Редактор *Я.В. Максименко*
Подписано в печать 21.06.2017.
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,86. Уч.-изд.л. 2,0 . Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации

Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова

А.Н. Иванченко, А.Ю. Шайда

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2016

УДК 681.3.06 (075.8)
ББК 018*32.973 Я73
И18

Рецензенты:

кандидат технических наук, доцент, доцент кафедры
«Информационные и измерительные системы и технологии»
Панфилов Александр Николаевич

кандидат технических наук, доцент, доцент кафедры
«Программное обеспечение вычислительной техники»
Кузнецова Алла Витальевна

Иванченко А.Н., Шайда А.Ю.

И18 Программирование приложений с использованием библиотек: учебное пособие / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова – Новочеркасск: ЮРГПУ (НПИ), 2016. – 85 с.

В пособии содержатся необходимые теоретические сведения по курсу «Программирование приложений с использованием библиотек», предназначенные для изучения программирования с использованием стандартной библиотеки шаблонов языка C++, с учетом последних изменений в языке, введенных стандартом C++14.

УДК 681.3.06 (075.8)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5
ТЕМА 1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ 7
1.1. Повторное использование кода 7
1.2. Библиотеки программного обеспечения 10
1.2.1. История развития понятия «библиотека» 10
1.2.2. Статические библиотеки 13
1.2.3. Динамические библиотеки 14
1.2.4. Компоновщик 15
1.3. Шаблон проектирования 18
1.4. Фреймворк 20
ТЕМА 2. СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ ЯЗЫКА С++ ... 22
2.1. Введение 22
2.2. Последовательные контейнеры 24
2.2.1. Создание, копирование и удаление последовательных контейнеров 25
2.2.2. Управление размером последовательных контейнеров 28
2.2.3. Операторы сравнения последовательных контейнеров 30
2.2.4. Операции присваивания последовательных контейнеров 31
2.2.5. Прямой доступ к элементам последовательного контейнера ... 33
2.2.6. Операции над итераторами последовательных контейнеров ... 34
2.2.7. Вставка и удаление элементов последовательных контейнеров 36
2.2.8. Специальные модифицирующие операции над списками 39
2.3. Ассоциативные контейнеры 41
2.3.1. Создание, копирование и удаление ассоциативных контейнеров 43
2.3.2. Управление размером ассоциативных контейнеров 47
2.3.3. Операторы сравнения ассоциативных контейнеров 48
2.3.4. Операции присваивания ассоциативных контейнеров 49

2.3.5. Прямой доступ к элементам ассоциативных контейнеров 50
2.3.6. Операции над итераторами ассоциативных контейнеров 51
2.3.7. Вставка и удаление элементов ассоциативных контейнеров 53
2.3.8. Специальные операции поиска..... 56
2.4. Итераторы..... 57
2.4.1. Основные понятия и определения 57
2.4.2. Вспомогательные функции для работы с итераторами 61
2.4.3. Адаптеры итераторов..... 63
2.5. Алгоритмы 66
2.5.1. Основные понятия и определения 66
2.5.2. Классификация библиотечных функций 70
2.5.3. Функции, не модифицирующие элементы контейнера..... 71
2.5.4. Модифицирующие функции 73
2.5.5. Функции разбиения..... 75
2.5.6. Функции сортировки..... 76
2.5.7. Функции двоичного поиска..... 77
2.5.8. Функции слияния отсортированных интервалов..... 78
2.5.9. Функции работы с пирамидой («кучей») 79
2.5.10. Функции поиска минимальных / максимальных элементов 80
2.5.11. Прочие функции..... 81
2.5.12. Функции для численных расчетов 82
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 84

ВВЕДЕНИЕ

Ни одна «серьезная» программа не может быть написана только с использованием «чистых» конструкций языка; программисты обязательно используют средства поддержки, которые, как правило, разрабатываются одновременно с языком, реализуются в виде библиотек поддержки и создают основу для успешной работы.

Стандартная библиотека C++ является «надстройкой» над языком и существенно расширяет его базовые возможности, предоставляя разработчикам большое количество удобных и полезных конструкций, которые могут использоваться ими в качестве «строительных блоков» для своих программ.

Физически библиотека состоит из набора заголовков (*заголовочных файлов*), в каждом из которых объявляются:

- макросы,
- значения (константы),
- типы,
- классы,
- функции и объекты, предназначенные для использования в программах.

Большая часть библиотеки реализована с помощью *шаблонов*, поэтому обычно ее реализация почти целиком располагается в заголовочных файлах.

Некоторые части библиотеки скомпилированы отдельно и должны быть скомпонованы с программой на C++. Детали этого процесса определяются реализацией.

В стандарте вся библиотека разбита на 13 категорий:

1. Language support library
2. Diagnostics library
3. General utilities library
4. Strings library
5. Localization library
6. **Containers library**
7. **Iterators library**
8. **Algorithms library**
9. Numerics library

10. Input/output library
11. Regular expressions library
12. Atomic operations library
13. Thread support library

Стандартная библиотека шаблонов (*Standard Template Library, STL*) была первоначально разработана сотрудниками Hewlett-Packard *А.Степановым* и *М.Ли* совместно с *Д.Муссером* из Ренсселэровского политехнического института (Rensselaer Polytechnic Institute). После внесения незначительных поправок Комитет по стандартизации языка C++ принял STL, сделав ее существенной составной частью стандартной библиотеки языка C++.

Использование STL дает возможность создавать более надежные, более переносимые и более универсальные программы, а также сократить расходы на их разработку. Это значит, что любой профессиональный программист должен знать эту библиотеку.

Концепция STL основана на разделении данных и операций. Данные находятся под управлением *контейнерных классов*, а операции определяются адаптируемыми *алгоритмами*. *Итераторы* выполняют функции «соединителя», связывающего эти два компонента. Благодаря им любой алгоритм может работать с любым контейнером.

ТЕМА 1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

1.1. Повторное использование кода

Повторное использование кода (**повторное использование программного обеспечения**, англ. *code reuse, software reuse*) – методология разработки программного обеспечения, заключающаяся в том, что компьютерная программа частично либо полностью должна состоять из частей, написанных ранее компонентов и/или частей другой программы, и эти компоненты должны применяться более одного раза хотя бы в разных проектах.

С самых ранних дней программирования разработчики повторно используют различные компоненты: *фрагменты кода, шаблоны, функции* и *процедуры*. Однако повторное использование программного обеспечения в качестве признанной области исследования в программной инженерии начинается с 1968 года, когда *Дуглас Макилрой* из *Bell Laboratories* предложил базировать индустрию программного обеспечения на повторно используемых компонентах.

Повторное использование кода стремится *сэкономить время и ресурсы* и *уменьшить избыточность* путем использования активов, которые уже были созданы в той или иной форме в рамках процесса разработки программного продукта. При этом повторное использование кода может означать создание отдельно поддерживаемой версии многократно используемых активов.

Помимо кода, являющегося наиболее распространенным ресурсом для повторного использования, другие активы, полученные в ходе разработки программного продукта, также могут повторно использоваться: *программные компоненты*, *наборы тестов*, *дизайн, документацию* и т.д.

К характеристикам, которые делают программный код более простым для многоразового использования, относятся: *модульность*, *слабая связанность*, *сокрытие информации и разделение задач*.

Чтобы использовать существующий код, в нем должны быть определены средства связи – *интерфейсы*, обычно включающие

в себя «вызов» или использование *подпрограммы*, *объекта*, *класса* или *прототипа*.

Общая практика использования предыдущей версии ныне живущей программы в качестве отправной точки для следующей версии, также является формой повторного использования кода.

Некоторые под «повторным использованием» понимают *простое копирование* частей или всего кода из существующей программы в новую, что впоследствии может привести к проблеме дублирования значительных объемов кода и к сложностям исправления ошибок в нем.

Многие исследователи работали, чтобы сделать повторное использование быстрее и проще, а также неотъемлемой частью нормального процесса программирования. Некоторые из основных идей были воплощены в методологии *объектно-ориентированного программирования*, которая стала одной из самых распространенных форм формализованного повторного использования. Несколько позднее повторное использование легло в основу методологии *обобщенного программирования*.

Виды повторного использования:

1. *Оппортунистический* (удобный, выгодный) - готовясь начать проект, команда понимает, что есть существующие компоненты, которые они могут повторно использовать.
2. *Планируемое* - команда стратегически проектирует компоненты, так что они будут повторно использоваться в будущих проектах.
3. *Внутреннее* - команда повторно использует свои собственные компоненты.
4. *Внешнее* - команда может выбрать лицензию компонент стороннего производителя. Команда должна учитывать стоимость лицензии и время, которое требуется, чтобы найти, изучить и интегрировать компонент.
5. *Реферировано* - клиентский код содержит ссылку на повторное использование кода, и, таким образом, они имеют различные жизненные циклы и могут иметь различные версии.

6. *Раздвоенный* - клиентский код содержит локальную или личную копию пересмотренного кода, и, таким образом, что они разделяют одну и жизненный цикл одной версии.

Достоинства повторного использования:

1. Разработчик новой системы снимает с себя заботу о реализации функциональности, заложенной в используемых компонентах. Весь цикл разработки и поддержки компонент осуществляется разработчиком данного компонента: устранение ошибок, развитие и улучшение работы, тестирование.
2. Повторения использования кода приводит к существенному уменьшению размера итоговой программы, а при недостаточной производительности носителя и к быстродействию.

Недостатки повторного использования:

1. Подключение к проекту сторонних компонент автоматически приводит к необходимости контроля совместимости версий компонент создаваемой системы и версий используемых компонент.
2. Многие компоненты коммерческие и требуют денежных затрат.
3. Часто компоненты недостаточно универсальны и не реализуют той функциональности, которая требуется создаваемой системе, либо, наоборот, слишком универсальны и в результате неэффективны, неудобны или содержат много избыточной (для данного проекта) функциональности.
4. Некоторые лицензии распространяемых компонент позволяют использовать её исходные коды и модифицировать их в соответствии с необходимостью. Но после этого ответственность за поддержку функциональности компоненты перекладывается на плечи разработчика новой системы.
5. Использование излишней модульности может привести к уменьшению скорости выполнения программы, когда скорость выполнения, заложенная в

модуль, не может перекрыть издержки на обращение к этому модулю.

Основными примерами повторного использования кода являются:

1. *Библиотеки программного обеспечения* (англ. *software libraries*).
2. *Шаблоны проектирования* (англ. *design pattern*).
3. *Каркасы* (англ. *frameworks*).

1.2. Библиотеки программного обеспечения

1.2.1. История развития понятия «библиотека»

Одно из самых первых упоминаний в литературе о библиотеках подпрограмм содержится в книге *Уилкса М., Уилера Д. и Гилла С.*, которая в русском переводе вышла в 1953 году [1]. В качестве одной из форм организации вычислений на ЭВМ авторами предлагалось использование библиотеки подпрограмм, и уже тогда была показана эффективность такого метода программирования. Хотя авторы не приводили строгого определения библиотеки подпрограмм, из изложения было ясно, что под **библиотекой** понимается набор подпрограмм, которые представляют собой «*короткие, заранее заготовленные программы для отдельных, часто встречающихся (стандартных) вычислительных операций*». Здесь же предлагалась форма хранения и способ использования подпрограмм, а также впервые вводились представления об открытой и закрытой подпрограммах, параметризации подпрограмм, рассматривались вопросы настройки программ по адресам в памяти. Был также поднят вопрос о создании объединяющей программы, которая взяла бы на себя выполнение носящих механический характер работ по объединению подпрограмм в более сложные программы.

Последующий прогресс вычислительной техники привел к включению в состав ЭВМ внешней памяти на магнитных носителях, отличавшейся значительно большей емкостью, чем оперативная память, что естественным образом привело к идее хранения библиотечных подпрограмм на внешней памяти и

возможности эффективной автоматизации процесса включения их в программы.

В работах того периода приводятся не достаточно четкие формулировки понятия библиотеки подпрограмм.

Можно выделить следующее определение **библиотеки подпрограмм**, данное Е.А.Жоголевым и Н.П.Трифоновым [2]: «*для большего удобства практической работы выбирается определенная система использования подпрограмм (способ обращения к подпрограммам, способ их размещения в памяти машины и т.д.), предъявляющая определенные требования к подпрограммам как в отношении их структуры, так и оформления. Подпрограммы, удовлетворяющие всем требованиям выбранной системы, называются стандартными. Имеющийся набор стандартных подпрограмм называют библиотекой подпрограмм*».

Следует отметить, что для рассматриваемых определений библиотек того периода было характерно следующее: эти определения не включали в себя ограничений на вид материального носителя, на котором представлены библиотечные подпрограммы. Т.е. под библиотекой мог пониматься и набор подпрограмм, представленных в виде последовательностей команд, так и записанных на бумаге.

Второй характерной чертой используемых прежде определений библиотек было употребление термина «*стандартная подпрограмма*». Термин «стандартный» в полной мере соответствовал изначальному смыслу слова «стандарт», т.е. обязательный, единственно правильный (для разработчиков подпрограмм, предназначенных для общего использования).

Сейчас все чаще пользуются термином «*библиотека подпрограмм*», опуская определение «стандартных».

С введением в употребление ряда языков программирования высокого уровня резко возросло количество специалистов, занимающихся созданием библиотечных подпрограмм, и возросло количество самых разных библиотек, находящихся как, в общем, так и в локальном использовании. Появилось представление о библиотеках исходных (на языке программирования), объектных и загрузочных модулей, о библиотеках общих и личных. Подпрограммы из разных

библиотек должны были отвечать своим, отличным от других, требованиям, как в рамках разных систем программирования, так и в рамках одной и той же системе программирования.

Термин «стандартный» продолжает употребляться в отношении широко распространенных библиотек (например, Стандартная библиотека языка C++).

Некоторые более поздние определения библиотек подпрограмм, встречающиеся в литераторе, включали в определение библиотеки подпрограмм не только саму совокупность подпрограмм, но и системы их использования и обслуживания, т.е. программные средства, выполняющие роль программ-библиотекарей (автоматизирующих включение, исключение и замену подпрограмм в библиотеке), *загрузчиков* и *редакторов связей*. Однако такое расширение понятия библиотеки представляется нежелательным, так как упомянутые программные средства, входящие в современные системы программирования, являются общими для многих разных библиотек, т.е. применяются при формировании на внешней памяти, поддержании и использовании любых библиотек, созданных в рамках конкретной системы программирования.

Библиотека (от англ. *library*) - это определенным образом организованный набор подпрограмм, представленный на внешней памяти вычислительной системы в соответствии с требованиями системы программирования, которые определяют возможность автоматического включения подпрограмм в процессы обработки данных.

В некоторых языках программирования (например, в *Python*) библиотека то же, что и модуль, в некоторых - несколько модулей. Большинство языков программирования имеют *стандартную библиотеку*, хотя программисты могут также создавать свои собственные *пользовательские библиотеки*.

Библиотеки не существуют изолированно, а подвергаются воздействию вызывающей программой или другой библиотекой, а также сами воздействуют на них. Библиотека представляет собой коллекцию реализаций *поведения*, написанных с учетом особенностей языка, имеющую четко определенный *интерфейс*, с помощью которого вызывается поведение. Для использования кода в библиотеке пользователь должен знать только интерфейс,

а не внутренние детали библиотеки. Например, в процедурном языке С, поведение в библиотеке вызывается с помощью обычного вызова функции языка.

Поведение, реализуемое с помощью библиотеки, может быть подключено к вызывающей программе на различных этапах ее жизненного цикла. С точки зрения операционной системы и прикладного программного обеспечения, библиотеки подразделяются на *статические* и *динамические*.

1.2.2. Статические библиотеки

Статическая библиотека (от англ. *static library*) или **статически подключаемая библиотека** представляет собой набор подпрограмм, внешних функций и переменных, которые размещаются в вызывающей программе во время компиляции и копируются в целевое приложение с помощью *компилятора* и *компоновщика*, создавая *объектный файл* и автономный исполняемый файл.

Статическая библиотека – файл с исходным кодом или *объектный файл*, предназначенный для вставки в программу на этапе компоновки.

Объектный модуль (также – *объектный файл*, англ. *object file*) – файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором.

Библиотеки, распространяемые в виде исходного кода, преобразуются компилятором в объектные файлы. Затем *компоновщик* соединяет объектные файлы библиотек и объектные файлы вашей программы в один исполняемый файл.

Например, в исходных текстах распространяются:

- библиотеки для языка Fortran;
- библиотека Boost для языка C++.

Библиотеки, распространяемые в виде объектных файлов, уже готовы к компоновке. Компоновщик выполняет соединение объектных файлов библиотек и объектных файлов вашей программы во время создания исполняемого файла.

Расширения объектных файлов статических библиотек в разных ОС, приведены в табл. 1.1.

Таблица 1.1

Расширение	**ОС**
lib	Microsoft Windows
a	UNIX

Стандартные библиотеки многих компилируемых языков программирования (Fortran, Pascal, C, C++ и других) распространяются в виде объектных файлов.

Достоинства:

- все необходимые функции включаются в один исполняемый файл.
- Недостатки:
- исполняемый файл занимает больше места на диске и в памяти;
- при обнаружении ошибок в библиотеке требуется повторная сборка всех программ.

1.2.3. Динамические библиотеки

Динамическая библиотека (от англ. *dynamic library*) – файл, содержащий машинный код. Загружается в память процесса *загрузчиком* программ операционной системы либо при создании процесса, либо по запросу уже работающего процесса, то есть динамически.

Расширения файлов динамических библиотек в разных ОС, приведены в табл. 1.2.

Таблица 1.2

Расширение	**ОС**	**Расшифровка**
dll	Microsoft Windows, OS/2	англ. ***d****ynamic* ***l****ink* ***l****ibrary*
so	UNIX	англ. ***s****hared* ***o****bject*
dylib	Mac OS	англ. ***d****y****lib****rary*

В зависимости от назначения различают:

- библиотеки, используемые одной программой и содержащие критические для работы программы

функции. Недостаток: при отсутствии библиотеки программа не сможет работать;
- библиотеки, используемые одной программой и содержащие дополнительные функции. Например, библиотеки плагинов используются для расширения функциональности программы;
- библиотеки общего пользования (англ. shared library). Содержат функции, используемые несколькими программами. Могут загружаться в адресное пространство ОС (англ. system library) для экономии памяти: одна копия библиотеки будет использоваться несколькими процессами.

При написании программы программисту достаточно указать *транслятору* (*компилятору* или *интерпретатору*) путь к библиотеке и имя функции. Ни исходный текст функции, ни её исполняемый код в состав программы не войдут.

Достоинства:

- экономия памяти за счёт использования одной библиотеки несколькими процессами;
- возможность исправления ошибок (достаточно заменить файл библиотеки и перезапустить работающие программы).

Недостатки:

- возможность нарушения API (при внесении изменений в библиотеку существующие программы могут перестать работать);
- конфликт версий динамических библиотек (разные программы могут ожидать разные версии библиотек);
- доступность одинаковых функций по одинаковым адресам в разных процессах (упрощает эксплуатацию уязвимостей)

1.2.4. Компоновщик

Компоновщик (*редактор связей*, *ли́нкер*, от англ. *link editor*, *linker*) – это инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или

несколько объектных модулей и собирает по ним исполнимый модуль (рис.1.1).

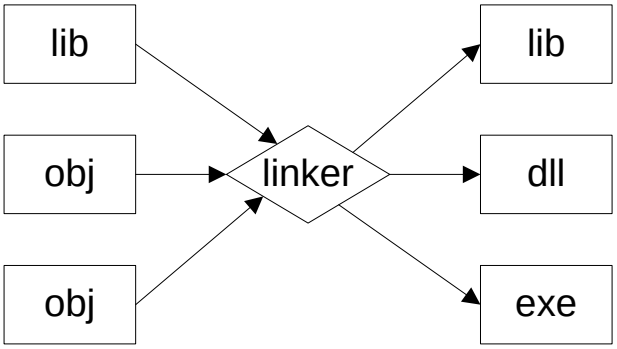


Рис.1.1. Компоновка

Изначально, до появления динамических библиотек, загрузчики могли выполнять некоторые функции компоновщика, однако сейчас, чаще всего, загрузка программ выделяется в отдельный процесс.

Для связывания модулей компоновщик использует таблицы символов, созданные компилятором в каждом из объектных модулей. Эти таблицы могут содержать символы следующих типов:

- определённые или экспортируемые имена – функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям;
- неопределённые или импортируемые имена – функции и переменные, на которые ссылается модуль, но не определяет их внутри себя;
- локальные – могут использоваться внутри объектного файла для упрощения процесса настройки адресов.

Для большинства компиляторов, один объектный файл является результатом компиляции одного файла с исходным кодом. Если программа собирается из нескольких объектных файлов, компоновщик собирает эти файлы в единый исполнимый модуль, вычисляя и подставляя адреса вместо символов, в течение времени компоновки (статическая компоновка) или во время исполнения (динамическая компоновка).

Компоновщик может извлекать объектные файлы из специальных коллекций, называемых библиотеками. Если не все

символы, на которые ссылаются пользовательские объектные файлы, определены, то компоновщик ищет их определения в библиотеках, которые пользователь подал ему на вход. Обычно, одна или несколько системных библиотек используются компоновщиком по умолчанию. Когда объектный файл, в котором содержится определение какого-либо искомого символа, найден, компоновщик может включить его (файл) в исполнимый модуль (в случае статической компоновки) или отложить это до момента запуска программы (в случае динамической компоновки).

Работа компоновщика заключается в том, чтобы в каждом модуле определить и связать ссылки на неопределённые имена. Для каждого импортируемого имени находится его определение в других модулях, упоминание имени заменяется на его адрес.

Компоновщик обычно не выполняет проверку типов и количества параметров процедур и функций. Если надо объединить объектные модули программ, написанные на языках со строгой типизацией, то необходимые проверки должны быть выполнены дополнительной утилитой перед запуском редактора связей.

Динамический компоновщик (от англ. *dynamic linker*) является частью операционной системы, которая загружает и связывает необходимые для исполняемого файла общие библиотеки во время выполнения, путем копирования содержимого библиотек из постоянного хранилища в оперативную память, и заполнение таблиц переходов и перемещения указателей.

Компоновщик часто упоминается как процесс, который выполняется, когда исполняемый файл скомпилирован, в то время как динамический компоновщик является специальной частью операционной системы, которая загружает внешние совместно используемые библиотеки в запущенном процессе, а затем привязывает эти общие библиотеки динамически к выполняемому процессу. Этот подход также называется динамическое связывание (от англ. *dynamic linking*) или позднее связывание (от англ. *late linking*).

1.3. Шаблон проектирования

Шаблон проектирования или **паттерн** (англ. *design pattern*) в разработке программного обеспечения – повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются *идиомами*. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

Идиома программирования – устойчивый способ выражения некоторой составной конструкции в одном или нескольких языках программирования. Идиома является шаблоном решения задачи, записи алгоритма или структуры данных путём комбинирования встроенных элементов языка.

На наивысшем уровне существуют *архитектурные шаблоны*, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Наиболее известным научным трудом в данной области является книга Э. Гамма, Р. Хелм, Р. Джонсон Дж., Влиссидес «*Приемы объектно-ориентированного проектирования. Паттерны проектирования*» [3]. В этой книге описаны 23 шаблона проектирования. Также команда авторов этой книги известна общественности под названием «Банда четырёх» (англ. *Gang of Four*, часто сокращается до *GoF*).

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта,— снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Многие шаблоны проектирования в объектно-ориентированном проектировании можно рассматривать как идиоматическое воспроизведение элементов функциональных языков. Питер Норвиг утверждает, что 16 из 23 шаблонов, описанных в книге «Банды Четверых», в Lisp или Dylan реализуются существенно проще, чем в C++, либо оказываются незаметны. Пол Грэхэм считает саму идею шаблонов проектирования — *антипаттерном*, сигналом о том, что система не обладает достаточным уровнем абстракции, и необходима её тщательная переработка. Нетрудно видеть, что само определение шаблона как «готового решения, но не прямого обращения к библиотеке» по сути означает отказ от повторного использования в пользу *дублирования*.

1.4. Фреймворк

Фре́ймворк, каркас, структура (англ. *framework*) – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

В отличие от библиотеки фреймворк диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию – «каркас», который нужно будет расширять и изменять, согласно указанным требованиям. Также фреймворк может содержать в себе большое число разных по тематике библиотек.

Другим ключевым отличием фреймворка от библиотеки может быть инверсия управления: пользовательский код вызывает функции библиотеки (или классы) и получает управление после вызова. Во фреймворке, пользовательский код может реализовывать конкретное поведение, встраиваемое в более общий – абстрактный код фреймворка. При этом, фреймворк вызывает функции (классы) пользовательского кода.

Одно из главных преимуществ при использовании «каркасных» приложений – «стандартность» структуры приложения. «Каркасы» стали популярны с появлением *графических интерфейсов пользователя*, которые имели тенденцию к реализации стандартной структуры для приложений. С их использованием стало гораздо проще создавать средства для автоматического создания графических интерфейсов, так как структура внутренней реализации кода приложения стала известна заранее. Для обеспечения каркаса, обычно используются техники объектно-ориентированного программирования (например, части приложения могут наследоваться от базовых классов фреймворка).

Одним из первых коммерческих фреймворков приложения был MacApp, написанный Apple для Macintosh. Первоначально созданный с помощью расширенной (объектно-ориентированной) версии языка Object Pascal, впоследствии он был переписан на C++.

Microsoft создала похожий продукт для Windows, который называется Microsoft Foundation Classes (MFC). На данный момент основным продуктом Microsoft для разработки ПО предлагается .NET Framework.

Фреймворк определяется как множество конкретных и абстрактных классов, а также определений способов их взаимоотношения. Конкретные классы обычно реализуют взаимные отношения между классами. Абстрактные классы представляют собой точки расширения, в которых каркасы могут быть использованы или адаптированы.

Точка расширения – это та «часть» фреймворка, для которой не приведена реализация. Соответственно, каркас концептуальной модели состоит из концептуальных классов, а каркас программной системы – из классов языка программирования общего назначения.

Процесс создания фреймворка заключается в выборе подмножества задач проблемы и их реализаций. В ходе реализаций, общие средства решения задач заключаются в конкретных классах, а изменяемые средства – выносятся в точки расширения.

ТЕМА 2. СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ ЯЗЫКА С++

2.1. Введение

Стандартная библиотека шаблонов (англ. *standard template library – STL*) является ядром стандартной библиотеки языка С++ – это библиотека универсальных компонентов для управления коллекциями данных с помощью современных и эффективных алгоритмов.

STL была первоначально разработана сотрудниками Hewlett-Packard *А.Степановым* и *М.Ли* совместно с *Д.Муссером* из Ренсселэровского политехнического института. В ее основе лежит взаимодействие разных хорошо структурированных компонентов, главными среди которых являются:

1. **Контейнеры** – используются для управления коллекциями объектов определенного типа. Каждый вид контейнеров имеет свои достоинства и недостатки, поэтому наличие разных контейнеров отражает разные требования к коллекциям, существующие в программах. Контейнеры делятся на четыре основные категории, которые подразделяются на типы:
   - последовательные контейнеры (5 типов);
   - ассоциативные контейнеры (2 типа);
   - неупорядоченные ассоциативные контейнеры (2 типа);
   - контейнерные адаптеры (2 типа).
2. **Итераторы** – используются для обхода элементов в коллекциях объектов. Этими коллекциями могут быть контейнеры или подмножества контейнеров. Основное преимущество итераторов заключается в том, что они предоставляют небольшой и в тоже время универсальный интерфейс для произвольного типа контейнера. Например, одна из операций этого интерфейса перемещает итератор на следующий элемент в коллекции. Выполнение этой операции не зависит от внутренней структуры коллекции. Чем бы

ни была коллекция – массивом, деревом или хеш-таблицей, – эта операция работает одинаково, поскольку каждый контейнер определяет свой собственный тип итератора, который просто «правильно работает», поскольку знает внутреннюю структуру своего контейнера.

3. **Алгоритмы** – предназначены для обработки элементов коллекций. Например, алгоритмы могут искать, сортировать, модифицировать и просто использовать элементы для разных целей. Алгоритмы используют итераторы. Таким образом, поскольку интерфейс итераторов является общим для всех типов контейнеров, алгоритм достаточно написать один раз, и он будет работать с любым контейнером

Концепция STL основана на разделении данных и операций. Данные управляются контейнерными классами, а операции определяются настраиваемыми алгоритмами. Связующим звеном между этими двумя компонентами являются итераторы. Они позволяют алгоритму взаимодействовать с любым контейнером (рис. 2.1).

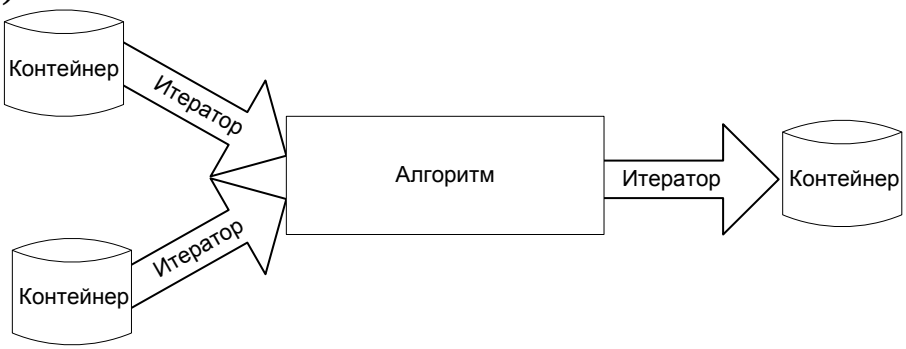


Рис. 2.1. Компоненты STL

В некотором смысле концепция STL противоречит исходной идее объектно-ориентированного программирования: STL отделяет данные от алгоритмов, а не объединяет их. Однако программист может объединять любой контейнер с любым алгоритмом, так что результат будет очень гибким и в то же время компактным.

Одной из основных особенностей STL является то, что все компоненты работают с произвольными типами. Как следует из названия «стандартная библиотека шаблонов», все компоненты

являются *шаблонами*, допускающими использование любого типа, при условии, что этот тип способен выполнять требуемые операции. Таким образом, *STL* — яркий пример концепции *обобщенного программирования* (*generic programming*). Контейнеры и алгоритмы являются обобщенными по отношению к произвольным типам и классам соответственно.

2.2. Последовательные контейнеры

Последовательные контейнеры (англ. *sequence containers*) – это упорядоченные коллекции, в которых каждый элемент занимает определенную позицию. Позиция зависит от времени и места вставки, но не связана со значением элемента.

Поддержка работы с последовательными контейнерами представлена в заголовочных файлах: *vector*, *deque*, *list*, *forward_list* и *array*, в каждом из которых содержится определение соответствующего шаблонного класса:

1. **Вектор (динамический массив)** – последовательный контейнер, который поддерживает итераторы произвольного доступа. Кроме того, *vector* поддерживает операции вставки и удаления в конце последовательности с постоянным временем; вставка и удаление в середине занимают линейное время.
2. **Двунаправленная очередь (дек)** – вид последовательности, которая, подобно вектору, поддерживает итераторы произвольного доступа. Кроме того она поддерживает операции вставки и удаления в начале или в конце последовательности за постоянное время; вставка и удаление в середине занимают линейное время.
3. **Двунаправленный (двусвязный) список** – вид последовательности, которая поддерживает двунаправленные итераторы и допускает операции вставки и удаления с постоянным временем в любом месте последовательности. В отличие от векторов и двусторонних очередей, быстрый произвольный доступ к элементам списка не поддерживается, но в

большинстве случаев бывает нужен только последовательный доступ.

4. **Однонаправленный (односвязный) список** – вид последовательности, которая поддерживает однонаправленные итераторы. Представляет собой ограниченный список, в котором не поддерживаются операции, предусматривающие перемещение назад или вызывающие снижение быстродействия.
5. **Массив фиксированного размера** – это последовательный контейнер фиксированного размера с возможностью прямого доступа к элементам. У него нет дополнительного расхода памяти для хранения элементов, он может инициализироваться с помощью списка инициализации, знает свой размер (количество элементов), и не может быть неявным образом преобразован к указателю. Этот контейнер очень похож на стандартный массив, но без его недостатков.

а также определение внешней шаблонной функции *swap* и перегруженных шаблонных операций: ==, !=, <, <=, >, >=.

Кроме того, заголовочный файл *vector*, содержит дополнительно определение специализации *vector<bool>*.

2.2.1. Создание, копирование и удаление последовательных контейнеров

Все последовательные контейнеры имеют несколько перегруженных конструкторов и могут создаваться разными способами. Для уничтожения всех элементов контейнера и высвобождения памяти используется деструктор (табл. 2.1).

Таблица 2.1

Синтаксис	**Описание**
1	2
vector <Elem> v *deque <Elem> d* *list <Elem> l* *forward_list <Elem> fl* *array <Elem, N> a*	создание пустого контейнера с помощью конструктора по умолчанию

1	2
vector <Elem> v(n) *deque <Elem> d(n)* *list <Elem> l(n)* *forward_list <Elem> fl(n)*	создание пустого контейнера заданного размера
vector <Elem> v(n, elem) *deque <Elem> d(n, elem)* *list <Elem> l(n, elem)* *forward_list <Elem> fl(n, elem)*	создание контейнера заданного размера, заполненного постоянным значением *elem*
vector <Elem> v(beg, end) *deque <Elem> d(beg, end)* *list <Elem> l(beg, end)* *forward_list <Elem> fl(beg, end)*	создание контейнера, заполненного диапазоном значений из интервала *[beg, end)*
vector <Elem> v(v2) *deque <Elem> d(d2)* *list <Elem> l(l2)* *forward_list <Elem> fl(fl2)* *array <Elem, N> a(a2)*	конструктор копирования (перемещения)
vector <Elem> v = v2 *deque <Elem> d = d2* *list <Elem> l = l2* *forward_list <Elem> fl= fl2* *array <Elem, N> a = a2*	
vector <Elem> v(initlist) *deque <Elem> d(initlist)* *list <Elem> l(initlist)* *forward_list <Elem> fl(initlist)*	создание контейнера с помощью списка инициализации
vector <Elem> v = initlist *deque <Elem> d = initlist* *list <Elem> l = initlist* *forward_list <Elem> fl= initlist* *array <Elem, N> a = initlist*	
v.~vector() *d.~deque()* *l.~list()* *fl.~forward_list()* *a.~array()*	вызов деструктора контейнера

Примечание: инициализация объектов класса *array* имеет уникальную семантику – элементы контейнера *array*

инициализируются по умолчанию, даже если для их инициализации ничего не передается.

array<int,4> x – инициализирует элементы массива неопределенными значениями.

Для корректного создания объектов класса *array* необходимо использовать *список инициализации.*

array<int,4> x={} – инициализирует элементы массива нулевыми значениями (*{} – пустой инициализатор*).

array<int,5> x={22, 35, 158, 9, 256} – инициализирует элементы массива конкретными значениями при его создании.

array<int,5> x={22} – инициализирует первый элемент массива значением 22, а остальные 4 значением 0.

Листинг 1. Конструирование контейнеров различных типов:

```
using namespace std;
template<class T>
void Print(string s, T v){
     cout << s;
     for (auto i : v) cout << " " << i;
     cout << endl;
}
int main() {
     vector <int> v0; //конструктор по умолчанию
     Print("v0 = ", v0); //вектор пуст v0 =
     array<int, 3> a0; //конструктор по умолчанию
     Print("a0 = ", a0); // элементы массива получают
неопределенные значения
     array<int, 3> a1 = {}; //пустой инициализатор
     Print("a1 = ", a1); //элементы массива
инициализируются 0
     deque<int> d1(5, 2); //создание дека заданного
размера
     Print("d1 = ", d1); // d1 = 2 2 2 2 2
     list<int> l1 = { 25, 33, 12, 49, 7 };//использование
списка инициализации
     Print("l1 = ", l1);
     forward_list <int> f1(l1.rbegin(), l1.rend());
//создание списка,
     //заполненного диапазоном значений
     Print("f1 = ", f1); //f1 = 7 49 12 33 25
     v0.push_back(10); v0.push_back(20);
     v0.push_back(30);
     vector <int> v2(v0); //конструктор копирования
     Print("v2 = ", v2); // v2 = 10 20 30
     vector <int> v3(move(v2)); //конструктор перемещения
     Print("v3 = ", v3); // v3 = 10 20 30
```

```
        Print("v2 = ", v2); // v2  =
        return 0;
    }
```

2.2.2. Управление размером последовательных контейнеров

Последовательные контейнеры имеют следующие функции для работы с размером (табл. 2.2).

Таблица 2.2

Функции	**Описание**	**вектор**	**дек**	**список**	**послед. список**	**массив**
max_size()	возвращает максимально возможное количество элементов	+	+	+	+	+
size()	возвращает текущее количество элементов	+	+	+	-	+
capacity()	возвращает количество элементов, максимально возможное без повторного выделения памяти	+	-	-	-	-
resize(num)	изменяет количество элементов до *num* (если размер *size ()* увеличивается, новые элементы создаются их конструкторами по умолчанию)	+	+	+	+	-
reserve(num)	резервирует память, гарантируя определенную емкость еще до того, как она действительно потребуется	+	-	-	-	-
shrink_to_fit()	уменьшает емкость контейнер до текущего количества элементов	+	+	-	-	-
empty()	возвращает признак того, что контейнер пуст (эквивалент *size() ==0*)	+	+	+	+	+

Примечание: контейнер *vector* реализован на базе обычного массива со специальной стратегией упреждающего выделения дополнительной памяти для возможного роста массива. В связи с этим добавление нового элемента в конец массива с помощью функции *push_back* почти всегда выполняется очень быстро. Также быстро происходит доступ к произвольному элементу вектора с помощью индекса.

Медленными операциями являются добавление нового элемента в конец массива в случае, когда *size() = capacity()*, а также вставка нового элемента в произвольное место вектора (выполняется физическое перемещение части элементов массива) (рис. 2.2).

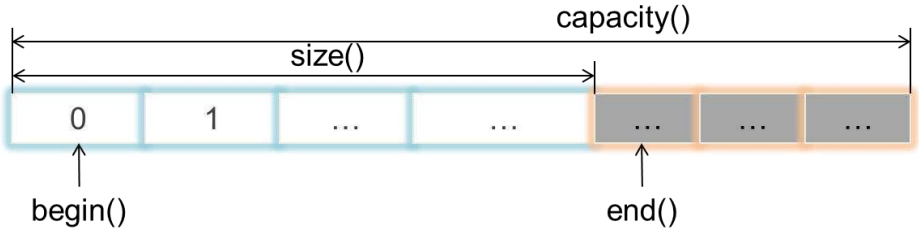


Рис. 2.2. Внутреннее устройство вектора

Листинг 2. Управление размером последовательных контейнеров:

```
int main() {
     deque <int> d1;
     cout << "d1: max_size = " << d1.max_size()
          << ",  size =  " << d1.size() << ", empty = " <<
d1.empty() << endl;
     d1.resize(10); //увеличивает кол-во элементов до 10,
инициализируя 0
     cout << "d1: size =  " << d1.size() << ", empty = "
<< d1.empty() << endl;
     d1.resize(2); //уменьшает кол-во элементов до 2
     cout << "d1: size =  " << d1.size() << endl;
     Print("d1 = ", d1); // d1 = 0 0
     list <string> l1;
     l1.push_front("one"); l1.push_front("two");
l1.push_front("three");
     cout << "l1: max_size = " << l1.max_size()
          << ", size = " << l1.size() << ", empty = " <<
l1.empty() << endl;
     Print("l1 = ", l1); // l1 = three two one
     forward_list <double> f1;
     f1.emplace_front(1.2); f1.emplace_front(4.5);
     cout<<"f1: max_size = "<<f1.max_size() << ", empty =
" << f1.empty() << endl;
     Print("f1 = ", f1); // f1 = 4.5  1.2
     vector <int> v1({ 1, 2, 3, 4, 5, 6, 7 });
     cout << "v1: max_size = " << v1.max_size()
          <<", size = "<< v1.size() << ", capacity = " <<
v1.capacity() << endl;
// size = 7 capacity = 7
     v1.push_back(8);
     cout << "v1: size = " << v1.size() << ", capacity = "
<<v1.capacity()<< endl;
```

```
// size = 8 capacity = 14
     for (int i = 9; i<15; i++)
          v1.push_back(i);
     cout << "v1: size = " << v1.size() << ", capacity =
"<< v1.capacity() <<endl;
// size = 14 capacity = 14
     v1.push_back(15);
     cout << "v1: size = " << v1.size() << ", capacity = "
<< v1.capacity()<<endl;
// size = 15 capacity = 28
     v1.reserve(250);
     cout << "v1: size = " << v1.size() << ", capacity = "
<< v1.capacity()<<endl;
// size = 15 capacity = 250
     v1.shrink_to_fit();
     cout << "v1: size = " << v1.size() << ", capacity = "
<< v1.capacity()<<endl;
// size = 15 capacity = 15
     Print("v1 = ", v1);
     return 0;
}
```

2.2.3. Операторы сравнения последовательных контейнеров

Для всех типов последовательных контейнеров определены операторы сравнения, подчиняющиеся трем правилам (табл. 2.3):

1. Оба контейнера должны быть однотипными.
2. Два контейнера считаются равными, если их элементы равны и располагаются в одинаковом порядке.
3. Для проверки того, какой из двух контейнеров меньше, используется *лексикографическое сравнение***.**

Таблица 2.3

Операторы сравнения	**Действие**
c1 == c2	возвращает *true*, если контейнер *c1* равен контейнеру *c2*
c1!= c2	возвращает *true*, если контейнер *c1* не равен контейнеру *c2* (эквивалент *! (c1==c2)*)
c1 < c2	возвращает *true*, если контейнер *c1* меньше контейнера *c2*
c1 > c2	возвращает *true*, если контейнер *c1* больше контейнера *c2*
c1 <= c2	возвращает *true*, если контейнер *c1* меньше или равен контейнеру *c2*
c1 >= c2	возвращает *true*, если контейнер *c1* больше или равен контейнеру *c2*

Листинг 3. Использование операторов сравнения на примере векторов:

```
int main() {
     vector <int> v1 = { 10, 20, 30 };
     vector <int> v2;
     Print("v1 = ", v1); Print("v2 = ", v2);
     cout << "v1==v2: " << (v1 == v2) << endl << "v1!=v2:
" << (v1 != v2) << endl
          << "v1<v2: " << (v1<v2) << endl << "v1>v2: " <<
(v1>v2) << endl
          << "v1<=v2: " << (v1 <= v2) << endl << "v1>=v2:
" << (v1 >= v2) <<endl;
     v2.resize(v1.size());
     copy(v1.begin(), v1.end(), v2.begin());
     Print("v2 = ", v2);
     cout << "v1==v2: " << (v1 == v2) << endl << "v1!=v2:
" << (v1 != v2) << endl
          << "v1<v2: " << (v1<v2) << endl << "v1>v2: " <<
(v1>v2) << endl
          << "v1<=v2: " << (v1 <= v2) << endl << "v1>=v2:
" << (v1 >= v2) <<endl;
     copy(v1.rbegin(), v1.rend(), v2.begin());
     Print("v2 = ", v2);
     cout << "v1==v2: " << (v1 == v2) << endl << "v1!=v2:
" << (v1 != v2) << endl
          << "v1<v2: " << (v1<v2) << endl << "v1>v2: " <<
(v1>v2) << endl
          << "v1<=v2: " << (v1 <= v2) << endl << "v1>=v2:
" << (v1 >= v2) <<endl;
     return 0;
}
```

2.2.4. Операции присваивания последовательных контейнеров

Во время присваивания контейнеров все элементы контейнера-источника копируются в контейнер-приемник, а старые элементы контейнера-приемника удаляются из него (табл. 2.4). Таким образом, присваивание контейнеров является относительно затратным. Для увеличения быстродействия можно использовать семантику перемещающего присваивания, при которой просто переставляются указатели в памяти, а не копируются все значения.

Таблица 2.4

Функции	**Описание**	**вектор**	**дек**	**список**	**послед. список**	**массив**
c = c2	присваивает все элементы контейнера *c2* контейнеру *c*	+	+	+	+	+
c = rv	перемещает все элементы контейнера *rv* в контейнер *c*	+	+	+	+	+
c. fill (val)	присваивает значение *val* каждому элементу контейнера *c*	-	-	-	-	+
c1.swap(c2) *swap (c1, c2)*	обменивает элементы контейнеров *c1* и *c2*	+	+	+	+	+
c= initlist	присваивает контейнеру *c* все элементы списка инициализации *initlist*	+	+	+	+	+
c.assign (n,elem)	присваивает *n* копий элемента *elem*	+	+	+	+	-
c. assign (beg, end)	присваивает элементы из интервала *[beg, end)*	+	+	+	+	-
c.assign (initlist)	присваивает контейнеру *c* все элементы из списка инициализации *initlist*	+	+	+	+	-

Листинг 4. Использование операций присваивания для последовательных контейнеров:

```
int main() {
     vector <int> v1 = { 1, 2, 3, 4, 5, 6, 7 }, v2;
     deque<int> d1{ 10, 11, 12 }, d2;
     list<string> l1;
     forward_list<double> f1;
     array<int, 3> a1 = {}, a2 = {};
     v2 = v1;
Print("v1 = ", v1); Print("v2 = ", v2); // v1 = v2 = 1 2 3
4 5 6 7
     d2 = move(d1);
Print("d1 = ", d1); Print("d2 = ", d2); // d1 =; d2 = 10
11 12
     l1 = { "1", "2", "3" };
          Print("l1 = ", l1); // l1 = 1 2 3
     f1 = { 1.2, 4.5, 0.05 };
          Print("f1 = ", f1); // f1 = 1.2 4.5 0.05
     a1 = { 1, 2, 4 };
          Print("a1 = ", a1); //a1 = 1 2 4
     a2.fill(5);
```

```
        Print("a2 = ", a2); //a2 = 5 5 5
    a2.swap(a1);
        Print("a1 = ", a1); //a1 = 5 5 5
        Print("a2 = ", a2); //a2 = 1 2 4
    d2.assign(v1.rbegin(), v1.rbegin() + 3);
        Print("d2 = ", d2); // d2 = 7 6 5
    return 0;
}
```

2.2.5. Прямой доступ к элементам последовательного контейнера

В табл. 2.5 перечислены операции над последовательными контейнерами, обеспечивающие прямой доступ к его элементам.

Таблица 2.5

Функции	**Описание**	**вектор**	**дек**	**список**	**послед. список**	**массив**
c[i]	возвращает элемент с индексом *i* (проверка выхода за пределы допустимого диапазона не выполняется)	+	+	-	-	+
c.at(i)	возвращает элемент с индексом *i* (при выходе за пределы допустимого диапазона генерирует исключение)	+	+	-	-	+
c.front	возвращает первый элемент (проверка существования первого элемента не выполняется)	+	+	+	+	+
c.back	возвращает последний элемент (проверка существования последнего элемента не выполняется)	+	+	+	-	+

Примечание: первый элемент контейнера имеет индекс *0*, а последний – *size()-1*, т.о. *n*-й элемент имеет индекс *n-1*.

Листинг 5. Использование операторов прямого доступа:

```
int main() {
    vector <int> v1({ 1, 2, 3, 4, 5, 6, 7 });
    deque <int> d1 = { 10, 11, 12 };
    list <string> l1({ "one", "two", "three" });
    forward_list <double> f1 = { 1.2, 4.5, 0.05 };
    array<string, 3>a1 = { "run", "this", "program" };
    Print("v1 = ", v1);
    v1[3] = 99; v1.at(4) = 100; Print("v1 = ", v1); // v1
= 1 2 3 99 100 6 7
```

```
     Print("d1 = ", d1);
     d1[0] = d1[2]; d1.at(1) = 55; Print("d1 = ", d1); //
d1 = 12 55 12
     Print("l1 = ", l1);
     l1.front() = "six"; Print("l1 = ", l1);
     l1.back() = "seven"; Print("l1 = ", l1); // l1 = six
two seven
     Print("f1 = ", f1);
     f1.front() = 0.002; Print("f1 = ", f1); // f1 = 0.002
4.5 0.05
     Print("a1 = ", a1);
     a1[1] = "Вася";
     a1.back() = a1.front();
     Print("a1 = ", a1); // a1 = run Вася run
return 0;
}
```

2.2.6. Операции над итераторами последовательных контейнеров

Итераторы *векторов, деков* и *массивов* являются итераторами произвольного доступа. Итераторы *списков* могут быть лишь двунаправленными, так как список не обеспечивает произвольного доступа к элементам. *Последовательный список* обеспечивает только прямой обход, его итераторы могут быть только однонаправленными, а обратные итераторы не поддерживаются. Таким образом, к спискам нельзя применять алгоритмы, требующие итераторы произвольного доступа (табл. 2.6).

Таблица 2.6

Функции	**Описание**	**вектор**	**дек**	**список**	**послед. список**	**массив**
1	2	3	4	5	6	7
c.begin()	возвращает итератор произвольного доступа, установленный на первый элемент	+	+	+	+	+
c.end()	возвращает итератор произвольного доступа, установленный на позицию, следующую за последним элементом	+	+	+	+	+

1	2	3	4	5	6	7
c.cbegin()	возвращает константный итератор произвольного доступа, установленный на первый элемент	+	+	+	+	+
c.cend()	возвращает константный итератор, установленный на позицию, следующую за последним элементом;	+	+	+	+	+
c.rbegin()	возвращает обратный итератор, установленный на первый элемент при обратном обходе	+	+	+	-	+
c.rend()	возвращает обратный итератор, установленный на позицию, следующую за последним элементом при обратном обходе	+	+	+	-	+
c.crbegin()	возвращает константный обратный итератор, установленный на первый элемент при обратном обходе	+	+	+	-	+
c.crend()	возвращает константный обратный итератор, установленный на позицию, следующую за последним элементом при обратном обходе	+	+	+	-	+
c.before_begin()	возвращает однонаправленный итератор, установленный на позицию, предшествующую первому элементу	-	-	-	+	-
c.cbefore_begin()	возвращает константный однонаправленный итератор, установленный на позицию, предшествующую первому элементу	-	-	-	+	-

Листинг 6. Работа с итераторами однонаправленного списка:

```
int main() {
     forward_list  <string> f1({ "one", "two", "three" });
     f1.insert_after(f1.before_begin(), "four");
     Print("f1 = ", f1); // f1 = four one two three
     f1.insert_after(f1.begin(), "five");
     Print("f1 = ", f1); // f1 = four five one two three
     return 0;
}
```

2.2.7. Вставка и удаление элементов последовательных контейнеров

При использовании операций, предусмотренных для вставки и удаления элементов контейнера, должно гарантироваться, что аргументы являются корректными. Итераторы должны ссылаться на корректные позиции, а начало интервала должно предшествовать его концу (табл. 2.7).

Таблица 2.7

Функции	**Описание**	**вектор**	**дек**	**список**	**послед. список**	**массив**
1	2	3	4	5	6	7
c.push_back (elem)	добавляет копию аргумента *elem* в конец контейнера	+	+	+	-	-
c.pop_back ()	удаляет последний элемент (не возвращая его)	+	+	+	-	
c.push_front (elem)	вставляет копию элемента *elem* в начало контейнера	-	+	+	+	
c.pop_front()	удаляет первый элемент (не возвращая его)	-	+	+	+	
c.insert(pos, elem)	вставляет копию аргумента *elem* перед позицией итератора *pos* и возвращает позицию нового элемента	+	+	+	-	
c.insert(pos,n, elem)	вставляет n копий аргумента *elem* перед позицией итератора *pos* и возвращает позицию первого нового элемента	+	+	+	-	
c.insert(pos, beg, end)	вставляет копии всех элементов интервала *[beg,end)* перед позицией итератора *pos* и возвращает позицию первого нового элемента	+	+	+	-	
c.insert(pos, initlist)	вставляет копии всех элементов списка инициализации *initlist* перед позицией итератора *pos* и возвращает позицию первого элемента	+	+	+	-	

1	2	3	4	5	6	7
c.insert_after(pos, elem)	вставляет копию аргумента *elem* после позиции итератора *pos* и возвращает позицию нового элемента	-	-	-	+	
c.insert_after(pos,n, elem)	вставляет *n* копий аргумента *elem* после позиции итератора *pos*, и возвращает позицию первого нового элемента (или *pos*, если нового элемента нет)	-	-	-	+	
c.insert_after(pos, beg, end)	вставляет копии всех элементов интервала *[beg,end)* после позиции итератора *pos* и возвращает первый новый элемент (или позицию *pos*, если нового элемента нет)	-	-	-	+	
c.insert_after(pos, initlist)	вставляет копии всех элементов списка инициализации *initlist* после позиции итератора *pos* и возвращает позицию первого нового элемента	-	-	-	+	
c.emplace(pos, args...)	вставляет новый элемент, инициализированный списком аргументов *args* перед позицией итератора *pos*, и возвращает позицию нового элемента	+	+	+	-	
c.emplace_back (args ...)	добавляет в конец контейнера новый элемент, инициализированный списком аргументов *args*	+	+	+	-	
c.emplace_front (args ...)	вставляет новый элемент, инициализированный списком аргументов *args*, в начало контейнера (не возвращает ничего)	-	+	+	+	
c.emplace_after (pos, args...)	вставляет новый элемент, инициализированный списком аргументов *args* после позиции итератора *pos* и возвращает позицию нового элемента	-	-	-	+	
c.erase(pos)	удаляет элемент в позиции итератора *pos* и возвращает позицию следующего элемента	+	+	+	-	

1	2	3	4	5	6	7
c.erase(beg, end)	удаляет все элементы интервала *[beg,end)* и возвращает позицию следующего элемента	+	+	+	-	
c.erase_after(pos)	удаляет элемент, следующий за позицией итератора *pos* (ничего не возвращая)	-	-	-	+	
c.erase_after(beg, end)	удаляет все элементы интервала (*beg,end)* (ничего не возвращая)	-	-	-	+	
c.remove(val)	удаляет все элементы, имеющие значение *val*	-	-	+	+	
c.remove_if(op)	удаляет все элементы, для которых операция *op (elem)* возвращает *true*	-	-	+	+	
c.clear()	удаляет все элементы (опустошает контейнер)	+	+	+	+	

Примечание: в отличие от векторов и деков, при работе со списками следует использовать функцию-член *remove()*, а не сам алгоритм, потому что она манипулирует только внутренними указателями, а не элементами.

Листинг 7. Модификация содержимого последовательного контейнера:

```
int main() {
     vector<int> v1;
          Print("v1 = ", v1); // v1 =
     auto i = v1.begin();
     i = v1.emplace(i, 1);
          Print("v1 = ", v1); // v1 = 1
     i = v1.insert(i, 2);
          Print("v1 = ", v1); // v1 = 2 1
     i = v1.emplace(i, 3);
          Print("v1 = ", v1); // v1 = 3 2 1
     v1.erase(i++, ++i);
          Print("v1 = ", v1); // v1 = 3 2 1
     v1.erase(v1.begin(), i);
          Print("v1 = ", v1); // v1 = 1
     return 0;
}
```

Листинг 8. Модификация «хвоста» последовательного контейнера:

```
int main() {
     list<string> v1 = { "one", "two" };
```

```
    v1.emplace_back("three");
    v1.push_back("four");
    Print("v1 = ", v1); // v1 = one two three four
    v1.pop_back();
    v1.pop_back();
    Print("v1 = ", v1); // v1 = one two
    return 0;
}
```

Листинг 9. Модификация «головы» последовательного контейнера:

```
int main() {
    forward_list<string> v1 = { "one", "two" };
    v1.emplace_front("three");
    v1.push_front("four");
    Print("v1 = ", v1); // v1 = four three one two
    v1.pop_front(); v1.pop_front(); v1.pop_front();
    Print("v1 = ", v1); // v1 = two
    return 0;
}
```

Листинг 10. Использование функций, специфичных для однонаправленного списка:

```
int main() {
    forward_list<string> v1 = { "one", "two" };
    v1.emplace_after(v1.begin(), "three");
    Print("v1 = ", v1); // v1 = one three two
    auto i = v1.begin();
    v1.insert_after(++i, "four");
    Print("v1 = ", v1); // v1 = one three four two
    i = v1.begin();
    v1.erase_after(++i);
    Print("v1 = ", v1); // v1 = one three two
    return 0;
}
```

2.2.8. Специальные модифицирующие операции над списками

Связанные и последовательные списки имеют дополнительные модифицирующие функции-члены для изменения порядка и изменения связей между элементами и интервалами. Эти операции можно выполнять для перемещения элементов в списке или между двумя списками, при условии, что списки имеют одинаковые типы (табл. 2.8).

Таблица 2.8

Функции	**Описание**	**список**	**послед. список**
1	2	3	4
c.unique()	удаляет дубликаты последовательных элементов, имеющих одинаковые значения	+	+
c.unique(op)	удаляет дубликаты последовательных элементов, для которых операция *op()* возвращает *true*	+	+
c. splice (pos, c2)	перемещает все элементы списка *c2* в список с и размещает их перед позицией итератора *pos*	+	-
c. splice (pos, c2, c2pos)	перемещает элемент, занимающий позицию *c2pos* в списке *c2*, в список *c* и размещает его перед позицией *pos* (списки *c* и *c2* могут совпадать)	+	-
c. splice (pos, c2, c2beg,c2end)	перемещает все элементы интервала *[c2beg,c2end)* списка *c2* в список *c* и размещает их перед позицией *pos* списка *c* (списки *c* и *c2* могут совпадать)	+	-
c. splice_after(pos, c2)	перемещает все элементы списка *c2* в список с и размещает их сразу за позицией итератора *pos*	-	+
c. splice_after (pos, c2, c2pos)	перемещает элемент, занимающий позицию за *c2pos* в списке *c2*, в список *c* и размещает его сразу за позицией *pos* (списки *c* и *c2* могут совпадать)	-	+
c. splice_after (pos, c2, c2beg,c2end)	перемещает все элементы интервала *(c2beg,c2end)* списка *c2* в список *c* и размещает их сразу за позицией *pos* списка *c* (списки *c* и *c2* могут совпадать)	-	+
c.sort ()	упорядочивает все элементы с помощью оператора <	+	+
c.sort (op)	упорядочивает все элементы с помощью функционального объекта *op ()*	+	+

1	2	3	4
c.merge(c2)	если оба контейнера содержат упорядоченные элементы, перемещает все элементы списка *c2* в список *c,* так что все элементы сливаются и остаются упорядоченными	+	+
c.merge (c2,op)	если оба контейнера содержат упорядоченные элементы в соответствии с критерием *op()*, перемещает все элементы списка *c2* в список c, так что все элементы сливаются и остаются упорядоченными в соответствии с критерием *op()*	+	+
c.reverse ()	изменяет порядок следования всех элементов на противоположный	+	+

Листинг 11. Использование операций, специфичных для списков:

```
int main() {
     forward_list<int> mylist = { 7, 80, 7, 2, 5, 15, 85,
52, 6 };
     Print("mylist = ", mylist); // mylist = 7 80 7 2 5 15
85 52 6
     mylist.reverse();
     Print("mylist = ", mylist); // mylist = 6 52 85 15 5
2 7 80 7
     mylist.sort();
     Print("mylist = ", mylist);// mylist = 2 5 6 7 7 15
52 80 85
     mylist.sort(std::greater<int>());
     Print("mylist = ", mylist); // mylist = 85 80 52 15 7
7 6 5 2
     mylist.unique();
     Print("mylist = ", mylist); // mylist = 85 80 52 15 7
6 5 2
     return 0;
}
```

2.3. Ассоциативные контейнеры

Ассоциативные контейнеры (англ. *associative containers*) – это упорядоченные коллекции, обеспечивающие быстрый поиск данных и основанные на ключах. Каждый элемент ассоциативного контейнера занимает определенную позицию,

которая связана со значением его ключа и не зависит от времени и места вставки.

Поддержка работы с ассоциативными контейнерами представлена в заголовочных файлах *set* и *map*, которые содержат определение следующих шаблонных классов:

1. **Множество** (*set*) – коллекция, в которой хранятся уникальные элементы, следующие в определенном порядке (он определяется упорядочивающим отношением). Значение элемента множества также идентифицирует сам элемент (т.е. значение элемента является его ключом). Значения элементов в контейнере *set* не могут быть изменены (т.е. элементы обладают свойством *const*), но элементы могут быть вставлены или удалены из контейнера.
2. **Мультимножество** (*multiset*) – это множество, отличается от *set* лишь тем, что допускает включение одного и того же элемента по нескольку раз, например: {10, 20, 10,10, 30, 20}.
3. **Отображение** (*map*) – коллекция, в которой хранятся уникальные элементы – пары «ключ-значение», следующие в определенном порядке, устанавливаемом по значению ключа. Ключи элементов в контейнере *map* не могут быть изменены, но значения элементов могут быть изменены.
4. **Мультиотображение** (*multimap*) – словарь с дубликатами, отличающийся от *map* лишь тем, что допускает не уникальные ключи. Мультиотображение может также использоваться как *словарь*.

а также определение внешней шаблонной функции *swap* и перегруженных шаблонных операций: ==, !=, <, <=, >, >=.

Все ассоциативные контейнеры используют в качестве параметров *Key* (ключ) и упорядочивающее отношение *Compare*, которое вызывает полное упорядочение по элементам *Key*.

Кроме того, *map* и *multimap* ассоциируют (связывают) данные произвольного типа с ключом. Объект типа *Compare* называется *сравнивающим объектом* (*comparison object*)

контейнера (по сути – это функция с двумя параметрами, возвращающая булевское значение).

Ассоциативные контейнеры реализованы в форме *красно-черных деревьев* (*Red-Black-Tree, RB-Tree*) – это одно из самобалансирующихся двоичных деревьев поиска, гарантирующее логарифмический рост высоты дерева от числа узлов и быстро выполняющее основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева – «цвета». Этот атрибут может принимать одно из двух возможных значений – «чёрный» или «красный» (рис. 2.3).

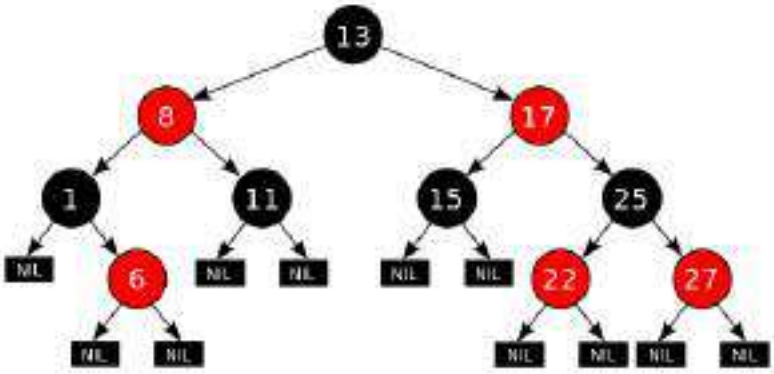


Рис. 2.3. Внутреннее устройство ассоциативных контейнеров

2.3.1. Создание, копирование и удаление ассоциативных контейнеров

Как и последовательные контейнеры, ассоциативные контейнеры имеют несколько перегруженных конструкторов и могут создаваться разными способами. Для уничтожения всех элементов контейнера и высвобождения памяти необходимо использовать деструктор (табл. 2.9).

Таблица 2.9

Синтаксис	**Описание**
1	2
set<Elem, Op> s *multiset<Elem, Op> ms* *map<Key, Val, Op> m* *multimap<Key, Val, Op> mp*	создание пустого контейнера с помощью конструктора по умолчанию.

1	2
set<Elem, Op> s(s2) *multiset<Elem, Op> ms(ms2)* *map<Key, Val, Op> m(m2)* *multimap<Key, Val, Op> mp(mp2)*	конструктор копирования (перемещения)
set<Elem, Op> s=s2 *multiset<Elem, Op> ms=ms2* *map<Key, Val, Op> m =m2* *multimap<Key, Val, Op> mp = mp2*	
set<Elem, Op> s (beg, end) *multiset<Elem, Op> ms(beg, end)* *map<Key, Val, Op> m (beg, end)* *multimap<Key, Val, Op> mp(beg, end)*	создание контейнера, заполненного диапазоном значений из интервала *[beg, end)*
set<Elem, Op> s (initlist) *multiset<Elem, Op> ms (initlist)* *map<Key, Val, Op> m (initlist)* *multimap<Key, Val, Op> mp(initlist)*	создание контейнера с помощью списка инициализации
set<Elem, Op> s = initlist *multiset<Elem, Op> ms = initlist* *map<Key, Val, Op> m = initlist* *multimap<Key, Val, Op> mp = initlist*	
s.~ set() *ms.~multiset()* *m.~map()* *mp.~ multimap()*	вызов деструктора контейнера

Примечание: предикат *Op* является необязательным *шаблонным параметром* определяющим критерий сортировки элементов ассоциативного контейнера. Если не указан никакой конкретный критерий сортировки, используется функциональный объект *less<>*, заданный по умолчанию и упорядочивающий элементы с помощью оператора <.

Конструкторы по умолчанию, диапазона и списка инициализации имеют перегрузки, позволяющие передать критерий сортировки как *параметр конструктора*.

Листинг 12. Конструирование множеств, используя критерий сортировки по умолчанию:

```
template <class T> void Print(string s, T q) {
     cout << s; for (auto& i : q) cout << " " << i; cout
<< endl;
}
int main() {
     set<int> s1; // Конструктор по умолчанию
     Print("s1 = ", s1); // s1 =
     set<int> s2({ 20, 10, 40, 30, 50 }); // Список
инициализации
     Print("s2 = ", s2); // s2 = 10 20 30 40 50
     set<int> s3(s2); // Конструктор копирования
     Print("s3 = ", s3); // s3 = 10 20 30 40 50
     auto i = s2.begin(), j = s2.end();
     set<int> s4(++i, --j); // Диапазон с итераторами
     Print("s4 = ", s4); // s4 = 20 30 40
     int myints[] = { 10, 20, 30, 20, 20 };
     multiset<int> ms1(myints, myints + 5); // Диапазон
значений
     Print("ms1 = ", ms1); // ms1 = 10 20 20 20 30
     return 0;
}
```

Листинг 13. Конструирование отображений, используя критерий сортировки по умолчанию:

```
template <class T> void Print(string s, T q) {
     cout << s; for (auto& i : q) cout << " " << i.first
<< ':' << i.second; cout << endl;
}
int main() {
     map<char, int> m1; // Конструктор по умолчанию
     m1['a'] = 10; m1['b'] = 30;
     m1['c'] = 50; m1['d'] = 70;
     Print("m1 = ", m1); // m1 = a:10 b:30 c:50 d:70
     map<char, int> m2(m1.begin(), m1.end());// Диапазон с
итераторами
     Print("m2 = ", m2); // m2 = a:10 b:30 c:50 d:70
     map<char, int> m3(m2); // Конструктор копирования
     Print("m3 = ", m3); // m3 = a:10 b:30 c:50 d:70
     auto i = m2.begin(), j = m2.end();
     map<char, int> m4(++i, --j); // Диапазон с
итераторами
     Print("m4 = ", m4); // m4 = b:30 c:50
     map<char, int> m5(move(m1)); // Конструктор
перемещения
     Print("m5 = ", m5); m5 = a:10 b:30 c:50 d:70
     Print("m1 = ", m1); // m1 =
     return 0;
}
```

Листинг 14. Конструирование множеств с указанием критерия сортировки:

```
bool fncomp(int lhs, int rhs) { return lhs>rhs; }
struct classcomp {
     bool operator() (const int& lhs, const int& rhs)
const
     {
          return lhs>rhs;
     }
};
int main()
{
set<int> s1({ 50, 10, 30, 20, 40 });//Список инициализации
                                          //с сортировкой по
умолчанию
     Print("s1 = ", s1); });// s1 = 10 20 30 40 50
     set<int, classcomp> s2; // Конструктор по умолчанию с
шаблонным
                                       //параметром в качестве
критерия сортировки
     Print("s2 = ", s2); // s2 =
     s2.insert({ 15, 25, 5, 45, 35 });
     Print("s2 = ", s2); // s2 = 45 35 25 15 5
     set<int, classcomp> s3({ 15, 25, 5, 45, 35 });
//Диапазон значений
     Print("s3 = ", s3); // s3 = 45 35 25 15 5
     set<int, classcomp> s4(s3); // Конструктор
копирования
     Print("s4 = ", s4); // s4 = 45 35 25 15 5
     set<int, classcomp> s5(s1.begin(), s1.end());//
Диапазон с итераторами
     Print("s5 = ", s5); // s5 = 50 40 30 20 10
     set<int, bool(*)(int, int)> s6(fncomp); //
Конструктор по умолчанию
                                        //с параметром в
качестве критерия сортировки
     s6.insert({ 15, 25, 5, 45, 35 });
     Print("s6 = ", s6); // s6 = 45 35 25 15 5
     set<int, bool(*)(int, int)> s7(s1.begin(), s1.end(),
fncomp);
                                                          //
Диапазон с итераторами
     Print("s7 = ", s7); // s7 = 50 40 30 20 10
     set<int, bool(*)(int, int)> s8({ 15, 25, 5, 45, 35 },
fncomp);
                                       //Список инициализации
     Print("s8 = ", s8); // s8 = 45 35 25 15 5
     return 0;
}
```

2.3.2. Управление размером ассоциативных контейнеров

Ассоциативные контейнеры имеют следующие функции для работы с размером (табл. 2.10).

Таблица 2.10

Функции	**Описание**
max_size()	возвращает максимально возможное количество элементов
size()	возвращает текущее количество элементов
empty()	возвращает признак того, что контейнер пуст (эквивалент *size() ==0*)

Листинг 15. Использование функций управления размером:

```
int main()
{
     set<int> s1;
     PrintS("s1 = ", s1);
     cout << "s1: max_size = " << s1.max_size()
          << ",  size =  " << s1.size() << ", empty = " <<
s1.empty() << endl;
// s1: max_size = 461168601842738790,  size =  0, empty =
1
     s1.insert({ 15, 25, 5, 45, 35 });
     PrintS("s1 = ", s1);
     cout << "s1: max_size = " << s1.max_size()
          << ", size =  " << s1.size() << ", empty = "
<<(s1.size() == 0)<< endl;
               // s1: max_size = 461168601842738790,  size
=  5, empty = 0
     map<char, int> m1;
     PrintM("m1 = ", m1);
     cout << "m1: max_size = " << m1.max_size()
          << ",  size =  " << m1.size() << ", empty = " <<
m1.empty() << endl;
// m1: max_size = 461168601842738790,  size = 0, empty = 1
     m1['a'] = 10; m1['b'] = 30;
     m1['c'] = 50; m1['d'] = 70;
     PrintM("m1 = ", m1);
     cout << "m1: max_size = " << m1.max_size()
          << ", size =  " << m1.size() << ", empty = " <<
(m1.size() == 0)<<endl;
// m1: max_size = 461168601842738790,  size = 4, empty = 0
     return 0;
}
```

2.3.3. Операторы сравнения ассоциативных контейнеров

Для всех типов ассоциативных контейнеров определены обычные операторы сравнения ==, ! =, <, <=, > и >=, подчиняющиеся правилам, используемым при сравнении последовательных контейнеров, а также две дополнительные функции возвращающие критерий сравнения *key_comp* и *value_comp* (табл. 2.11).

Таблица 2.11

Операторы сравнения	**Действие**
c.key_comp()	возвращает копию сравнивающего объекта (*comparison object*), который используется для сравнения ключей контейнера (фактически возвращается указатель на функцию)
c.value_comp()	возвращает копию сравнивающего объекта (*comparison object*), который используются для сравнения элементов контейнера по ключу (т.е. value_comp работает непосредственно с элементами, а не с ключами, как key_comp)
c1 == c2	возвращает признак того, что контейнер *c1* равен контейнеру *c2*
c1!= c2	возвращает признак того, что контейнер *c1* не равен контейнеру *c2* (эквивалент *! (c1==c2)*)
c1 < c2	возвращает признак того, что контейнер *c1* меньше контейнера *c2*
c1 > c2	возвращает признак того, что контейнер *c1* больше контейнера *c2*
c1 <= c2	возвращает признак того, что контейнер *c1* меньше или равен контейнеру *c2*
c1 >= c2	возвращает признак того, что контейнер *c1* больше или равен контейнеру *c2*

Листинг 16. Сравнение ключей контейнера:

```
int main()
{
     set<int> s1{ 1, 2, 3, 4, 5 };
     PrintS("s1 = ", s1); // s1 = 1 2 3 4 5
     auto mycomp = s1.key_comp();
     cout << boolalpha << mycomp(3, 5) << endl; // true
     auto f = [](int lhs, int rhs) {return lhs>rhs; };
     set<int, bool(*)(int, int)> s2({ 1, 2, 3, 4, 5 }, f);
     PrintS("s2 = ", s2); // s1 = 5 4 3 2 1
```

```
    auto mycomp1 = s2.key_comp();
    cout << boolalpha << mycomp1(3, 5) << endl; // false
    return 0;
}
```

Листинг 17. Сравнение элементов контейнера по ключу:

```
int main()
{
    map<char, int> m1;
    auto mycomp1 = m1.key_comp();
    cout << boolalpha << "key_comp ('a', 'f'):" <<
mycomp1('a', 'f')<<endl;// true
    cout << boolalpha << "key_comp ('f', 'a'):"
<<mycomp1('f', 'a')<<endl;// false
    pair<char, int> p1{ 'w', 1 }, p2{ 'f', 10 };
    auto mycomp2 = m1.value_comp();
    cout << boolalpha << "value_comp:" << mycomp2(p1, p2)
<< endl; // false
    return 0;
}
```

2.3.4. Операции присваивания ассоциативных контейнеров

Ассоциативные контейнеры предусматривают только основные операции присваивания, характерные для всех контейнеров (табл. 2.12). При выполнении этих операций оба контейнера должны иметь одинаковый тип. В частности, тип критерия сравнения должен быть тем же самым, хотя сам критерий сравнения может быть другим.

Таблица 2.12

Функции	**Описание**
c = c2	присваивает все элементы контейнера *c2* контейнеру *c*
c = rv	перемещает все элементы контейнера *rv* в контейнер *c*
c1.swap(c2) *swap (c1, c2)*	обменивает элементы контейнеров *c1* и *c2*
c= initlist	присваивает контейнеру *c* все элементы списка инициализации *initlist*

Листинг 18. Присваивание контейнеров:

```
int main()
{
    map<char, int> first({ { 'f', 1 }, { 's', 2 }, { 'b',
3 } }), second;
```

```
     PrintM("first = ", first);  // first =  b:3 f:1 s:2
     second = first; PrintM("second = ", second); //
second =  b:3 f:1 s:2
     map<double, char>
third({ { 1.2, 'x' }, { 0.05, 'y' }, { 4.5, 'z' } }),
fourth;
     PrintM("third = ", third); // third =  0.05:y 1.2:x
4.5:z
     fourth = move(third);
     PrintM("fourth = ", fourth); // fourth =  0.05:y
1.2:x 4.5:z
PrintM("third = ", third); // third =
     map<string, int> fifth;
     fifth = {{"Valencia", 15}, {"Anita", 19}, { "Sofia",
16 }, { "Agata", 18 } };
PrintM("fifth = ", fifth); // third = Agata:18 Anita:19
Sofia:16 Valencia:15
     third = { { 1.5, 'x' }, { -5, 'y' }, { 0, 'z' } };
     third.swap(fourth);
     PrintM("fourth = ", fourth); // fourth =  -5:y 0:z
1.5:x
PrintM("third = ", third); // third =  0.05:y 1.2:x 4.5:z
     return 0;
}
```

2.3.5. Прямой доступ к элементам ассоциативных контейнеров

Отображение является единственным ассоциативным контейнером, обеспечивающим прямой доступ к элементам (табл. 2.13).

Таблица 2.13

Функции	**Описание**
c[key]	вставляет элемент с ключом *key*, если его еще не было, и возвращает ссылку на значение элемента с ключом *key* (только для не константных отображений)
c.at(key)	возвращает ссылку на значение элемента с ключом *key*

В операторе *[]* индекс также является ключом, используемым для идентификации элемента. Это значит, что в операторе *[]* индекс может иметь любой тип, а не только целочисленный. Такой интерфейс называется *ассоциативным массивом*.

Тип индекса в операторе *[]* для ассоциативных массивов – не единственное отличие от обычных массивов. Кроме того, индекс не бывает неправильным. Если ключом является индекс, для которого не существует ни одного элемента, в отображение автоматически вставляется новый элемент. Значение нового элемента инициализируется конструктором по умолчанию для соответствующего типа. Таким образом, для использования этой возможности нельзя использовать тип значения, не имеющего конструктора, заданного по умолчанию.

Листинг 19. Использование операторов прямого доступа:

```
int main()
{
     map<char, int> first({ { 'f', 1 }, { 's', 2 }, { 'b',
3 } });
     PrintM("first = ", first); // first =  b:3 f:1 s:2
     cout << first['f'] << "  " << first[3] << "  " <<
first[102] << endl;
// 1 0 1
     PrintM("first = ", first); // first =  ♥:0 b:3 f:1
s:2
     cout << first.at('b') << endl;  // 3
     return 0;
}
```

2.3.6. Операции над итераторами ассоциативных контейнеров

Ассоциативные контейнеры не предоставляют прямой доступ к элементам (кроме случаев, описанных в п. 2.3.5), а их итераторы являются двунаправленными, следовательно, к ним нельзя применять алгоритмы, требующие итераторы произвольного доступа (табл. 2.14).

Таблица 2.14

Функции	**Описание**
1	2
c.begin()	возвращает двунаправленный итератор, установленный на первый элемент
c.end()	возвращает двунаправленный итератор, установленный на позицию, следующую за последним элементом

1	2
c.cbegin()	возвращает константный двунаправленный итератор, установленный на первый элемент
c.cend()	возвращает константный двунаправленный итератор, установленный на позицию, следующую за последним элементом
c.rbegin()	возвращает обратный итератор, установленный на первый элемент в обратном обходе
c.rend()	возвращает обратный итератор, установленный на позицию, следующую за последним элементом в обратном обходе
c.crbegin()	возвращает константный обратный итератор, установленный на первый элемент при обратном обходе
c.crend()	возвращает константный обратный итератор, установленный на позицию, следующую за последним элементом при обратном обходе

Примечание: с точки зрения итератора все элементы ассоциативных контейнеров считаются константными. Необходимо гарантировать, что при изменении значений элементов порядок их следования не будет нарушен. Вследствие этого ограничения к элементам ассоциативных контейнеров невозможно применить ни один модифицирующий алгоритм. Например, нельзя вызвать алгоритм *remove()*, потому что он удаляет элементы путем замещения удаленных элементов следующими элементами. Для удаления элементов можно использовать только члены-функции контейнера.

Листинг 20. Использование итераторов на примере отображения:

```
 int main()
{
     map<char, int> first({ { 'f', 1 }, { 's', 2 }, { 'b',
3 } });
     auto i = first.begin();
     while (i != first.end())
     {
          cout << " " << (*i).first << ':' << (*i).second;
          i++;
     } // b:3 f:1 s:2
     cout << endl;
     auto j = first.rbegin();
```

```
        while (j != first.rend())
        {
             cout << " " << (*j).first << ':' << (*j).second;
             j++;
        } // s:2 f:1 b:3
        return 0;
}
```

2.3.7. Вставка и удаление элементов ассоциативных контейнеров

При использовании операций, предусмотренных для вставки и удаления элементов контейнера, должно гарантироваться, что аргументы являются корректными. Итераторы должны ссылаться на корректные позиции, а начало интервала должно предшествовать его концу (табл. 2.15).

Таблица 2.15

Функции	**Описание**	**множества**	**мультимн-ва**	**отображение**	**мультиотобр.**
1	2	3	4	5	6
c. insert (val)	вставляет копию значения *val*, возвращает пару *pair<iterator,bool>*, первый элемент которой *pair::first* – это итератор, указывающий на новый (вставленный) элемент или на существующий точно такой же элемент, второй элемент пары *pair::second* имеет булевское значение – признак успешного выполнения вставки.	+	-	+	-
c.insert(pos, val)	вставляет копию значения *val* и возвращает позицию нового элемента (параметр *pos* используется как подсказка, указывающая на позицию, с которой следует начинать поиск места для вставки)	+	+	+	+
c.insert(beg, end)	вставляет копии всех элементов интервала *[beg,end)* (не возвращая ничего)	+	+	+	+

1	2	3	4	5	6
c.insert(initlist)	вставляет копии всех элементов списка инициализации *initlist* (не возвращая ничего)	+	+	+	+
c.emplace(args...)	вставляет новый элемент, инициализированный списком аргументов *args*, и возвращает позицию нового элемента признак успешного выполнения вставки	+	-	+	-
c.emplace_hint (pos, args...)	вставляет новый элемент, инициализированный списком аргументов *args* , и возвращает позицию нового элемента (параметр *pos* используется как подсказка, указывающая на позицию, с которой следует начинать поиск места для вставки)	+	+	+	+
c.erase(val)	удаляет все элементы, равные *val*, и возвращает количество удаленных элементов	+	+	+	+
c.erase(pos)	удаляет элемент, занимающий позицию, на которую указывает итератор *pos*, и возвращает следующую позицию	+	+	+	+
c.erase(beg, end)	удаляет все элементы интервала *[beg,end)* и возвращает следующую позицию	+	+	+	+
c.clear()	удаляет все элементы (опустошает контейнер)	+	+	+	+

Примечание: типы значений, возвращаемых функциями-членами для вставки элементов *insert()* и *emplace()*, отличаются друг от друга. Разница между типами возвращаемых значений объясняется тем, что мультимножества и мультиотображения допускают дубликаты, а множества и отображения – нет. Таким образом, вставка элемента в множество или отображение может завершиться неудачно, если оно уже содержит элемент с таким же значением.

Листинг 21. Использование операций вставки:

```
int main() {
     set<int> myset;
     auto it = myset.begin();
```

```
    for (int i = 1; i <= 5; ++i) myset.insert(i * 10);
    Print("myset = ", myset); // myset = 10 20 30 40 50
    auto ret = myset.insert(20); //  элемент 20 не будет
вставлен
    Print("myset = ", myset);
    if (ret.second == false) it = ret.first;  // "it"
указывает на 20
    myset.insert(it, 25); // вставка с «подсказкой»
(эффективно)
    myset.insert(it, 24); // вставка с «подсказкой»
(эффективно)
    myset.insert(it, 26); // вставка с «подсказкой» (не
эффективно)
    Print("myset = ", myset); myset = 10 20 24 25 26 30
40 50
    int myints[] {5, 10, 15};
    myset.insert(myints, myints + 3); // элемент 10 не
будет вставлен
    Print("myset = ", myset); myset = 5 10 15 20 24 25 26
30 40 50
    myset.insert({ 44, 33, 22, 11 });
    Print("myset = ", myset); myset = 5 10 11 15 20 22 24
25 26 33 30 40 44 50
    return 0;
}
```

Листинг 22. Использование операций удаления и обмена:

```
int main() {
    set<int> myset, myset1;
    for (int i = 1; i<10; ++i) myset.insert(i * 10);
    Print("myset = ", myset); //myset = 10 20 30 40 50 60
70 80 90
    myset1 = myset;
    auto it = myset.begin();
    ++it;  // "it" указывает на 20
    myset.erase(it); // удаление по итератору
    Print("myset = ", myset); //myset = 10 30 40 50 60 70
80 90
    myset.erase(40); // удаление по значению
    Print("myset = ", myset); //myset = 10 30 50 60 70 80
90
    myset.erase(myset.find(60), myset.end()); // удаление
диапазона
    Print("myset = ", myset); //myset = 10 30 50
    myset.swap(myset1);
    Print("myset = ", myset); // myset = 10 20 30 40 50
60 70 80 90
    Print("myset1 = ", myset1); //myset1 = 10 30 50
    myset.clear();
    Print("myset = ", myset); // myset =
```

```
    return 0;
}
```

2.3.8. Специальные операции поиска

Ассоциативные контейнеры оптимизированы для поиска элементов, и поэтому имеют функции-члены поиска, являющиеся специальными версиями общих алгоритмов (табл. 2.16).

Таблица 2.16

Функции	**Описание**
c.count(val)	возвращает количество элементов, имеющих значение *val*
c. find(val)	возвращает позицию первого элемента, имеющего значение *val* (или позицию *end ()*, если элемент не найден)
c.lower_bound (val)	возвращает первую позицию, в которую можно вставить элемент со значением *val* (первый элемент, удовлетворяющий условию >= *val*)
c.upper_bound (val)	возвращает последнюю позицию, в которую можно вставить элемент со значением *val* (первый элемент, удовлетворяющий условию > *val*)
c.equal_range (val)	возвращает интервал со всеми элементами, значения которых равны *val* (т.е. первую и последнюю позиции, в которые можно вставить элемент со значением *val*)

Листинг 23. Работа с границами (диапазонами):

```
int main() {
    multiset<int> ms{ 10, 20, 30, 20, 20, 40, 50, 30, 50
};
    Print("ms = ", ms); // ms = 10 20 20 20 30 30 40 50
50
    cout << "count(20) = " << ms.count(20) << endl;
    auto i = ms.lower_bound(20),
    j = ms.upper_bound(40);
    multiset<int> ms1(i, j);
    Print("ms1 = ", ms1); // ms1 = 20 20 20 30 30 40
    auto p = ms.equal_range(30);
    multiset<int> ms2(p.first, p.second);
    Print("ms2 = ", ms2); // ms2 = 30 30
    return 0;
}
```

2.4. Итераторы

2.4.1. Основные понятия и определения

Итераторы (*англ. iterator - перечислитель*) – это объекты, которые могут обходить элементы последовательности с помощью универсального интерфейса, напоминающего интерфейс обычных указателей. Итераторы следуют принципу чистой абстракции: *все, что ведет себя как итератор, является итератором*.

Каждый контейнер определяет собственные типы итераторов, а для использования специальных итераторов (например, обратных) и некоторых функции для итераторов необходимо подключить заголовочный файл <*iterators*>.

Итераторы подразделяются на 5 категорий:

1. **Итераторы вывода** *(output iterators)* – выполняют перемещение вперед, не поддерживая повторного обхода одного и того же диапазона, записывая элемента только один раз. К итераторам вывода не применяются операции сравнения.
2. **Итераторы ввода** *(input iterators)* – выполняют перемещение вперед, не поддерживая повторного обхода одного и того же диапазона, считывая элемент только один раз. Для итераторов ввода перегружены операции сравнения == и !=.
3. **Однонаправленные итераторы** *(forward iterators)* – это итераторы ввода, гарантирующие, что при чтении в прямом направлении для двух однонаправленных итераторов, ссылающихся на один и тот же элемент, операция == возвращает значение *true* и что после инкремента обоих итераторов они будут ссылаться на один и тот же элемент.
4. **Двунаправленные итераторы** *(bidirectional iterators)* – это однонаправленные итераторы, обладающие дополнительной возможностью перемещаться по элементам в обратном направлении, т.е. предусматривают оператор декремента для перемещения назад

5. **Итераторы произвольного доступа** *(random access iterators)* – это двунаправленные итераторы, обладающие возможностью произвольного доступа к элементам. Следовательно, они реализуют операции арифметики итераторов (по аналогии с арифметикой обычных указателей).

Примечание: однонаправленные, двунаправленные и итераторы произвольного доступа, удовлетворяющие требованиям итераторов вывода, являются *модифицирующими итераторами* и могут использоваться как для чтения, так и для записи.

Итераторы представляют собой иерархию, где более мощная категория итератора поддерживает деятельность менее мощной категории. Ниже в табл. 2.17 приведена иерархия итераторов с указанием допустимых операций для каждой категории.

Таблица 2.17

Категории итераторов				**Операции**	**Описание**
Все категории				*X b(iter);*	копирует итератор (копирующий конструктор)
				++ iter	перемещает итератор вперед (возвращает новую позицию)
				iter ++	перемещает итератор вперед (возвращает старую позицию)
Произвольного доступа	*Двунаправленные*	*Однонаправленные*	*Ввода*	*iter == iter2*	возвращает *true*, если два итератора равны
				iter != iter2	возвращает *true*, если два итератора не равны
				** iter*	обеспечивает доступ для чтения к текущему элементу
				iter ->m	обеспечивает доступ для чтения к члену текущего элемента
			Вывода	** iter =val*	записывает значение *val* в то место, на которое ссылается итератор
				X iter; X()	создает итератор (конструктор по умолчанию)
				iter1 = iter2;	присваивает итератор
				-- iter	перемещает итератор назад (возвращает новую позицию)
				iter --	перемещает итератор назад (возвращает старую позицию)

		iter[n]	обеспечивает доступ к элементу с индексом *n*
		iter+=n	перемещает итератор на *n* элементов вперед (или назад, если *n* отрицательное)
		iter-=n	перемещает итератор на *n* элементов назад (или вперед, если *n* отрицательное)
		iter+n	возвращает итератор, установленный на элемент, расположенный на *n* позиций вперед
		iter-n	возвращает итератор, установленный на элемент, расположенный на *n* позиций назад
		iter1-iter2	возвращает расстояние между итераторами *iterl* и *iter2*
		iter1<iter2	возвращает результат проверки того, что итератор *iterl* предшествует итератору *iter2*
		iter1 >iter2	возвращает результат проверки того, что итератор *iter2* предшествует итератору *iter2*
		iter1 <=iter2	возвращает результат проверки того, что итератор *iter2* не предшествует итератору *iter2*
		iterl>=iter2	возвращает результат проверки того, что итератор *iter1* предшествует итератору *iter2*

Листинг 24. Использование итератора ввода в алгоритме *for_each*:

```
template <typename InputIterator, typename Function>
Function for_each (InputIterator first, InputIterator
last, Function f)
{
while (first != last) f(*first++);
return f;
}

void PrintVal(int num)
{
     cout << num << " ";
}

int main() {
     int init[] = { 1, 2, 3, 4, 5 };
     auto _f = for_each(init, init + 5, PrintVal); //1 2 3
4 5
     _f(100); //1 2 3 4 5 100
     return 0;
}
```

Листинг 25. Пример использования итераторов вывода:

```
int main()
{
```

```
    int init1[] = { 1, 2, 3, 4, 5 };
    int init2[] = { 6, 7, 8, 9, 10 };
    vector <int> v(10);
    merge(init1, init1 + 5, init2, init2 + 4, v.begin());
    copy(v.begin(), v.end(), ostream_iterator<int>(cout,
", "));
    //1, 2, 3, 4, 5, 6, 7, 8, 9, 0,
    return 0;
}
```

Листинг 26. Использование однонаправленного итератора в алгоритме *replace*:

```
template<typename ForwardIterator, typename T>
void replace(ForwardIterator first, ForwardIterator last,
const T& old_value,
    const T& new_value)
{
    while (first != last)
    {
        if (*first == old_value) *first = new_value;
        ++first;
    }
}

int main()
{
    int init[] = { 1, 2, 3, 4, 5 };

    replace(init, init + 5, 0, 2); //1 2 3 4 5
    replace(init, init + 5, 1, 0); //0 2 3 4 5
    replace(init, init + 5, 2, 1); //0 1 3 4 5
    copy(init, init + 5, ostream_iterator<int>(cout, "
"));
    return 0;
}
```

Листинг 27. Использование двунаправленного итератора в алгоритме *reverse*:

```
template<typename BidirectionalIterator>
void reverse(BidirectionalIterator first,
BidirectionalIterator last)
{
    if (first == last)
        return;
    --last;
    while (first < last)
    {
        iter_swap(first, last);
        ++first;
```

```
            --last;
        }
    }
    int main()
    {
        int init[] = { 1, 2, 3, 4, 5 };
        reverse(init, init + 5);
        copy(init, init + 5, ostream_iterator<int>(cout, "
    ")); //5 4 3 2 1
        return 0;
    }
```

Листинг 28. Использование итератора произвольного доступа:

```
    int main()
    {
        const int init[] = { 1, 2, 3, 4, 5 };
        vector<int> v(5);
        vector<int>::iterator it; //итератор произвольного
    доступа
        copy(init, init + 5, it = v.begin());
        cout << *(it + 4) << endl; //5
        cout << *(it += 3) << endl; //4
        cout << *(it -= 1) << endl; //3
        cout << *(it = it - 1) << endl;//2
        cout << *(--it) << endl; //1

        for (int i = 0; i < (v.end() - v.begin()); i++)
            cout << it[i] << " ";   //1 2 3 4 5
    }
```

2.4.2. Вспомогательные функции для работы с итераторами

Заголовочный файл <*iterators*> содержит несколько вспомогательных функций для работы с итераторами (табл. 2.18).

Таблица 2.18

Функции	**Описание**
1	2
void advance (InputIterator& it, Distance n)	переносит итератор ввода *it* вперед на указанное количество позиций *n*. Для двунаправленных итераторов и итераторов произвольного доступа число *n* может быть отрицательным, задавая перемещение назад

1	2
ForwardIterator next (ForwardIterator it, typename iterator_traits<ForwardIterator>::difference_type n = 1)	возвращают позицию прямого итератора *it*, в которую он попал бы, если бы переместился на *n* позиций вперед. Для двунаправленных итераторов и итераторов произвольного доступа число *n* может быть отрицательным, задавая перемещение назад
BidirectionalIterator prev (BidirectionalIterator it, typename iterator_traits<BidirectionalIterator>::difference_type n = 1)	возвращают позицию двунаправленного итератора *it*, в которую он попал бы, если бы переместился на *n* позиций назад. Для перехода на последующие позиции число *n* может быть отрицательным.
typename iterator_traits<InputIterator>::difference_type distance (InputIterator first, InputIterator last)	возвращает расстояние между итераторами ввода *first* и *last*. Оба итератора должны ссылаться на элементы, принадлежащие одному и тому же контейнеру. Если итераторы не являются итераторами произвольного доступа, то позиция итератора *pos2* должна быть достижимой из позиции итератора *posl*; иначе говоря, итератор *pos2* должен иметь ту же позицию или позицию, расположенную правее

Листинг 29. Использование вспомогательных функций для работы с итераторами:

```
 int main() {
     list<int> mylist;
     for (int i = 0; i<10; i++) mylist.push_back(i * 10);
     Print("mylist = ", mylist); // mylist =  0 10 20 30
40 50 60 70 80 90
     list<int>::iterator it = mylist.begin();
     advance(it, 5);
     cout << *it << '\n'; // 50
     list<int>::iterator it_next = next(it, 5);
     cout << boolalpha << (it_next == mylist.end()) <<
'\n'; // true
     it_next = next(it, -5);
     cout << *it_next << '\n'; // 0
     cout << distance(it_next, mylist.end()) << '\n'; //
10
     return 0;
}
```

2.4.3. Адаптеры итераторов

Заголовочный файл <*iterators*> содержит специальные итераторы позволяющие алгоритмам выполнять обратный обход в режиме вставки и работать с потоками:

1. **Обратные итераторы** *(reverse iterators)* – переопределяют операции инкремента и декремента так, что их семантика становится обратной. Следовательно, если использовать эти итераторы вместо обычных, алгоритмы будут выполняться в обратном направлении. Функция-член *rbegin()* возвращает позицию первого элемента при обратном обходе. Следовательно, она возвращает позицию последнего элемента. Функция-член *rend()* возвращает позицию, расположенную за последним элементом при обратном обходе. Следовательно, она возвращает позицию, предшествующую первому элементу.
2. **Итераторы вставки** *(insert iterators)* – это адаптеры итераторов, преобразовывающие присвоение нового значения во вставку этого нового значения. Используя итераторы вставки, алгоритмы могут вставлять элементы, а не заменять их. Все итераторы вставки относятся к категории итераторов вывода. Таким образом, они обеспечивают только возможность присваивать новые значения.
3. **Итератор потока** *(stream iterator)* – это адаптер итератора, позволяющий использовать поток в качестве источника или получателя данных для алгоритмов. В частности, итератор потока ввода можно использовать для чтения элементов из потока ввода, а итератор потока вывода — для записи элементов в поток вывода.
4. **Константные итераторы** *(constant iterator)* – образуется путем модификации основного итератора и не допускает изменения данных, на которые ссылается, т.е *не являются итераторами вывода*. Можно считать константный итератор указателем на константу.

5. **Итераторы перемещения** *(move iterator)* – позволяют алгоритмам перемещать, а не копировать элементы из одного диапазона в другой.

Листинг 30. Использование реверсивных итераторов:

```
void print(int elem)
{
     cout << elem << " ";
}
int main() {
     vector<int> coll = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
     for_each(coll.begin(), coll.end(), print); // 1 2 3 4
5 6 7 8 9
     cout << endl;
     for_each(coll.rbegin(), coll.rend(), print); //9 8 7
6 5 4 3 2 1
     cout << endl;
     vector<int>::iterator pos;
     pos = find(coll.begin(), coll.end(), 5);
     cout << "pos = " << *pos << endl; //5
     vector<int>::reverse_iterator rpos(pos);
     cout << "rpos = " << *rpos << endl; //4
     vector<int>::iterator pos1 = find(coll.begin(),
coll.end(), 2);
     vector<int>::iterator pos2 = find(coll.begin(),
coll.end(), 7);
     for_each(pos1, pos2, print); // 2 3 4 5 6
     cout << endl;
     vector<int>::reverse_iterator rpos1(pos1);
     vector<int>::reverse_iterator rpos2(pos2);
     for_each(rpos2, rpos1, print); //6 5 4 3 2
     cout << endl;
     return 0;
}
```

Примечание: разработчики обратных итераторов использовали трюк: они «физически» изменили на противоположный «принцип полуоткрытости»: начало *не включено* в диапазон, а конец *включен*. Следовательно, существует различие между физической позицией, определяющей элемент, на который ссылается итератор, и логической позицией, определяющей значение, на которое ссылается итератор (рис. 2.4).

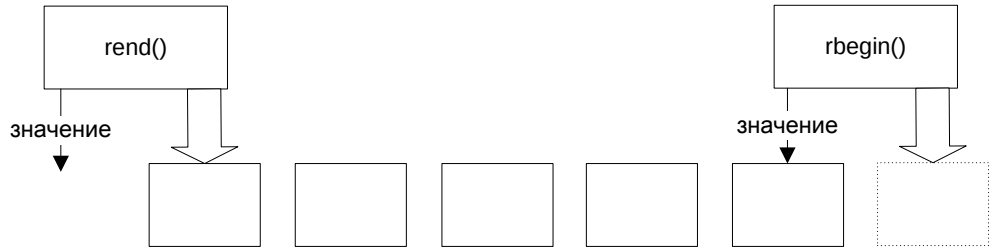


Рис.2.4. Позиция и значение обратных итераторов

Листинг 31. Применение итераторов вставки:

```
int main() {
     list<int> foo, bar;
     for (int i = 1; i <= 5; i++)
     {
          foo.push_back(i);
          bar.push_back(i * 10);
     }
     Print("foo= ", foo); // foo=  1 2 3 4 5
     Print("bar= ", bar); // bar=  10 20 30 40 50
     list<int>::iterator it = foo.begin();
     advance(it, 3); // *it =  4
     insert_iterator<list<int>> insert_it(foo, it); //
вставляет новый элемент в
                    //контейнере в положении, на который
указывает итератор
     copy(bar.begin(), bar.end(), insert_it);
     Print("foo= ", foo); // foo=  1 2 3 10 20 30 40 50 4
5
     front_insert_iterator<list<int> > front_it(foo); //
вставляет новый элемент в
                    // контейнере в начало цепочки в
обратном порядке
     copy(bar.cbegin(), bar.cend(), front_it);
     Print("foo= ", foo); // foo=  50 40 30 20 10 1 2 3 10
20 30 40 50 4 5
     back_insert_iterator<list<int>> back_it(foo); //
вставляет новый элемент в
                    // контейнере в конец  цепочки
     copy(bar.begin(), bar.end(), back_it);
     Print("foo= ", foo);
//foo= 50 40 30 20 10 1 2 3 10 20 30 40 50 4 5 10 20 30 40
50
     return 0;
}
```

Листинг 32. Использование итераторов потока ввода и вывода:

```
int main()
{
```

```
     istream_iterator<int> is(cin);
     ostream_iterator<int> os(cout, "\n");
     int input;
     while ((input = *is) != 0)
     {
          *os++ = input; is++;
     }
     return 0;
}
```

Листинг 33. Использование константных итераторов:

```
int main(){
     list<int> coll = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
     list<int>::const_iterator pos;
     pos = find(coll.cbegin(), coll.cend(), 5);
     cout << "pos: " << *pos << endl; //5
     list<int>::const_reverse_iterator rpos(pos);
     cout << "rpos: " << *rpos << endl; //4
     return 0;
}
```

Листинг 34. Применение итераторов перемещения:

```
int main() {
     vector<string> foo(3);
     vector<string> bar{ "one", "two", "three" };
     typedef vector<string>::iterator Iter;
     copy(move_iterator<Iter>(bar.begin()),
          move_iterator<Iter>(bar.end()),
          foo.begin());
     Print("bar = ", bar); // bar =
     Print("foo = ", foo); //foo = one two three
     return 0;
}
```

2.5. Алгоритмы

2.5.1. Основные понятия и определения

В *STL* реализовано большое количество стандартных алгоритмов, предназначенных для различной обработки элементов коллекций: поиска, сортировки, копирования, переупорядочения, модификации, численных расчетов.

Алгоритмы не являются методами контейнерных классов; это – внешние (глобальные) шаблонные функции, работающие с итераторами. У такого подхода есть важное достоинство – вместо того, чтобы реализовывать каждый алгоритм для каждого типа

контейнера, достаточно реализовать его один раз для обобщенного типа контейнера.

Для использования алгоритмов в некоторой программе к ней необходимо подключить заголовочный файл <*algorithm*>. Может также потребоваться подключение файлов <*functional*> и <*numeric*>.

Любая функция работает с одним или несколькими интервалами. В частном случае интервал может содержать и весь контейнер. Как правило, интервал задается для функции двумя аргументами – итераторами (начало и конец интервала). Корректность задания этих границ интервала должна обеспечивать вызывающая сторона (клиент).

Возможные ошибки (итераторы относятся к разным контейнерам, итератор начала находится после итератора конца и пр.) могут привести к непредсказуемым последствиям (например, к зацикливанию или нарушению защиты памяти).

Все функции работают с полуоткрытыми интервалами: интервал включает заданную начальную позицию, но конечная позиция в него не включается.

Некоторые функции требуют задания сразу несколько интервалов (двух или трех). В таких случаях обычно необходимо задать начало и конец только первого интервала, а для других – задается только их начало. При этом конечная позиция второго и третьего интервалов определяется по количеству элементов в первом интервале.

Листинг 35. Задание несколько интервалов:

```
int main() {
     int first[] {10, 40, 90, 40, 10};
     int second[] {1, 2, 3, 4, 5};
     int results[5];
     transform(first, first + 5, second, results,
divides<int>());

     for (auto x : results) cout << x << ' '; // 10 20 30
10 2
     cout << '\n';
     return 0;
}
```

Некоторые функции из раздела библиотеки *algorithm* допускают использование внешних функций, которые вызываются в процессе их внутренней работы.

Такая возможность делает эти функции еще более гибкими и мощными. Например, функция *accumulate()* по умолчанию вычисляет сумму элементов, но путем задания внешней функции ее можно «перепрограммировать» на любую другую обработку элементов контейнера.

Листинг 36. Использование внешних функций:

```
int myfunction(int x, int y) { return x + 2 * y; }
struct myclass {
     int k;
     myclass(int k_) : k(k_) {}
     int operator()(int x, int y) { return x + k*y; }
};
int main() {
     int init = 100;
     int numbers[] {10, 20, 30};
     int m = 4;
     cout << accumulate(numbers, numbers + 3, init) <<
endl; // 160
     cout << accumulate(numbers, numbers + 3, 1,
multiplies<int>()) << endl; // 6000
     cout << accumulate(numbers, numbers + 3, init,
myfunction) << endl; // 220
     cout << accumulate(numbers, numbers + 3, init,
myclass(3)) << endl; // 280
     cout << accumulate(numbers, numbers + 3, init,
          [m](int x, int y) {return x + m*y; }) << endl;
// 340
     return 0;
}
```

Особую разновидность внешних функций, используемых для работы библиотечных функций, составляют **предикаты** – функции, возвращающие логическое значение (*bool*). С их помощью часто определяют критерии сортировки и поиска. В зависимости от количества используемых аргументов предикаты делятся на унарные и бинарные.

Унарные предикаты проверяют некоторое свойство единственного своего аргумента, например, является ли число простым. Кроме того, унарный предикат может использоваться как критерий, определяющий, к каким элементам контейнера должна применяться операция.

Бинарные предикаты обычно сравнивают некоторое свойство двух аргументов. Часто такая возможность используется для выполнения сортировки, где требуется

проверять, в каком соотношении находятся два соседних элемента контейнера.

Листинг 37. Использование унарных предикатов:

```
bool isPrime(int n) {
     n = abs(n);
     if (n == 0 || n == 1) return false;
     int d;
     for (d = n / 2; n%d != 0; --d) {}
     return d == 1;
}

int main() {
     list<int> col1;
     for (int i = 24; i <= 30; ++i) col1.push_back(i);
     auto pos = find_if(col1.begin(), col1.end(),
isPrime);
     if (pos != col1.end())
          cout << "первое найденное простое число - " <<
*pos <<  endl; // 29
     else cout << "простых чисел не найдено" << endl;
}
```

Стандартная библиотека C++ содержит много шаблонных классов, собранных в заголовочном файле <*functional*>, и позволяющих создавать полезные функциональные объекты. Например, практически для всех унарных и бинарных операций языка C++ определены соответствующие шаблонные классы (табл. 2.19).

Таблица 2.19

Операций	**Шаблонные классы**	**Результат**
1	2	3
Арифметические	*plus<type>()*	paraml + param2
	minus<type>()	paraml - param2
	multiplies<type>()	paraml * param2
	divides<type>()	paraml / param2
	modulus<type>()	paraml % param2
	negate<type>()	-param

1	2	3
Сравнения	*equal_to<type>()*	paraml == param2
	not_equal_to<type>()	paraml != param2
	greater<type>()	paraml > param2
	greater_equal<type>()	paraml >= param2
	less<type>()	paraml < param2
	less_equal<type>()	paraml <= param2
Логические	*logical_and<type>()*	paraml && param2
	logical_not<type>()	! param
	logical_or<type>()	paraml || param2
Битовые	*bit_and<type>()*	paraml & param2
	bit_or<type>()	paraml | param2
	bit_xor<type>()	paraml ^ param2

Листинг 38. Использование шаблонного класса *greater<type>()*:

```
int main() {
     int numbers[] {20, 40, 50, 10, 30};
     sort(numbers, numbers + 5, greater<int>());
     cout << "numbers = ";
     for (auto x : numbers) cout << x << ' ';
// numbers = 50 40 30 20 10
     cout << '\n';
     return 0;
}
```

Примечание: здесь *greater<int>()* – это вызов конструктора шаблонного класса, который создает экземпляр класса, являющийся функциональным объектом, который передается в функцию *sort*.

2.5.2. Классификация библиотечных функций

Разные функции предназначены для решения разных задач, поэтому их можно классифицировать по основным областям применения. Например, одни функции только читают данные, другие модифицируют их, а третьи изменяют порядок следования элементов в контейнере.

Одна из возможных классификаций библиотечных функций следующая:

1. Функции, не модифицирующие элементы контейнера
2. Модифицирующие функции
3. Функции разбиения
4. Функции сортировки
5. Функции двоичного поиска (применяются для секционированных/ отсортированных интервалов)
6. Функции слияния отсортированных интервалов
7. Функции работы с пирамидой («кучей»)
8. Функции поиска минимальных / максимальных элементов
9. Прочие функции
10. Функции для численных расчетов (заголовочный файл <*numeric*>)

2.5.3. Функции, не модифицирующие элементы контейнера

Эти функции сохраняют как порядок следования обрабатываемых элементов, так и их значения. Могут вызываться для всех стандартных контейнеров (табл. 2.20).

Таблица 2.20

Функции	**Описание**
1	2
all_of()	возвращает *true*, если условие выполняется для всех элементов
any_of()	возвращает *true*, если условие выполняется хоты бы для одного элемента
none_of()	возвращает *true*, если условие не выполняется ни для одного элемента
for_each()	выполняет операцию с каждым элементом (!)
find()	ищет первый элемент с заданным значением
find_if()	ищет первый элемент, удовлетворяющий критерию
find_if_not()	ищет первый элемент, не удовлетворяющий заданному критерию

1	2
find_end()	ищет последнее вхождение подинтервала
find_first_of()	ищет первый из нескольких возможных элементов
adjacent_find()	ищет два смежных элемента, удовлетворяющих заданному критерию
count()	возвращает количество элементов
count_if()	возвращает количество элементов, удовлетворяющих заданному критерию
mismatch()	возвращает первый различающийся элемент в двух интервалах
equal()	проверяет, равны ли два интервала
is_permutation()	возвращает *true*, если два интервала содержат одни и те же элементы (возможно, в другом порядке)
search()	ищет первое вхождение подинтервала
search_n()	ищет первые n последовательных элементов с заданными свойствами

Листинг 39. Пример использования функции *all_of()*:

```
int main() {
     list<int> f(8);
     iota(f.begin(), f.end(), 2);
     Print("f = ", f); // f = 2 3 4 5 6 7 8 9
     for_each(f.begin(), f.end(), [](int& k) {return k *=
k; });
     Print("f = ", f); // f = 4 9 16 25 36 49 64 81
     if (all_of(f.begin(), f.end(), [](int i){return i %
2; }))// false
          cout << "1. Все элементы - нечетные числа.\n";
     if (any_of(f.begin(), f.end(), [](int i){return i % 3
== 0; })) // true
          cout << "2. Некоторые элементы делятся на 3.\n";
     if (none_of(f.begin(), f.end(), [](int i){return
i>50; }))// false
          cout << "3. Нет элементов больше 50.\n";
     return 0;
}
```

Примечание: функция *for_each()* выполняет операцию, которая передается ей в качестве аргумента, с каждым элементом интервала. Это позволяет использовать эту функцию и в качестве модифицирующей, если передать аргумент операции по ссылке.

2.5.4. Модифицирующие функции

Эти функции изменяют значения элементов контейнера. Модификация производится непосредственно внутри интервала или в процессе копирования в другой интервал (в этом случае исходный интервал остается без изменений) (табл. 2.21).

Таблица 2.21

Функции	**Описание**
1	2
copy()	копирует входной интервал в выходной, начиная с первого элемента
copy_n()	копирует первые n элементов из входного интервала в выходной
copy_if()	копирует элементы, удовлетворяющие заданному условию, из входного интервала в выходной
copy_backward()	копирует входной интервал в выходной, начиная с последнего элемента
move()	перемещает входной интервал в выходной, начиная с первого элемента
move_backward()	перемещает входной интервал в выходной, начиная с последнего элемента
swap()	обменивает значения двух контейнеров
swap_ranges()	обменивает значения двух интервалов
iter_swap()	обменивает значения двух элементов, заданных итераторами
transform()	модифицирует (и копирует) элементы; объединяет элементы двух интервалов
replace()	заменяет элементы с заданным значением другим значением
replace_if()	заменяет элементы, соответствующие критерию, заданным значением
replace_copy()	заменяет элементы с заданным значением при копировании интервала

1	2
replace_copy_if()	заменяет элементы, соответствующие критерию, при копировании интервала
fill()	заменяет каждый элемент заданным значением
fill_n()	заменяет n элементов заданным значением
generate()	заменяет каждый элемент результатом операции
generate_n()	заменяет n элементов результатом операции
remove()	удаляет элементы с заданным значением
remove_if()	производит удаляет элементы по заданному критерию
remove_copy()	копирует элементы, значение которых отлично от заданного
remove_copy_if()	копирует элементы, не соответствующие заданному критерию
unique()	удаляет смежные дубликаты (элементы, равные своему предшественнику)
unique_copy()	копирует элементы с удалением смежных дубликатов
reverse()	переставляет элементы в обратном порядке
reverse_copy()	копирует элементы, переставленные в обратном порядке
rotate()	производит циклический сдвиг элементов
rotate_copy()	копирует элементы с циклическим сдвигом
random_shuffle()	переставляет элементы в случайном порядке
shuffle()	переставляет элементы в случайном порядке, который задается одним из стандартных генераторов случайных чисел

Листинг 40. Пример использования функции *generate()*:

```
struct c_unique {
     int current{ 0 };
     int operator()() { return ++current; }
} UniqueNumber;
int main() {
     srand(unsigned(time(0)));
     vector<int> myvector(8);
     generate(myvector.begin(), myvector.end(),
```

```
          [](){ return (rand() % 100); });
     Print("myvector : ", myvector); // случайные значения
при каждом запуске
     generate(myvector.begin(), myvector.end(),
UniqueNumber);
     Print("myvector : ", myvector); // myvector : 1 2 3 4
5 6 7 8
     return 0;
}
```

2.5.5. Функции разбиения

Эти функции изменяют порядок элементов (но не их значения), выполняя разбиение интервала в соответствии с критерием (табл. 2.22).

Таблица 2.22

Функции	Описание
is_partitioned()	проверяет, удовлетворяет ли интервал критерию разбиения
partition()	изменяет порядок следования элементов так, что элементы, удовлетворяющие критерию разбиения, оказываются спереди; результат – итератор, указывающий на границу разбиения
stable_partition()	то же, что и *partition()*, но с сохранением относительного расположения элементов, соответствующих и не соответствующих критерию
partition_copy()	выполняет разбиение интервала на два с сохранением результата в двух новых интервалах
partition_point()	возвращает итератор, указывающий на место разбиения

Листинг 41. Пример использования функций разбиения:

```
int main() {
     vector<int> v{ 3, 7, 11, 5, 6, 4, 9, 8, 1, 10, 0, 12
}, v1(v);
     Print("v : ", v);
     auto bound = partition(v.begin(), v.end(),
          [](int i) { return (i % 2) == 1; });
     cout << "partition\nodd elements:";
     for (auto it = v.begin(); it != bound;) cout << ' '
<< *(it++);//3 7 11 5 1 9
     cout << '\n';
     cout << "even elements:";
     for (auto it = bound; it != v.end();) cout << ' ' <<
*(it++); //4 8 6 10 0 12
     cout << '\n';
```

```
    bound = stable_partition(v1.begin(), v1.end(),
        [](int i) { return (i % 2) == 1; });
    cout << "\nstable_partition\nodd elements:";
    for (auto it = v1.begin(); it != bound;) cout << ' '
<< *(it++);//3 7 11 5 9 1
    cout << '\n';
    cout << "even elements:";
    for (auto it = bound; it != v1.end();) cout << ' ' <<
*(it++);//6 4 8 10 0 12
    cout << '\n';
    return 0;
}
```

2.5.6. Функции сортировки

Функции сортировки изменяют порядок следования элементов, упорядочивая их, и представляют собой специальную разновидность модифицирующих алгоритмов. Сложность этих алгоритмов больше линейной и требует использования итераторов произвольного доступа (табл. 2.23).

Таблица 2.23

Функции	**Описание**
sort()	сортирует все элементы интервала
stable_sort()	сортирует все элементы интервала с сохранением порядка следования равных элементов
partial_sort()	сортирует начальную часть интервала
partial_sort_copy()	помещает в выходной интервал отсортированную часть входного интервала
is_sorted()	возвращает *true*, если интервал отсортирован
is_sorted_until()	возвращает итератор на первый по порядку элемент, нарушающий порядок сортировки интервала
nth_element()	перестраивает элементы интервала таким образом, чтобы слева от элемента, расположенного в позиции *n*, оказались все элементы, меньшие его, а справа – большие

Листинг 42. Пример использования функций сортировки:

```
int main() {
    vector<int> v{ 3, 7, 11, 5, 6, 4, 9, 8, 1, 10, 0, 12
}, v1(v), r(6);
    Print("v : ", v);
    partial_sort(v.begin(), v.begin() + 5, v.end());
```

```
    Print("v : ", v); //v: 0 1 3 4 5 11 9 8 7 10 6 12
    partial_sort_copy(v1.begin(), v1.end(), r.begin(),
r.end());
    Print("r : ", r); //r: 0 1 3 4 5 6
    random_shuffle(v1.begin(), v1.end());
    nth_element(v1.begin(), v1.begin() + 6, v1.end());
    Print("v1 : ", v1); //v1: 1 0 5 4 3 6 7 8 12 9 11 10
    return 0;
}
```

2.5.7. Функции двоичного поиска

Эти функции требуют, чтобы интервалы, с которыми они работают, были предварительно упорядочены по соответствующему критерию. Достоинством этих функций является их невысокая сложность (табл. 2.24).

Таблица 2.24

Функции	**Описание**
lower_bound()	находит первый элемент со значением, большим либо равным заданному
upper_bound()	находит первый элемент со значением, большим заданному
equal_range()	возвращает количество элементов в интервале, равных заданному значению
binary_search()	проверяет, содержит ли интервал заданный элемент

Примечание: функции *lower_bound() / upper_bound()* удобны для поиска в упорядоченной последовательности границ полуоткрытого интервала, заполненного заданным значением. Они также имеют вторую перегрузку, позволяющую задавать внешнюю функцию в качестве сравнивающего выражения.

Листинг 43. Пример использования функций двоичного поиска:

```
int main() {
    vector<int> v{ 10, 20, 30, 30, 20, 10, 10, 20 };
    sort(v.begin(), v.end());
    Print("v : ", v); //v: 10 10 10 20 20 20 30 30
    auto low = lower_bound(v.begin(), v.end(), 20);
    auto up = upper_bound(v.begin(), v.end(), 20);
    cout << "lower_bound at position " << (low -
v.begin()) << '\n'; //3
```

```
     cout << "upper_bound at position " << (up -
v.begin()) << '\n'; //6
     return 0;
}
```

2.5.8. Функции слияния отсортированных интервалов

Эти функции изменяют порядок элементов (но не их значения), выполняя слияние двух интервалов (табл. 2.25).

Таблица 2.25

Функции	**Описание**
merge()	выполняет слияние двух интервалов
inplace_merge()	выполняет слияние двух упорядоченных интервалов, размещенных последовательно в одном контейнере
includes()	проверяет, что каждый элемент интервала также является элементом другого интервала
set_union()	вычисляет упорядоченное объединение двух интервалов
set_intersection()	вычисляет упорядоченное пересечение двух интервалов
set_difference()	вычисляет упорядоченный интервал, который содержит все элементы интервала, не входящие в другой интервал
set_symmetric_difference()	вычисляет упорядоченный интервал, который содержит все элементы интервала, входящие только в один из двух интервалов

Листинг 44. Пример использования функции *merge()*:

```
int main() {
     int first[] {5, 10, 15, 20, 25};
     int second[] {50, 40, 30, 20, 10};
     vector<int> v(10);
     sort(first, first + 5);
     sort(second, second + 5);
     merge(first, first + 5, second, second + 5,
v.begin());
     Print("Result : ", v); // Result: 5 10 10 15 20 20 25
30 40 50
     return 0;
}
```

Листинг 45. Пример использования функции *inplace_merge()*:

```
int main() {
     int first[] {5, 10, 15, 20, 25};
     int second[] {50, 40, 30, 20, 10};
     vector<int> v(10);
     sort(first, first + 5);
     sort(second, second + 5);
     auto it = copy(first, first + 5, v.begin());
     copy(second, second + 5, it);
     Print("Before : ", v); // Before: 5 10 15 20 25 10 20
30 40 50
     inplace_merge(v.begin(), it, v.end());
     Print("Result : ", v); // Result: 5 10 10 15 20 20 25
30 40 50
     return 0;
}
```

2.5.9. Функции работы с пирамидой («кучей»)

В табл. 2.26 перечислены функции работы с пирамидой.

Таблица 2.26

Функции	Описание
push_heap()	добавляет элемент в пирамиду
pop_heap()	удаляет элемент из пирамиды
make_heap()	преобразует интервал в пирамиду
sort_heap()	сортирует пирамиду (которая после вызова перестает быть пирамидой)
is_heap()	возвращает *true*, если интервал является пирамидой
is_heap_until()	возвращает итератор, указывающий на первый элемент интервала, который не удовлетворяет условию упорядочения пирамиды

Листинг 46. Пример использования функций работы с пирамидой:

```
int main() {
     vector <int> v{ 16, 14, 9, 8, 7, 10, 3, 1, 4, 2 };
     Print("Вектор v : ", v); // Вектор v: 16 14 9 8 7 10
3 1 4 2
     if (is_heap(v.begin(), v.end())) cout << "Это
пирамида\n";
     else cout << "Это не пирамида\n";// true
```

```
     auto last = is_heap_until(v.begin(), v.end());
     cout << "Первый элемент-нарушитель : " << *last <<
endl; ";//10
     // Преобразуем v в кучу с упорядочением по умолчанию
("<")
     //(неубывающая пирамида)
     make_heap(v.begin(), v.end());
     Print("v как неубывающая пирамида : ", v); //16 14 10
8 7 9 3 1 4 2
     // Преобразуем v в кучу с упорядочением ">"
     //(невозрастающая пирамида)
     make_heap(v.begin(), v.end(), greater<int>());
     Print("v как невозрастающая пирамида : ", v); //1 2 3
4 7 9 10 8 14 16
     make_heap(v.begin(), v.end());
     Print("v как неубывающая пирамида : ", v);  //16 14
10 8 7 9 3 1 4 2
     v.push_back(13);
     Print("Добавили элемент в \"хвост\" v : ", v); //16
14 10 8 7 9 3 1 4 2 13
     push_heap(v.begin(), v.end());
     Print("Восстановили свойство пирамиды v : ", v); //16
14 10 8 13 9 3 1 4 2 7
     pop_heap(v.begin(), v.end());
     Print("Удалили вершину пирамиды v : ", v); //14 13 10
8 7 9 3 1 4 2 16
     v.pop_back();
     Print("Выбросили последний элемент v : ", v); //14 13
10 8 7 9 3 1 4
     sort_heap(v.begin(), v.end());//1 2 3 4 7 8 9 10 13
14
     Print("Отсортированный v : ", v);
     if (is_heap(v.begin(), v.end()))
          cout << "Это пирамида\n";
     else cout << "Это не пирамида\n"; // true
}
```

2.5.10. Функции поиска минимальных / максимальных элементов

Эти функции позволяют выполнять поиск максимального (минимального) элемента двух и более элементов (табл. 2.27).

Таблица 2.27

Функции	**Описание**
1	2
min()	возвращает минимальный элемент (из двух или из списка инициализации)

1	2
max()	возвращает максимальный элемент (из двух или из списка инициализации)
minmax()	возвращает пару: *first* – минимальный, *second* –максимальный
min_element()	возвращает итератор, указывающий на элемент интервала с минимальным значением
max_element()	возвращает итератор, указывающий на элемент интервала с максимальным значением
minmax_element()	возвращает первый минимальный, но последний максимальный элемент

Листинг 47. Пример использования функции поиска минимального/максимального элементов:

```
int main() {
     cout << "min(1,2)==" << min(1, 2) << '\n'; //1
     cout << "min(2,1)==" << min(2, 1) << '\n'; //1
     cout << "min(2,1)==" << min(2, 1,
          [](int a, int b){return a > b; }) << '\n'; //2
     cout << "min('a','z')==" << min('a', 'z') << '\n';
//a
     cout << "min(3.14,2.72)==" << min(3.14, 2.72) <<
'\n'; //2.72
     cout << "min({5,4,7,2})==" << min({ 5, 4, 7, 2 }) <<
'\n'; //2

     vector<int> v{ 3, 7, 2, 5, 6, 4, 9 };
     cout << "min:" << *min_element(v.begin(), v.end()) <<
'\n'; //2
     cout << "max: " << *max_element(v.begin(), v.end())
<< '\n'; //9
     auto result = minmax_element(v.begin(), v.end());
     cout << "min is " << *result.first; //2
     cout << ", at position " << (result.first -
v.begin()) << '\n'; //2
     cout << "max is " << *result.second; //9
     cout << ", at position " << (result.second -
v.begin()) << '\n'; //6
     return 0;
}
```

2.5.11. Прочие функции

В табл. 2.28 перечислены дополнительные функции стандартной библиотеки шаблонов.

Таблица 2.28

Функции	Описание
lexicographical_compare()	проверяет, что один интервал меньше другого по лексикографическому критерию
next_permutation()	переставляет элементы
prev_permutation()	переставляет элементы

Листинг 48. Пример использования лексикографического сравнения:

```
int main() {
     char s1[] {"Apple"}; char s2[] {"apartment"};
     cout << boolalpha;
//По умолчанию используется оператор <
     cout << lexicographical_compare(s1, s1 + 5, s2, s2 +
9); // true
     cout << '\n';
//Использование лямбда в качестве объекта сравнения
     cout << lexicographical_compare(s1, s1 + 5, s2, s2 +
9,        [](char c1, char c2) { return
tolower(c1)<tolower(c2); }); //false
     cout << '\n';
     return 0;
}
```

2.5.12. Функции для численных расчетов

Эти функции выполняют разнообразную обработку числовых элементов и отличаются очень большой гибкостью и мощью (табл. 2.29). Например, функция *accumulate()* по умолчанию вычисляет сумму элементов. Однако если элементами контейнера являются строки, то вычисляется их конкатенация. А если переключиться с оператора + на оператор *, то функция вычислит произведение элементов. Для использования этих функций необходимо подключить заголовочный файл <*numeric*>.

Таблица 2.29

Функции	Описание
1	2
accumulate()	объединяет все значения элементов (вычисляет сумму, произведение и т.д.)

inner_product()	объединяет все элементы двух интервалов
adjanced_difference()	для каждого элемента входного интервала выполняется некоторая операция (по умолчанию – вычитание) с его предшественником, а результат сохраняется в выходном интервале
partial_sum()	объединяет каждый элемент со своими предшественниками
iota()	заполняет интервал значениями по возрастанию (от заданного начального значения)

Листинг 48. Пример использования функции *iota()*:

```
 int main() {
     vector<int> numbers(10);

     iota(numbers.begin(), numbers.end(), 100);
     Print("numbers =", numbers);
// numbers = 100 101 102 103 104 105 106 107 108 109
     return 0;
}
```

Листинг 49. Пример использования функции *partial_sum()*:

```
int myop (int x, int y) {return x+y+1;}

int main () {
  vector <int> val= {1,2,3,4,5};
  vector <int> result(5);

   partial_sum (val.begin(), val.end(), result.begin());
   Print("result = ",result); // result = 1 3 6 10 15

  partial_sum (val.begin(), val.end(), result.begin(),
multiplies<int>());
  Print("result = ",result) ; // result = 1 2 6 24 120

   partial_sum (val.begin(), val.end(), result.begin(),
myop);
  Print("result = ",result) ; // result = 1 4 8 13 19
  return 0;
}
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Уилкс, М. Составление программ для электронных счетных машин / М.Уилкс, Д.Уилер, С.Гилл. - М.: Иностр. лит., 1953. - 208 с.
2. Жоголев, Е.А. Курс программирования / Е.А.Жоголев, Н.П.Трифонов. - М.: Наука, 1971 – 400 с.
3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон Дж., Влиссидес. СПб: Питер, 2001. — 368 с.
4. Джосаттис, Н.М. Стандартная библиотека C++. Справочное руководство, 2 изд.: Пер. с англ. – М.: Издат. дом «Вильямс», 2014. – 1136 с., ил.
5. Аммерааль Л. STL для программистов на C++.: Пер. с англ. – М.: ДМК, 1999. – 240 с., ил.
6. Страуструп, Б. Язык программирования C++, 2 доп. изд.: Пер. с англ. – М.: «Бином», 2011. – 1136 с., ил.
7. Николас А. Солтер, Скотт Дж. Клепер - C++ для профессионалов.: Пер. с англ. – М.: Издат. дом «Вильямс», 2006. – 912 с., ил.
8. Кнут, Д. Искусство программирования. Т1. Основные алгоритмы, 3 изд.: Пер. с англ. – М.: Издат. дом «Вильямс», 2000. – 736 с., ил.
9. Library (computing) [Электронный ресурс]: Режим доступа: https://en.wikipedia.org/wiki/Library_(computing)
10. Code reuse [Электронный ресурс]: Режим доступа: https://en.wikipedia.org/wiki/Code_reuse
11. C reference [Электронный ресурс]: Режим доступа: http://en.cppreference.com/w/c

Учебное издание

Иванченко Александр Николаевич
Шайда Алексей Юрьевич

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК

Учебное пособие

Редактор *Н.А. Юшко*

Подписано в печать 21.09.2016. Формат 60x84 1/16.
Бумага офсетная. Печать цифровая. Усл. печ. л. 5,11.
Уч.-изд.л. 5,00. Тираж 100 экз. Заказ _____.

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова

ВВЕДЕНИЕ В КОМПЬЮТЕРНЫЕ НАУКИ

Методические указания к выполнению лабораторных и самостоятельных работ

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 004(076.5)
ББК З 973.2-018

Рецензент – канд. техн. наук, доц., доцент кафедры «Информационные и измерительные системы и технологии»
Скоба Андрей Николаевич

Мохов В.А., Кущий Д.Н.

Введение в компьютерные науки: методические указания к выполнению лабораторных и самостоятельных работ / Южно-Российский государственный политехнический университет (НПИ) имени М. И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 28 с.

Издание содержит методические материалы по освоению навыков работы с числами в различных позиционных системах счисления, представления данных в памяти ЭВМ, разработки алгоритмов и моделей предметной области, создания реляционных баз данных и применения стандартных программ операционной системы Windows. Приведена программа лабораторных работ, варианты заданий, а также контрольные вопросы для проверки знаний.

Методические указания предназначены для студентов, обучающихся по программам бакалавриата следующих направлений подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия.

УДК 004(076.5)

1. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ, ИХ НАИМЕНОВАНИЕ И ОБЪЕМ В ЧАСАХ

1.1. Очная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	Перевод чисел из одной позиционной системы счисления в другую. Арифметические операции в различных системах счисления	2	Защита отчета	10-15.10
2	Представление данных в памяти ЭВМ	2	Защита отчета	10-15.10
3	*Разработка блок-схем алгоритмов. Знакомство с MS Visio*	2	Защита отчета	10-15.10
4	*Построение моделей предметной области с использованием case-средства BPwin*	4	Защита отчета	15-20.11
5	*Знакомство с файл-серверной СУБД MS Access*	4	Защита отчета	15-20.11
6	*Использование стандартных программ операционной системы Windows*	2	Защита отчета	15-20.12
7	Создание блога. Выбор облачного хранилища	2	Защита отчета	15-20.12

Лабораторная работа №1

Перевод чисел из одной позиционной системы счисления в другую. Арифметические операции в различных системах счисления

Цель работы: освоение навыков работы с системами счисления, используемыми в компьютере: двоичной, восьмеричной, шестнадцатеричной, изучение принципов представления чисел в памяти ЭВМ.

Программа работы

1. Перевести данное число из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную системы счисления.
2. Перевести данное число в десятичную систему счисления.
3. Сложить числа.
4. Выполнить вычитание.
5. Выполнить умножение.

Примечание. В заданиях 1-2 необходимо привести полную процедуру перевода чисел. В заданиях 3–5 необходимо проверить правильность вычислений переводом исходных данных и результатов в десятичную систему счисления.

Варианты заданий

Вариант 1	1. а) $666_{(10)}$; б) $305_{(10)}$; в) $153{,}25_{(10)}$; г) $162{,}25_{(10)}$; д) $248{,}46_{(10)}$ 2. а) $1100111011_{(2)}$; б) $10000000111_{(2)}$; в) $10110101{,}1_{(2)}$; г) $100000110{,}10101_{(2)}$; д) $671{,}24_{(8)}$; е) $41A{,}6_{(16)}$. 3. а) $10000011_{(2)}+1000011_{(2)}$; б) $110010{,}101_{(2)}+1011010011{,}01_{(2)}$; в) $356{,}5_{(8)}+1757{,}04_{(8)}$; г) $293{,}8_{(16)}+3CC{,}98_{(16)}$. 4. а) $100111001_{(2)}-110110_{(2)}$; б) $1101111011{,}01_{(2)}-101000010{,}0111_{(2)}$; в) $2025{,}2_{(8)}-131{,}2_{(8)}$; г) $2D8{,}4_{(16)}-A3{,}B_{(16)}$. 5. а) $1100110_{(2)}\times 1011010_{(2)}$; б) $2001{,}6_{(8)}\times 125{,}2_{(8)}$; в) $2C{,}4_{(16)}\times 12{,}98_{(16)}$.
Вариант 2	1. а) $164_{(10)}$; б) $255_{(10)}$; в) $712{,}25_{(10)}$; г) $670{,}25_{(10)}$; д) $11{,}89_{(10)}$ 2. а) $1001110011_{(2)}$; б) $1001000_{(2)}$; в) $1111100111{,}01_{(2)}$; г) $1010001100{,}101101_{(2)}$; д) $413{,}41_{(8)}$; е) $118{,}8C_{(16)}$. 3. а) $1100001100_{(2)}+1100011001_{(2)}$; б) $111111111{,}001_{(2)}+1111111110{,}0101_{(2)}$; в) $1443{,}1_{(8)}+242{,}44_{(8)}$; г) $2B4{,}C_{(16)}+EA{,}4_{(16)}$. 4. а) $1001101100_{(2)}-1000010111_{(2)}$; б) $1101100110{,}01_{(2)}-111000010{,}1011_{(2)}$; в) $1567{,}3_{(8)}-1125{,}5_{(8)}$; г) $416{,}3_{(16)}-255{,}3_{(16)}$. 5. а) $100001_{(2)}\times 1001010_{(2)}$; б) $1723{,}2_{(8)}\times 15{,}2_{(8)}$; в) $54{,}3_{(16)}\times 9{,}6_{(16)}$.

Вариант 3	1. а) $273_{(10)}$; б) $661_{(10)}$; в) $156,25_{(10)}$; г) $797,5_{(10)}$; д) $53,74_{(10)}$ 2. а) $1100000000_{(2)}$; б) $1101011111_{(2)}$; в) $1011001101,00011_{(2)}$; г) $1011110100,011_{(2)}$; д) $1017,2_{(8)}$; е) $111,B_{(16)}$. 3. а) $1110001000_{(2)}+110100100_{(2)}$; б) $111100010,0101_{(2)}+1111111,01_{(2)}$; в) $573,04_{(8)}+1577,2_{(8)}$; г) $108,8_{(16)}+21B,9_{(16)}$. 4. а) $1010111001_{(2)}-1010001011_{(2)}$; б) $1110111000,011_{(2)}-111001101,001_{(2)}$; в) $1300,3_{(8)}-464,2_{(8)}$; г) $37C,4_{(16)}-1D0,2_{(16)}$. 5. а) $1011010_{(2)}\times 1000010_{(2)}$; б) $632,2_{(8)}\times 141,34_{(8)}$; в) $2A,7_{(16)}\times 18,8_{(16)}$.
Вариант 4	1. а) $105_{(10)}$; б) $358_{(10)}$; в) $377,5_{(10)}$; г) $247,25_{(10)}$; д) $87,27_{(10)}$ 2. а) $1100001001_{(2)}$; б) $1100100101_{(2)}$; в) $1111110110,01_{(2)}$; г) $11001100,011_{(2)}$; д) $112,04_{(8)}$; е) $334,A_{(16)}$. 3. а) $101000011_{(2)}+110101010_{(2)}$; б) $10011011,011_{(2)}+1111100001,0011_{(2}$ в) $1364,44_{(8)}+1040,2_{(8)}$; г) $158,A_{(16)}+34,C_{(16)}$. 4. а) $1111111000_{(2)}-100010011_{(2)}$; б) $1001100100,01_{(2)}-10101001,1_{(2)}$; в) $1405,3_{(8)}-346,5_{(8)}$; г) $3DD,4_{(16)}-303,A_{(16)}$. 5. а) $1011100_{(2)}\times 1100100_{(2)}$; б) $347,2_{(8)}\times 125,64_{(8)}$; в) $10,A8_{(16)}\times 35,4_{(16)}$.
Вариант 5	1. а) $500_{(10)}$; б) $675_{(10)}$; в) $810,25_{(10)}$; г) $1017,25_{(10)}$; д) $123,72_{(10)}$ 2. а) $1101010001_{(2)}$; б) $100011100_{(2)}$; в) $1101110001,011011_{(2)}$; г) $110011000,111001_{(2)}$; д) $1347,17_{(8)}$; е) $155,6C_{(16)}$. 3. а) $1000101101_{(2)}+1100000010_{(2)}$; б) $1001000011,1_{(2)}+10001101,101_{(2)}$; в) $415,24_{(8)}+1345,04_{(8)}$; г) $113,B_{(16)}+65,8_{(16)}$. 4. а) $1101111100_{(2)}-100100010_{(2)}$; б) $1111011110,1101_{(2)}-1001110111,1_{(2)}$; в) $1333,2_{(8)}-643,2_{(8)}$; г) $176,7_{(16)}-E5,4_{(16)}$. 5. а) $1101100_{(2)}\times 1010011_{(2)}$; б) $516,54_{(8)}\times 44,64_{(8)}$; в) $61,8_{(16)}\times 48,9_{(16)}$.
Вариант 6	1. а) $218_{(10)}$; б) $808_{(10)}$; в) $176,25_{(10)}$; г) $284,25_{(10)}$; д) $253,04_{(10)}$ 2. а) $111000100_{(2)}$; б) $1011001101_{(2)}$; в) $10110011,01_{(2)}$; г) $1010111111,011_{(2)}$; д) $1665,3_{(8)}$; е) $FA,7_{(16)}$. 3. а) $11100000_{(2)}+1100000000_{(2)}$; б) $10011011,011_{(2)}+1110110100,01_{(2)}$; в) $1041,2_{(8)}+1141,1_{(8)}$; г) $3C6,8_{(16)}+B7,5_{(16)}$. 4. а) $10110010_{(2)}-1010001_{(2)}$; б) $1100101111,1101_{(2)}-100111000,1_{(2)}$; в) $1621,44_{(8)}-1064,5_{(8)}$; г) $1AC,B_{(16)}-BD,7_{(16)}$. 5. а) $1000000_{(2)}\times 110110_{(2)}$; б) $714,34_{(8)}\times 133,4_{(8)}$; в) $16,B_{(16)}\times 2B,6_{(16)}$.

Вариант 7	1. а) $306_{(10)}$; б) $467_{(10)}$; в) $218,5_{(10)}$; г) $667,25_{(10)}$; д) $318,87_{(10)}$ 2. а) $1111000111_{(2)}$; б) $11010101_{(2)}$; в) $1001111010,010001_{(2)}$; г) $1000001111,01_{(2)}$; д) $465,3_{(8)}$; е) $252,38_{(16)}$. 3. а) $1000001101_{(2)}+1100101000_{(2)}$; б) $1100111,00101_{(2)}+101010110,011_{(2)}$; в) $520,4_{(8)}+635,4_{(8)}$; г) $2DB,6_{(16)}+15E,6_{(16)}$. 4. а) $1101000101_{(2)}-111111000_{(2)}$; б) $1011101011,001_{(2)}-1011001000,01001_{(2)}$; в) $1034,4_{(8)}-457,44_{(8)}$; г) $239,A_{(16)}-9C,4_{(16)}$. 5. а) $1101101_{(2)}\times 101010_{(2)}$; б) $310,2_{(8)}\times 40,5_{(8)}$; в) $18,4_{(16)}\times 35,4_{(16)}$.
Вариант 8	1. а) $167_{(10)}$; б) $113_{(10)}$; в) $607,5_{(10)}$; г) $828,25_{(10)}$; д) $314,71_{(10)}$ 2. а) $110010001_{(2)}$; б) $100100000_{(2)}$; в) $1110011100,111_{(2)}$; г) $1010111010,1110111_{(2)}$; д) $704,6_{(8)}$; е) $367,38_{(16)}$. 3. а) $10101100_{(2)}+111110010_{(2)}$; б) $1110111010,10011_{(2)}+1011010011,001_{(2)}$; в) $355,2_{(8)}+562,04_{(8)}$; г) $1E5,18_{(16)}+3BA,78_{(16)}$. 4. а) $1010110010_{(2)}-1000000000_{(2)}$; б) $1101001010,101_{(2)}-1100111000,011_{(2)}$; в) $1134,54_{(8)}-231,2_{(8)}$; г) $2DE,6_{(16)}-12A,4_{(16)}$. 5. а) $10101_{(2)}\times 11010_{(2)}$; б) $575,2_{(8)}\times 102,2_{(8)}$; в) $55,4_{(16)}\times 6,5_{(16)}$.
Вариант 9	1. а) $342_{(10)}$; б) $374_{(10)}$; в) $164,25_{(10)}$; г) $520,375_{(10)}$; д) $97,14_{(10)}$. 2. а) $1000110110_{(2)}$; б) $111100001_{(2)}$; в) $1110010100,1011001_{(2)}$; г) $1000000110,00101_{(2)}$; д) $666,16_{(8)}$; е) $1C7,68_{(16)}$. 3. а) $1101010000_{(2)}+1011101001_{(2)}$; б) $1100100001,0101_{(2)}+1110111111,011_{(2)}$; в) $242,2_{(8)}+1153,5_{(8)}$; г) $84,8_{(16)}+27E,8_{(16)}$. 4. а) $1111110_{(2)}-1111011_{(2)}$; б) $1111011111,1001_{(2)}-1010111100,01_{(2)}$; в) $1241,34_{(8)}-1124,3_{(8)}$; г) $15F,A_{(16)}-159,4_{(16)}$. 5. а) $1001010_{(2)}\times 1101111_{(2)}$; б) $1616,3_{(8)}\times 61,3_{(8)}$; в) $3A,38_{(16)}\times 64,4_{(16)}$.
Вариант 10	1. а) $524_{(10)}$; б) $222_{(10)}$; в) $579,5_{(10)}$; г) $847,625_{(10)}$; д) $53,35_{(10)}$. 2. а) $101111111_{(2)}$; б) $1111100110_{(2)}$; в) $10011000,1101011_{(2)}$; г) $1110001101,1001_{(2)}$; д) $140,22_{(8)}$; е) $1DE,54_{(16)}$. 3. а) $1101010000_{(2)}+11100100_{(2)}$; б) $1111100100,11_{(2)}+1111101000,01_{(2)}$; в) $1476,3_{(8)}+1011,1_{(8)}$; г) $3E0,A_{(16)}+135,8_{(16)}$. 4. а) $1010010100_{(2)}-11101110_{(2)}$; б) $1110100111,01_{(2)}-110000001,1_{(2)}$; в) $1542,5_{(8)}-353,24_{(8)}$; г) $3EB,8_{(16)}-3BA,8_{(16)}$. 5. а) $111000_{(2)}\times 100111_{(2)}$; б) $157,4_{(8)}\times 101,1_{(8)}$; в) $19,7_{(16)}\times 58,78_{(16)}$.

Лабораторная работа №2

Перевод чисел из одной позиционной системы счисления в другую. Арифметические операции в различных системах счисления

Цель работы: изучение принципов представления чисел и текстовых данных в памяти ЭВМ.

Программа работы

1. Перевести данное число из десятичной системы счисления в двоично-десятичную.
2. Перевести данное число из двоично-десятичной системы счисления в десятичную.
3. Зашифровать данный текст, используя таблицу ASCII-кодов.
4. Дешифровать данный текст, используя таблицу ASCII-кодов.
5. Записать прямой код числа, интерпретируя его как восьмибитовое целое без знака.
6. Записать дополнительный код числа, интерпретируя его как восьмибитовое целое со знаком.
7. Записать прямой код числа, интерпретируя его как шестнадцатибитовое целое без знака.
8. Записать дополнительный код числа, интерпретируя его как шестнадцатибитовое целое со знаком.
9. Записать в десятичной системе счисления целое число, если дан его дополнительный код.
10. Записать код действительного числа, интерпретируя его как величину типа ***Double***.
11. Дан код величины типа ***Double***. Преобразовать его в число.

Примечание. В задания 6, 8, 10, 11 необходимо привести подробную последовательность выполняемых действий.

Варианты заданий

Вариант 1	1. а) $585_{(10)}$; б) $673_{(10)}$; в) $626_{(10)}$. 2. а) $010101010101_{(2\text{-}10)}$; б) $10011000_{(2\text{-}10)}$; в) $010000010110_{(2\text{-}10)}$. 3. IBM PC. 4. 8A AE AC AF EC EE E2 A5 E0. 5. а) $224_{(10)}$; б) $253_{(10)}$; в) $226_{(10)}$. 6. а) $115_{(10)}$; б) $-34_{(10)}$; в) $-70_{(10)}$. 7. а) $22491_{(10)}$; б) $23832_{(10)}$. 8. а) $20850_{(10)}$; б) $-18641_{(10)}$. 9. а) 0011010111010110; б) 1000000110101110. 10. а) –578,375; б) –786,375. 11. а) 408E130000000000; б) C077880000000000.
Вариант 2	1. а) $285_{(10)}$; б) $846_{(10)}$; в) $163_{(10)}$. 2. а) $000101010001_{(2\text{-}10)}$; б) $010101010011_{(2\text{-}10)}$; в) $011010001000_{(2\text{-}10)}$. 3. Автоматизация. 4. 50 72 6F 67 72 61 6D. 5. а) $242_{(10)}$; б) $135_{(10)}$; в) $248_{(10)}$. 6. а) $81_{(10)}$; б) $-40_{(10)}$; в) $-24_{(10)}$. 7. а) $18509_{(10)}$; б) $28180_{(10)}$. 8. а) $28882_{(10)}$; б) $-19070_{(10)}$. 9. а) 0110010010010101; б) 1000011111110001. 10. а) –363,15625; б) –487,15625. 11. а) C075228000000000; б) 408B9B0000000000.
Вариант 3	1. а) $905_{(10)}$; б) $504_{(10)}$; в) $515_{(10)}$. 2. а) $010010010100_{(2\text{-}10)}$; б) $001000000100_{(2\text{-}10)}$; в) $01110000_{(2\text{-}10)}$. 3. Информатика. 4. 50 72 6F 63 65 64 75 72 65. 5. а) $207_{(10)}$; б) $210_{(10)}$; в) $226_{(10)}$. 6. а) $98_{(10)}$; б) $-111_{(10)}$; в) $-95_{(10)}$. 7. а) $19835_{(10)}$; б) $22248_{(10)}$. 8. а) $18156_{(10)}$; б) $-28844_{(10)}$. 9. а) 0111100011001000; б) 1111011101101101. 10. а) 334,15625; б) 367,15625. 11. а) C07C08C000000000; б) C0811B0000000000.

Вариант 4	1. а) $483_{(10)}$; б) $412_{(10)}$; в) $738_{(10)}$. 2. а) $001101011000_{(2\text{-}10)}$; б) $100010010010_{(2\text{-}10)}$; в) $010101000110_{(2\text{-}10)}$. 3. Computer. 4. 84 88 91 8A 8E 82 8E 84. 5. а) $185_{(10)}$; б) $224_{(10)}$; в) $193_{(10)}$. 6. а) $89_{(10)}$; б) $-65_{(10)}$; в) $-8_{(10)}$. 7. а) $29407_{(10)}$; б) $25342_{(10)}$. 8. а) $23641_{(10)}$; б) $-23070_{(10)}$. 9. а) 0111011101000111; б) 1010110110101110. 10. а) 215,15625; б) –143,375. 11. а) C071760000000000; б) 407FF28000000000.
Вариант 5	1. а) $88_{(10)}$; б) $153_{(10)}$; в) $718_{(10)}$. 2. а) $000110000100_{(2\text{-}10)}$; б) $100110000111_{(2\text{-}10)}$; в) $100100011000_{(2\text{-}10)}$. 3. Printer. 4. 43 4F 4D 50 55 54 45 52. 5. а) $158_{(10)}$; б) $134_{(10)}$; в) $190_{(10)}$. 6. а) $64_{(10)}$; б) $-104_{(10)}$; в) $-47_{(10)}$. 7. а) $30539_{(10)}$; б) $26147_{(10)}$. 8. а) $22583_{(10)}$; б) $-28122_{(10)}$. 9. а) 0100011011110111; б) 1011101001100000. 10. а) –900,546875; б) –834,5. 11. а) 407C060000000000; б) C0610C0000000000.
Вариант 6	1. а) $325_{(10)}$; б) $112_{(10)}$; в) $713_{(10)}$. 2. а) $100101100010_{(2\text{-}10)}$; б) $001001000110_{(2\text{-}10)}$; в) $011100110110_{(2\text{-}10)}$. 3. компьютеризация. 4. 50 52 49 4E 54. 5. а) $239_{(10)}$; б) $160_{(10)}$; в) $182_{(10)}$. 6. а) $55_{(10)}$; б) $-89_{(10)}$; в) $-22_{(10)}$. 7. а) $17863_{(10)}$; б) $25893_{(10)}$. 8. а) $24255_{(10)}$; б) $-26686_{(10)}$. 9. а) 0000010101011010; б) 1001110100001011. 10. а) –969,15625; б) –434,15625. 11. а) C082B30000000000; б) C086EB0000000000.

Вариант 7	1. а) $464_{(10)}$; б) $652_{(10)}$; в) $93_{(10)}$. 2. а) $000110010010_{(2\text{-}10)}$; б) $001100011000_{(2\text{-}10)}$; в) $011000010000_{(2\text{-}10)}$. 3. YAMAHA. 4. 4D 4F 44 45 4D. 5. а) $237_{(10)}$; б) $236_{(10)}$; в) $240_{(10)}$. 6. а) $95_{(10)}$; б) $-68_{(10)}$; в) $-77_{(10)}$. 7. а) $28658_{(10)}$; б) $29614_{(10)}$. 8. а) $31014_{(10)}$; б) $-24013_{(10)}$. 9. а) 0001101111111001; б) 1011101101001101. 10. а) –802,15625; б) –172,375. 11. а) C085EB0000000000; б) C07D428000000000.
Вариант 8	1. а) $342_{(10)}$; б) $758_{(10)}$; в) $430_{(10)}$. 2. а) $010110010000_{(2\text{-}10)}$; б) $011101100101_{(2\text{-}10)}$; в) $011100010111_{(2\text{-}10)}$. 3. световое перо. 4. 4C 61 73 65 72 5. а) $136_{(10)}$; б) $130_{(10)}$; в) $239_{(10)}$. 6. а) $82_{(10)}$; б) $-13_{(10)}$; в) $-77_{(10)}$. 7. а) $27898_{(10)}$; б) $24268_{(10)}$. 8. а) $19518_{(10)}$; б) $-16334_{(10)}$. 9. а) 0000110100001001; б) 1001110011000000. 10. а) 635,5; б) –555,15625. 11. а) C07848C000000000; б) C085394000000000.
Вариант 9	1. а) $749_{(10)}$; б) $691_{(10)}$; в) $1039_{(10)}$. 2. а) $100100010001_{(2\text{-}10)}$; б) $001000111001_{(2\text{-}10)}$; в) $001101100011_{(2\text{-}10)}$. 3. Микропроцессор. 4. 88 AD E4 AE E0 AC A0 E2 A8 AA A0. 5. а) $230_{(10)}$; б) $150_{(10)}$; в) $155_{(10)}$. 6. а) $74_{(10)}$; б) $-43_{(10)}$; в) $-21_{(10)}$. 7. а) $18346_{(10)}$; б) $25688_{(10)}$. 8. а) $31397_{(10)}$; б) $-21029_{(10)}$. 9. а) 0110101101111000; б) 1110100100110101. 10. а) 110,546875; б) –743,375. 11. а) C08B794000000000; б) 407CB28000000000.

Лабораторная работа №3

Разработка блок-схем алгоритмов. Знакомство с MS Visio

Цель работы: изучить принципы построения блок-схем для различных типов алгоритмов.

Программа работы

1. Изучить и закрепить основы разработки блок-схем.
2. Освоить векторный редактор MS Visio в части разработки блок-схем.
3. Разработать блок-схему в соответствии с вариантом.

Варианты заданий

Вариант 1	Дана функция $y=a\cdot x^2+b\cdot x+c$. Коэффициенты a, b, c являются константами, а x находится в интервале [–10, 18] и изменяется с шагом h, значение которого вводится с клавиатуры. Найти все значения функции для заданных x.
Вариант 2	Дана функция $y=2^{n-1}+3$. Найти значение переменной n, при котором значение функции превысит M=1000.
Вариант 3	Найти значение функции. В данной функции t, a, s – константы, x – вводится с клавиатуры. $$y=\begin{cases} t, & x\geq 3 \\ a\cdot x-s, & -5{,}5\leq x\leq 3 \\ x^3, & x\leq -5{,}5 \end{cases}$$
Вариант 4	Найти число M натуральных чисел n таких, что $n^2+n^3\leq N$, где N – заданное натуральное число.

Вариант 5	Найти число M натуральных чисел n=1…M и сумму $S=\sum_{n=1}^{M} n^2$, чтобы выполнялось условие $S \le N$, где N – заданное натуральное число.
Вариант 6	Найти число M натуральных чисел n=1…M таких, что и $n^2 < N$ и вычислить сумму $S=\sum_{n=1}^{M} \frac{(n-a)^2}{N}$, где N, a – заданные числа; N – натуральное число.
Вариант 7	Для числовой последовательности $a_n = (n-1)/n^2$, n = 1,2,… Найти первый член и его номер M такой, чтобы $a_n < \varepsilon$, где ε – заданное число, например, ε=0,001 и вычислить сумму $S=\sum_{n=1}^{M} a_n$.
Вариант 8	Пользуясь тем, что $\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-...+(-1)^n\frac{x^{2n}}{(2n)!}$, вычислить значение $\cos(x)$ для указанного значения x, заданного в радианах, с точностью ε = 0,001. Точность вычисления считается выполненной, если последний по модулю член в сумме меньше ε.
Вариант 9	Пользуясь тем, что $e=1+1+\frac{1}{2!}+\frac{1}{3!}+...+\frac{1}{n!}$ вычислить значение e с точностью ε=0,0001. Точность вычисления считается выполненной, если последний член в сумме меньше ε.
Вариант 10	Для числовой последовательности $a_n=\sqrt{n+1}-\sqrt{n}$, n = 1,2,…. найти первый член и его номер M такой, чтобы $a_n < \varepsilon$, где ε – заданное число, например, ε=0,001, и вычислить сумму $S=\sum_{n=1}^{M} a_n$.

Лабораторная работа №4

Построение моделей предметной области с использованием case-средства BPwin

Цель работы: создание в среде BPwin функциональной модели системы в нотации IDEF0.

Программа работы

1. Кратко описать выбранную предметную область (чем занимается предприятие, какие основные процессы в нем происходят).
2. Определить контекст моделирования.
3. Указать тип модели.
4. Построить контекстную диаграмму в нотации IDEF0.
5. Построить диаграмму декомпозиции первого уровня в нотации IDEF0.
6. Построить две диаграммы декомпозиции второго уровня для двух наиболее интересных блоков с диаграммы декомпозиции первого уровня.
7. Выполнить работу по варианту, приведенному ниже:

Варианты заданий

№	Задание
1	Строительство загородного коттеджа и обустройство участка
2	Написание курсового проекта по дисциплине «Программирование»
3	Расчет себестоимости изделия
4	Определение величины таможенных сборов на базе контрактов коммерческой фирмы
5	Программа учета жилищного фонда
6	Программа определения затрат рабочего времени на выполнения строительных работ
7	Расчет необходимых объемов сырья промышленного предприятия
8	Работа отдела кадров предприятия

9	Сервисное обслуживание банкомата и его работа
10	Автоматизированная система работы автовокзала

Лабораторная работа №5

Знакомство с файл-серверной СУБД MS Access

Цель работы: изучение возможностей СУБД MS Access при обработке информации.

Программа работы

Создать базу данных из двух таблиц по заданной предметной области.

1. Создать 2 таблицы в режиме **Конструктор,** определить поля, их типы, установить ограничения на значения полей (не менее 10 полей на две таблицы, использовать различные типы данных: текстовый, числовой, дата-время, логический, денежный).
2. Осуществить связывание таблиц с помощью **Мастера подстановок**, сформировать схему базы данных, определить тип связи.
3. Внести не менее 5 записей в основную и не менее 15 записей в подчиненную таблицы.
4. Создать сложную форму, работающую с таблицами, поместить на форме несколько кнопок, воспользовавшись мастером.
5. Создать запросы к таблицам:
   а) с шаблоном ***Like***;
   б) со сложным условием;
   в) многотабличный.
6. Создать с помощью мастера отчет на один из запросов.

В отчет по лабораторной работе поместить: схему базы данных; твердую копию экрана с заполненными таблицами; формами; текстом запросов; конструктором запросов; отчетом. Ответить на контрольные вопросы.

Варианты заданий

1. Каталог фильмов (ЖАНР – ФИЛЬМ).
2. Каталог товаров (ВИД ТОВАРА – ТОВАР).
3. Каталог книг (ИЗДАТЕЛЬСТВО – КНИГА).
4. Журнал класса (КЛАСС – УЧЕНИК).
5. Каталог музыки (АЛЬБОМ – ПЕСНЯ).
6. Современные процессоры, их характеристики (ПРОИЗВОДИТЕЛЬ – ПРОЦЕССОР).
7. Программное обеспечение (ВИД – ПРОГРАММА).
8. Компьютерные игры (ЖАНР – ИГРА).
9. Футбол (КОМАНДА – ИГРОК).
10. Компьютерное оборудование (ТИП – ОБОРУДОВАНИЕ).

Лабораторная работа №6

Использование стандартных программ операционной системы Windows

Цель работы: изучение возможностей стандартных программ для работы с текстовой, числовой, изобразительной, мультимедийной информацией, с информацией Интернета. Выполнение архивации, записи лазерных дисков, обслуживание дисков средствами Windows.

Программа работы

Познакомиться с функциональными возможностями следующих стандартных программ операционной системы Windows (их список может быть скорректирован версией ОС):

1. Записки, текстовые редакторы Блокнот и WordPad.
2. Калькулятор, его режимы работы.
3. Графический редактор Paint.
4. Проигрыватель Windows Media.
5. Браузер Internet Explorer.
6. Запись информации на лазерные диски.
7. Служебные программы проверки, дефрагментации и очистки дисков.

Отчет по лабораторной работе должен содержать иллюстрацию проделанных действий с их описанием.

Учебное задание

1. Создать несколько **Записок** с текстом, задать их цвет и размер, научиться сворачивать, закрывать и удалять **Записки.**
2. В программе **Блокнот** создать и открыть текстовый файл и записать в него:
   а) список стандартных программ операционной системы Windows;
   б) способы копирования файлов с использованием Проводника;
   в) способы выделения всех файлов, группы файлов от 1-го до n-го и произвольного набора файлов в Проводнике;
   г) способы запуска программ и открытия файлов данных.
3. Открыть в программе **WordPad** созданный в **Блокноте** документ. Переоформить списки пунктов 2.а – 2.г в виде нумерованных и маркированных списков.
   Вставить в текст:
   а) картинку проводника с Вашей папкой;
   б) нарисовать в **WordPad**'е картинку – настольный компьютер;
   в) познакомиться, какие еще объекты можно вставить в текст;
   г) сохранить файл в формате RTF.
4. Запустить **Калькулятор**, выполнить вычисления в режимах Обычный, Инженерный, Статистика. Скопировать все вычисления и их результаты в свой текстовый файл с заголовком «*Расчеты в Калькуляторе*», сохранить файл.
5. В графическом редакторе **Paint** продемонстрировать использование различных кистей, фигур с разным типом заливки, текста с разами параметрами шрифта. Сохранить рисунок в различных форматах и сравнить размер полученных файлов и вставить его в текстовый файл.
6. Открыть в **Проигрывателе Windows Media** аудио файл, затем видеофайл. Открыть в **Проигрывателе** эквалайзер, посмотреть варианты его стандартных настроек. Открыть из *Дополнительных возможностей* контекстного меню *Эффекты SRS WOW*,

включить TruBass и настроить эффект объемного звучания WOW.

7. Открыть в браузере **Internet Explorer** сайт университета, найти информацию о своем институте и специальности.
8. Изучите разницу между форматами записи оптических дисков **UDF** и **CDFS**.
9. Открыть окно свойств одного из дисков и найти программы проверки диска и дефрагментации. Запустить программу очистки диска и просмотреть список предлагаемых к удалению файлов.

Лабораторная работа №7

Создание блога. Выбор облачного хранилища

Цель работы: Ознакомиться с современными способами обмена информацией через Интернет, создать блог и облачное хранилище, как средства отчета по дисциплине и взаимодействия с преподавателем.

Программа работы

1. Зарегистрируйте аккаунт в системе Google.
2. Создайте блог-портфолио студента, опубликуйте приветственное сообщение с аватаром.
3. Выберите технологию облачного хранения файлов, выложите выполненные лабораторные работы по дисциплине «Введение в компьютерные науки» в блог.

Отчет по лабораторной работе должен содержать иллюстрацию проделанных действий с их описанием.

1.2. Заочная форма обучения

№	Наименование тем занятий	Кол-во часов	Форма контроля	Сроки контроля
1	*Представление данных в памяти ЭВМ*	1	Защита отчета	В период лабораторно-экзаменационной сессии
2	Использование стандартных программ операционной системы Windows	1	Защита отчета	
3	Разработка блок-схем алгоритмов. Знакомство с MS Visio	1	Защита отчета	
4	*Знакомство с файл-серверной СУБД MS Access*	1	Защита отчета	

Лабораторная работа №1
Представление данных в памяти ЭВМ

Соответствует лабораторной работе №2 из раздела 1.1.

Лабораторная работа №2
Использование стандартных программ операционной системы Windows

Соответствует лабораторной работе №6 из раздела 1.1.

Лабораторная работа №3
Разработка блок-схем алгоритмов. Знакомство с MS Visio

Соответствует лабораторной работе №3 из раздела 1.1.

Лабораторная работа №4
Знакомство с файл-серверной СУБД MS Access

Соответствует лабораторной работе №5 из раздела 1.1.

2. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

2.1. Очная форма обучения

СРС – темы и (или) разделы тем для самостоятельного изучения, в том числе конспектирование – 124,2 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 10. Архитектура ЭВМ** Арифметические и логические команды. Взаимодействие центрального процессора с периферийными устройствами. Иные типы архитектуры компьютеров.	25	[1-6]
2	**Тема 11. Диспетчеризация процессов** Алгоритмы диспетчеризации в централизованных и распределённых системах. Организация доступа к ресурсам.	25	[1-6]
3	**Тема 12. Основные понятия ООП** Классы. Свойства, поля и методы. Инкапсуляция. Наследование полиморфизм.	24,2	[1-6]
4	**Тема 13. Методы проектирования ПО** Структурная и функциональная декомпозиция. Диаграммы SADT, IDEF0. Разновидности диаграмм UML.	25	[1-6]
5	**Тема 14. Компьютерные сети** Назначение и функции сетевых адаптеров. Современные сетевые протоколы и их организация. Физическое и логическое перемещение данных.	25	7 [1-6]

Задания для самостоятельного изучения и конспектирования

Тема 10. Архитектура ЭВМ. Рассмотреть следующие вопросы:

- понятие и основные виды архитектуры ЭВМ;
- система команд ЭВМ и способы обращения к данным;
- логические основы работы процессора;
- логические операции с числами;
- одноразрядный сумматор;

– арифметические операции с числами;
– представление чисел в памяти ЭВМ;
– архитектура центрального процессора, назначение его составных частей;
– основные параметры процессора;
– сопроцессор ввода-вывода;
– контроллер прямого доступа к памяти.

Тема 11. Диспетчеризация процессов. Рассмотреть следующие вопросы:

– основные задачи системы диспетчеризации;
– основные требования к оптимальному построению системы диспетчеризации;
– способы передачи информации в системах диспетчеризации;
– алгоритм синхронизации логических часов;
– алгоритмы взаимного исключения: централизованный алгоритм, распределенный алгоритм, алгоритм Token Ring;
– неделимые транзакции;
– управление приоритетами.

Тема 12. Основные понятия ООП. Рассмотреть следующие вопросы:

– классификация подвидов ООП;
– принцип инкапсуляции в ООП;
– классы и объекты, понятие экземпляра класса, понятие членов класса;
– интерфейс и реализация, наследование реализации;
– состояние объекта, понятие областей доступа, конструкторы;
– типы наследования;
– полиморфизм подтипов, сочетание разновидностей полиморфизма.

Тема 13. Методы проектирования ПО. Рассмотреть следующие вопросы:

– основные задачи системного анализа;
– основные стратегии декомпозиции;
– формирование общего представления системы;
– формирование детального представления системы;

- основные концепции методологии SADT;
- иерархия SADT-диаграмм;
- синтаксис SADT-диаграмм;
- синтаксис моделей и работа с ними;
- основные концепции методологии IDEF0;
- классификация входов и выходов работ в методологии IDEF0;
- основные элементы диаграммы;
- принцип декомпозиции при построении модели бизнес процессов.

Тема 14. Компьютерные сети. Рассмотреть следующие вопросы:

- настройка сетевого адаптера и трансивера;
- основные операции сетевого адаптера при приеме или передаче сообщения;
- типы сетевых адаптеров;
- семиуровневая модель взаимодействия открытых систем OSI;
- стек протоколов TCP/IP.

2.2. Заочная форма обучения

СРС – темы и (или) разделы тем для самостоятельного изучения – 163,4 ч.

№	Наименование тем	Кол-во часов	Литература
1	**Тема 1. Хранение и обработка данных**	40	[1-8]
2	**Тема 2. Операционные системы и сети**	40	[1-8]
3	**Тема 3. Алгоритмы и языки программирования**	40	[1-8]
4	**Тема 4. Организация данных. Компьютерная графика**	43,4	[1-8]

СРС зач. – самостоятельная работа по подготовке к зачету во время лабораторно-экзаменационной сессии – 5,75 ч.

Задания для самостоятельного изучения и конспектирования

Тема 1. Хранение и обработка данных. Рассмотреть следующие вопросы:

- основная память и массовая память;
- представление информации в виде комбинации двоичных разрядов;
- двоичная система счисления;
- представление целых чисел и дробных значений;
- универсальные методы сжатия данных;
- ошибки при передаче информации.

Тема 2. Операционные системы и сети. Рассмотреть следующие вопросы:

- эволюция операционных систем;
- архитектура операционных систем;
- координация действий машины;
- организация конкуренции между процессами;
- основы компьютерных сетей;
- всемирная паутина (WWW);
- интернет-протоколы.

Тема 3. Алгоритмы и языки программирования. Рассмотреть следующие вопросы:

- понятие алгоритма;
- представление алгоритма;
- создание алгоритма;
- итерационные структуры;
- рекурсивные структуры;
- концепции традиционного программирования;
- объектно-ориентированное программирование;
- программирование параллельных процессов;
- декларативное программирование.

Тема 4. Организация данных. Компьютерная графика. Рассмотреть следующие вопросы:

- массивы, списки, стеки, очереди;
- древовидные структуры;

– специализированные типы данных;
– указатели в машинном языке;
– общие понятия баз данных;
– реляционная модель;
– объектно-ориентированные базы данных;
– обеспечение целостности баз данных;
– возможности компьютерной графики;
– моделирование и визуализация (Rendering);
– работа с глобальным освещением;
– анимация.

Контактная внеаудиторная работа

Очная форма обучения:

СРС – групповые консультации с преподавателем в течение семестра – 1,8 ч.

Заочная форма обучения:

СРС – групповые консультации во время лабораторно-экзаменационной сессии – 0,6 ч.

СРС зач. – сдача зачета – 0,25 ч.

Зачетные вопросы по курсу

Очная форма обучения:

1. Понятие и роль алгоритма.
2. Происхождение вычислительных машин.
3. Эволюция компьютерных наук.
4. Роль абстракции.
5. Этические, социальные и правовые аспекты компьютерных наук.
6. Вентили и триггеры.
7. Хранение битов.
8. Шестнадцатеричная система счисления.
9. Основная память.
10. Массовая память.

11. Представление информации в виде комбинации двоичных разрядов.
12. Двоичная система счисления.
13. Представление целых чисел.
14. Представление дробных значений.
15. Сжатие данных.
16. Ошибки при передаче информации.
17. Центральный процессор.
18. Концепция хранимой программы.
19. Выполнение программы.
20. Арифметические и логические команды.
21. Взаимодействие с другими устройствами.
22. Другие типы архитектуры компьютеров.
23. Эволюция операционных систем.
24. Архитектура операционных систем.
25. Координация действий машины.
26. Организация конкуренции между процессами.
27. Основы компьютерных сетей.
28. Интернет.
29. Всемирная паутина (WWW).
30. Интернет-протоколы.
31. Сетевая безопасность.
32. Понятие алгоритма.
33. Представление алгоритма.
34. Создание алгоритма.
35. Итерационные структуры.
36. Рекурсивные структуры.
37. Эффективность и правильность алгоритмов.
38. Исторический обзор языков программирования.
39. Концепции традиционного программирования.
40. Процедуры и функции.
41. Реализация языка программирования.
42. Объектно-ориентированное программирование.
43. Программирование параллельных процессов.
44. Декларативное программирование.
45. Предмет технологии разработки программного обеспечения.

46. Жизненный цикл программного обеспечения.
47. Модульность программного обеспечения.
48. Методы проектирования программного обеспечения.
49. Тестирование программного обеспечения.
50. Документирование программного обеспечения.
51. Право собственности и ответственность за создаваемое программное обеспечение.
52. Массивы.
53. Списки.
54. Стеки.
55. Очереди.
56. Древовидные структуры.
57. Специализированные типы данных.
58. Указатели в машинном языке.
59. Роль операционной системы.
60. Последовательные файлы.
61. Текстовые файлы.
62. Индексация.
63. Хеширование.
64. Общие понятия баз данных.
65. Многоуровневый подход к реализации баз данных.
66. Реляционная модель баз данных.
67. Объектно-ориентированные базы данных.
68. Обеспечение целостности баз данных.
69. Влияние технологий баз данных на общество.
70. Машины и интеллект.
71. Распознавание изображений.
72. Способность к рассуждению.
73. Искусственные нейронные сети.
74. Генетические алгоритмы.
75. Приложения теории искусственного интеллекта.
76. Осмысливание последствий ИИ.
77. Простейший язык программирования.
78. Машины Тьюринга.
79. Вычислимые функции.
80. Невычислимые функции.

81. Сложность задач.
82. Криптография с использованием открытых ключей.

Заочная форма обучения:

1. Понятие и роль алгоритма.
2. Вентили и триггеры.
3. Шестнадцатеричная система счисления.
4. Хранение битов.
5. Двоичная система счисления.
6. Центральный процессор.
7. Эволюция операционных систем.
8. Архитектура операционных систем.
9. Основы компьютерных сетей.
10. Интернет.
11. Всемирная паутина (WWW).
12. Интернет-протоколы.
13. Понятие алгоритма.
14. Представление алгоритма.
15. Создание алгоритма.
16. Исторический обзор языков программирования.
17. Объектно-ориентированное программирование.
18. Программирование параллельных процессов.
19. Декларативное программирование.
20. Жизненный цикл программного обеспечения.
21. Методы проектирования программного обеспечения.
22. Тестирование программного обеспечения.
23. Роль операционной системы.
24. Текстовые файлы.
25. Общие понятия баз данных.
26. Простейший язык программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Советов Б.Я. Информационные технологии: теоретические основы [Электронный ресурс]: учеб. пособие / Б.Я. Советов, В.В. Цехановский. – Электрон. дан. – СПб. : Лань, 2016. – 442 с. –

Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=71733 – Загл. с экрана.
2. Цехановский В.В. Управление данными [Электронный ресурс]: учебник / В.В. Цехановский, В.Д. Чертовской. – Электрон. дан. – СПб.: Лань, 2015. – 432 с. – Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=65152 – Загл. с экрана.
3. Кудинов Ю.И. Основы современной информатики [Электронный ресурс]: учеб. пособие / Ю.И. Кудинов, Ф.Ф. Пащенко. – Электрон. дан. – СПб.: Лань, 2011. – 256 с. – Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=68468 – Загл. с экрана.
4. Информатика. Алгоритмы. Модели. Программы : учеб. пособие / А. А. Ренсков, А. Ю. Чижов, А. С. Родионов, В. Е. Меркулов; Гос. образовательное учреждение высшего профессионального образования "НВВКУС (Военный ин-т)". – Новочеркасск : изд-во НВВКУС, 2009. – 229 с.
5. Ренсков А. А. Информатика. Информационные технологии в науке и образовании : учеб. пособие / А. А. Ренсков, А. Ю. Чижов. - Новочеркасск : изд-во НВВКУС, 2008. - 128 с.
6. Воробьев С. П. Информатика. Сетевые технологии : учеб. пособие / С. П. Воробьев. - Новочеркасск : изд-во НВВКУС, 2006. – 103 с.
7. Георгица И.В., Мохов В.А., Есаулов В.А., Синецкий Р.М. Введение в компьютерные науки учебное пособие : учеб. пособие / Юж.-Рос. гос. политехн. ун-т (НПИ). – Новочеркасск : ЮРГПУ (НПИ), 2015. – 91 с.
8. Георгица И.В. Информатика: учебно-методическое пособие к лабораторнымработам для студентов направления «Информатика и вычислительная техника» / Юж.-Рос. гос. техн. ун-т (НПИ). – Новочеркасск: ЮРГТУ (НПИ), 2012. – 108 с.

Учебно-методическое издание

Мохов Василий Александрович
Кущий Дарья Николаевна

Введение в компьютерные науки

Методические указания
к выполнению лабораторных
и самостоятельных работ

Редактор *Я.В. Максименко*
Подписано в печать 14.04.2017
Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,63. Уч.-изд.л. 1,75. Тираж 50 экз. Заказ .

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет
(Новочеркасский политехнический институт)

Т.Н.Воронцова

ФИЛОСОФИЯ

Учебно-методическое пособие

для подготовки к семинарским занятиям

и организации самостоятельной работы студентов

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 167.1 (076.5)

Рецензент – доктор философских наук, профессор, зав. кафедрой «Философия и право» ЮРГПУ (НПИ) **В.С. Любченко**.

Воронцова Т.Н.

Философия: учебно-методическое пособие для подготовки к семинарским занятиям и организации самостоятельной работы студентов / Воронцова Т.Н.; Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2017. – 67 с.

Учебно-методическое пособие по философии для подготовки к семинарским занятиям и организации самостоятельной работы студентов разработано на основе рабочей программы по курсу философии в соответствии с требованиями Федерального государственного образовательного стандарта высшего образования.

Содержит планы семинарских занятий, подробное разъяснение каждой из тем, рекомендации по самостоятельному овладению учебным материалом, вопросы для самопроверки, список рекомендуемой литературы, а также примерный перечень вопросов к зачету.

Целью данного пособия является оказание помощи студентам в изучении курса философии при подготовке к семинарским занятиям и организации самостоятельной работы.

Учебно-методическое пособие предназначено для студентов факультета информационных технологий и управления (направления 02.03.03 Математическое обеспечение и администрирование, 09.03.01 Информатика и вычислительная техника, 09.03.02 Информационные системы и технологии, 09.03.04 Программная инженерия, 09.03.03 Прикладная информатика, 11.03.02 Инфокоммуникационные технологии и системы связи, 11.03.04 Электроника и наноэлектроника, 12.03.01 Приборостроение, 27.03.04 Управление в технических системах), энергетического факультета (направление 13.03.02 Электроэнергетика и электротехника) и механического факультета (направления 15.03.02 Технологическое оборудование химических и нефтехимических производств, 15.04.02 Машины и аппараты пищевых производств, 15.03.05 Конструкторско-технологическое оборудование машиностроительных производств, 23.05.01 Наземные транспортно-технологические комплексы, 23.03.03 Эксплуатация транспортно-технологических машин и комплексов) очной и заочной форм обучения.

ВВЕДЕНИЕ

В настоящее время философия представляет собой комплексную систему знаний, основным содержанием которой являются наиболее общие принципы бытия и познания, законы функционирования и развития объективного мира. Она раскрывает культурно-историческое единство человечества, обосновывает критически-рациональный подход к процессам и явлениям, необходимость разумного начала во взаимодействии человека с миром.

Изучение дисциплины «Философия» призвано способствовать развитию интеллекта, выработки мировоззренческих ориентиров, расширению эрудиции, развитию абстрактного мышления и формированию навыков самостоятельного творческого мышления.

Данное учебно-методическое пособие разработано на основе рабочей программы учебной дисциплины «Философия» в соответствии с требованиями Федерального государственного образовательного стандарта высшего образования. В нем представлены планы семинарских занятий, в которых предложены к обсуждению вопросы по истории мировой и отечественной философии, современным философским проблемам человека, общества и науки. Содержится подробное разъяснение каждой из тем.

В пособие включены рекомендации по самостоятельному овладению учебным материалом, вопросы для самопроверки, список рекомендуемой литературы, а также примерный перечень вопросов к зачету.

Приведенный перечень тем семинарских занятий и самостоятельной работы имеет разный количественный набор для студентов очной и заочной форм обучения в соответствии с рабочими программами дисциплины. В связи с этим преподавателям следует конкретизировать задания студентам для успешного освоения материала по курсу «Философия» вне зависимости от формы обучения.

Список литературы включает в себя новейшие учебники, рекомендованные Министерством образования и науки Российской Федерации, которые обеспечиваются научно-технической библиотекой университета. Кроме того, предлагается перечень электронных ресурсов, доступ к которым возможен через интернет.

Учебно-методическое пособие предназначено для студентов факультета информационных технологий и управления (направления 02.03.03 МОиА, 09.03.01 ИиВТ, 09.03.02 ИСиТ, 09.03.04 Программная инженерия, 09.03.03 Прикладная информатика, 11.03.02 ИТиСС, 11.03.04 ЭиН, 12.03.01 Приборостроение, 27.03.04 УвТС очной и заочной форм обучения), энергетического факультета (направление 13.03.02 Электроэнергетика и электротехника) и механического факультета (направления 15.03.02 ТОХиНХП, 15.04.02 МАПП, 15.03.05 КТОМП,23.05.01 НТТС,23.03.02 НТТК, 23.03.03 ЭТТМиК) очной и заочной форм обучения.

Тематика занятий семинарского типа

Тема 1. Мировоззрение, его виды. Философия как форма мировоззрения.
Тема 2. Диалектика взаимосвязи природы и общества.
Тема 3. Общество как система. Социальная структура общества. Понятие классов и страт.
Тема 4. Типы общества по экономической основе и по социокультурным характеристикам.
Тема 5. Содержание и основные направления философии истории. Формационная концепция социального развития.
Тема 6. Концепции «индустриализма» и «постиндустриализма».
Тема 7. Цивилизационный подход к истории человечества.
Тема 8. Проблема человека в философии.
Тема 9. Глобальные проблемы современности.

Темы для самостоятельной работы студентов

Тема 1. Изменение предмета философии в ходе ее исторического развития.
Тема 2. Натурфилософия античности. Учение о первоначалах мира. Античная атомистика. Учение о стихийной диалектике в античной философии. Классическая древнегреческая философия. Сократ. Платон. Аристотель. Философия эпохи эллинизма. Скептицизм. Эпикуреизм. Стоицизм.
Тема 3. Средневековая философия, ее характерные черты. Патристика. Основные проблемы средневековой схоластики. Проблема соотношения веры и разума в средневековой философии.
Тема 4. Философия и наука эпохи Возрождения.
Тема 5. Философия Нового времени.
Тема 6. Антропологические идеи французского Просвещения. Социальные взгляды Ф.Вольтера и Ж.-Ж. Руссо. Этическое учение И.Канта. Категорический императив и нравственный долг человека. Абсолютная идея Г. Гегеля как смысл исторического процесса.
Тема 7. Философские аспекты труда, частной собственности и отчуждения, идея коммунизма в философии К.Маркса. Проблема культурного самоопределения России в отечественной философии. Философия русского зарубежья.
Тема 8. Современная западная философия. Экзистенциализм. Философия позитивизма и его исторические формы. Психоанализ и его эволюция.
Тема 9. Глобальные научные революции в истории естествознания. Понятия «картина мира» и «парадигма».
Тема 10. Наука как особый вид деятельности. Закономерности развития науки. Синергетика как новое мировидение XXI в. Мир как самоорганизующаяся система.
Тема 11. Идеальность сознания. Концепции идеального в отечественной философии.
Тема 12. Проблема познаваемости мира в истории европейской философии. Проблема истины. Характеристики истины. Многообразие концепций истины и ее критерии.
Тема 13. Основные категории морали, их ценностная характеристика. Общечеловеческие ценности.
Тема 14. Глобализация как ведущая тенденция развития современного мира: основные направления и проблемы.

ПЛАНЫ СЕМИНАРСКИХ ЗАНЯТИЙ ПО ФИЛОСОФИИ

Тема 1. Мировоззрение, его виды. Философия как форма мировоззрения

1. Понятие мировоззрения, его структура. Типы мировоззрения: мифологическое, религиозное, научное, философское. Специфика философского мировоззрения.
2. Основной вопрос философии. Суть двух противоположных типов философского мировоззрения: идеалистического и материалистического. Формы материализма. Разновидности идеализма.
3. Возникновение философии.

Методические рекомендации к изучению темы

1.Знакомство с философией, как правило, начинается с выяснения понятия «мировоззрение», ибо философия представляет собой определенный тип его.

В общем виде мировоззрение – это совокупность наиболее общих представлений о мире и человеке, о месте человека в мире, о цели и смысле его существования. Мировоззрение организует человеческую деятельность, направляет общество и отдельного человека на достижение определенных целей.

Студентам следует выделить интеллектуальный (миропонимание) и эмоциональный (мироощущение) уровни в структуре мировоззрения и охарактеризовать их.

Выделение типов мировоззрения достаточно условно и может производиться по различным основаниям. Чаще всего различают *мифологическое, религиозное, научное и философское* мировоззрение. Их сравнение позволяет выяснить специфику философского мировоззрения, понять его отличия и сходство с другими типами мировоззрения. В частности, философское мировоззрение так же, как и научное, опирается на логику и доказательство в создании представлений о мире, однако, науку в большей степени интересует мир сам по себе, философия же сосредотачивается на человеке, осмыслении его места в мире. Философское мировоззрение имеет теоретический характер, в науке большую роль играют эмпирические доказательства.

В решении смысложизненных проблем философия имеет много общего с религией: обе устремлены в вечность, выясняют цели и ценности человеческого существования, однако религиозное мировоззрение опирается на веру, обращается к чувствам и эмоциям человека, а философия – к разуму.

При рассмотрении специфики философского мировоззрения студентам следует выделить такие его черты как теоретичность, субстанциальность, универсальность, системность, доказательность, концептуальность и охарактеризовать их.

2. При рассмотрении второго вопроса следует обратить внимание на то, что отношение «человек – мир» занимает центральное место в философском мировоззрении. Постепенно оно трансформируется в более абстрактную проблему соотношения идеального (субъективного, духовного) и материального (объективного, природного), называемую «основным вопросом философии». В зависимости от его решения сформировались два основных направления – *материализм* и *идеализм*. Они дают противоположные ответы на главные мировоззренческие вопросы: 1) создан ли мир Богом или существует вечно; 2) чем определяются происходящие в мире процессы; 3) изменяется ли мир или остается неизменным; 4) что такое человек; 5) в чем смысл его существования.

Студентам следует охарактеризовать исторические формы материализма и выяснить особенности разновидностей идеализма – субъективного и объективного.

Необходимо отметить, что проблема познаваемости мира является одной из важнейших в философии. Она была поставлена уже в античности. Следует выделить различные философские позиции в решении этого вопроса – гносеологический оптимизм и пессимизм – и кратко охарактеризовать их.

3. Понимание того, что такое философия, предполагает исследование возникновения философского знания. Решить проблему генезиса философии – значит ответить на вопросы: где, когда, из чего и почему она возникла?

Философия возникла в VII-VI вв. до н.э. в Древней Индии, Древнем Китае и Египте, но своей классической формы достигла в Древней Греции. Во многом это связано с особенностями ее социально-политического устройства: Древняя Греция – родина демократии. В отличие от восточных деспотий, основанных на безусловном подчинении правителю и его обожествлении, античные полисы управлялись выборной и сменяемой властью. Законы, по которым жило общество, открыто обсуждались, что способствовало формированию рациональной составляющей мировоззрения, развитию свободомыслия, стремления к знанию, открытости к новизне.

На вопрос «Из чего возникает философия?» существуют несколько ответов: из мифологии, из религии, из обыденного сознания (житейской мудрости). Наиболее распространенной считается мифологическая концепция происхождения философии.

Мифология является исторически первой формой мировоззрения, которую вырабатывает первобытное (архаическое) общество – общество эпохи охоты и собирательства, ранних форм земледелия.

В архаических обществах миф (в переводе с древнегреческого – рассказ) служил средством закрепления знания, помогавшего выживать родовому коллективу. Повествуя о действиях предков, культурных героев, о происхождении мира, природных явлений, норм, обычаев и правил поведения и т.д., миф учил тому, как следует жить, регулировал отношения в первобытной общине. Община сохранялась благодаря тому, что младшие повторяли

опыт старших. Соблюдение традиций было условием выживания последующих поколений.

Студентам следует охарактеризовать особенности мифологического сознания – антропоморфизм, анимативизм, алогичность, образность и эмоциональность, ассоциативность и др.

Далее рекомендуется осветить причины разрушения мифологической картины мира и предпосылки возникновения философии, которые связаны с социально-экономическими изменениями, происходившими в Древней Греции в VII-VI вв. до н.э.

К ним относится, в первую очередь, совершенствование орудий труда – переход от «века бронзы» к «веку железа». Более эффективные орудия труда укрепили позиций людей в природе, что не могло не сказаться на изменении их мировоззрения и увеличении объема знаний. Железные орудия труда позволили производить гораздо больше благ, появились излишки, что изменило отношения в обществе. Закончился «пещерный коммунизм» и началось классовое расслоение. Появление классов и разделение труда привело к развитию ремесла, торговли, росту городов. Возникли товарно-денежные отношения, которые потребовали развития способностей к счету и абстрактному мышлению на уровне массового сознания. Все эти процессы расшатывали мифологическое мировоззрение и вызывали общественную потребность в новой форме мировоззрения, отвечающей прогрессивным изменениям в обществе. Философия возникла из мифологии, которая дала ей мировоззренческую направленность, а зарождающаяся наука – рациональный характер.

Основная литература: [1, 2, 3, 9]

Вопросы для самопроверки

1. Что такое мировоззрение?
2. В чем специфика философского мировоззрения?
3. В чем суть основного вопроса философии? Почему именно этот вопрос является основным в философском мировоззрении?
4. Чем субъективный идеализм отличается от объективного?
5. Когда, где, из чего и почему возникает философия?

Тема 2. Диалектика взаимосвязи природы и общества

1. Природа как естественная основа жизни и развития общества. Качественное отличие объектов живой природы от неживой.
2. Диалектика взаимосвязи природы и общества на разных этапах исторического развития.
3. Современный экологический кризис.

Методические рекомендации к изучению темы

1. Изучение данной темы необходимо начать с определения понятия «природы», которое имеет как широкое, так и узкое толкование. В первом случае природа рассматривается как материальный мир в целом. Однако чаще понятие «природа» употребляется в более ограниченном смысле как совокупность естественных условий существования человека и общества. Именно в этом смысле и должна рассматриваться природа в данной теме.

Естественной средой обитания человечества является природа Земли, подразделяемая на неживую (неорганическую) и живую (органическую) составляющие. Следует отметить, что еще в древнегреческой философии появилось учение об универсальной одушевленности материального мира, позднее получившее название *гилозоизма*. Приписывая всей материи способность ощущать, гилозоизм снимал принципиальное различие между органической и неорганической природой.

Студенты должны уяснить, что объекты живой природы качественно отличаются от объектов неорганической природы, имеют свою специфику. Они выделяются обменом веществ с окружающей средой, раздражимостью, способностью к размножению и росту.

Особенность Земли состоит в том, что на ней в процессе взаимодействия неорганической геосферы с энергией космического пространства возникла особая оболочка – биосфера (сфера жизни). Однажды возникнув, биосфера стала проявлять особого рода эволюционную динамичность, которая не только «напластовывалась» на геологическую эволюцию нашей планеты, но и определяла этот процесс.

В ходе длительной эволюции живой природы на земном шаре возникло человеческое общество. И хотя человечество в процессе своего развития все больше отдалялось от природы, тем не менее, в основании его жизнедеятельности всегда лежали и продолжают лежать природные факторы. Это дает основание говорить о существовании единой истории природы и общества.

2. По мере развития общества менялась степень влияния на него природной среды. Вместе с тем изменялась и степень воздействия общества на природу. В связи с этим студентам необходимо рассмотреть исторические этапы взаимодействия общества с природой.

Природные условия издавна влияли на расселение людей. На ранних этапах развития общества наибольшую значимость для людей имело естественное богатство средствами жизни (съедобные растения, дикие животные, рыбные богатства и т.д.). В первобытном обществе борьба за обеспеченность пищей охватывала всю деятельность человека. В этих условиях человек одухотворял окружающую природную среду, чувствовал себя во власти сверхъестественных сил. Однако постепенно происходило развитие орудий, позволявших более эффективно воздействовать на эту среду. Появление железных орудий привело к неолитической революции в жизни древних обществ и значительной активизации процесса преобразования природы.

Рабовладельческий период характеризовался более глубоким по сравнению с первобытной эпохой и негативным воздействием общества на при-

роду. Рабы, насильственно привозимые из других регионов, не умели и не хотели заботиться о восстановлении плодородия земли, о сохранении других ресурсов чуждой им природы.

Средневековый феодализм видоизменил систему эксплуатации людей. Крепостной крестьянин, трудившийся на своем клочке земли, был заинтересован в сохранении плодородия почвы. Ограниченность миграции населения, низкие темпы хозяйственного развития предохраняли природу от негативных воздействий человека. Господство церкви породило в период средневековья особый подход к пониманию и оцениванию природы: она считалась результатом божественного творения, но ставилась ниже человека, получившего особое божественное начало – душу. При таком подходе природа представлялась как нечто неодушевленное, противостоящее человеку.

Начавшееся в эпоху Возрождения, а затем Нового времени оживление социально-экономической жизни Европы было связано с началом развития капиталистического производства. С развитием промышленной революции в Европе возрастал спрос на полезные ископаемые. Технический прогресс неизбежно привел к усилению утилитарного отношения к природе. Формируется установка на господство над природой, которая оказывает разрушительное воздействие на нее.

3. Современная эпоха характеризуется возникновением и быстрым углублением экологического кризиса. Если практически до конца XIX в. масштабы и темпы технической деятельности человека были еще сравнительно невелики, и эта деятельность вносила относительно небольшие, локальные изменения в природную среду, то в XX в. отрицательное влияние человечества на природу резко усилилось. Наметилось истощение целого ряда природных ресурсов. В связи с этим студентам рекомендуется рассмотреть полную картину воздействий человека на все важнейшие компоненты природной среды и оценить масштабы и последствия этих воздействий.

Необходимо также рассмотреть понятие ноосферы, которое было введено еще в первой половине XX в. академиком В.И. Вернадским. Он указал на то, что человек стал могучей геологической силой, преобразующей лик Земли. Период в развитии биосферы, когда все усиливающимся фактором этого развития становится разумная человеческая деятельность, он и назвал ноосферой (от греч. «ноос» – разум), т. е. сферой разума. Оптимизм Вернадского был вызван бурными успехами современной ему науки, которые позволяли надеяться на действенный контроль с ее стороны над развитием производства, техники, технологии. Однако, глядя на нынешние трагические для природной среды последствия человеческой деятельности, трудно согласиться, что биосфера вступила в стадию разумного, а не бездумного управления.

Основная литература: [1, 2, 3, 9]

Вопросы для самопроверки

1. Чем отличается отношение к природе человека от отношения к ней животного?
2. В чем заключается качественная специфика объектов живой природы?
3. Можно ли утверждать, что по мере развития общества его воздействие на природу становится все более негативным?
4. В чем сущность и причины современного экологического кризиса?

Тема 3. Общество как система. Социальная структура общества. Понятие классов и страт

1. Общество. Связи и отношения в обществе.
2. Социальная структура общества. Понятие классов.
3. Социальная стратификация.

Методические рекомендации к изучению темы

1. Приступая к рассмотрению данного вопроса, следует обратить внимание на то, что разнообразные определения общества фиксируют наличие форм совместной жизнедеятельности, связей и отношений между людьми. Причем эти связи и отношения имеют устойчивый характер и воспроизводятся в деятельности людей.

Основы современных представлений об обществе были заложены К.Марксом, который впервые осуществил системно-структурный анализ общества. По мысли К. Маркса, общество – это продукт взаимодействия людей, сумма тех связей и отношений, в которых индивиды находятся друг к другу. Важнейшей соединительной связью в обществе является производственная связь, а производственные отношения лежат в основе всех других отношений.

Следует выделить отношения, возникающие в процессе совместной деятельности индивидов, то есть отношения, обусловленные разделением труда в обществе. Здесь наряду с «горизонтальными» связями, где каждый выступает как выполняющий определенную функцию, существуют «вертикальные» связи руководства – подчинения. Еще один тип отношений возникает между людьми на основе распределения совместно созданных результатов труда, где каждый получает определенное вознаграждение за свой труд. Таким образом, любая реальная социальная группа – это набор функций-ролей и статусов. Статус – престижность роли, ее значимость. Статус индивида указывает его место в социальной иерархии. Тем самым общество предстает не только как «горизонтальная» структура, где взаимодействуют равные индивиды, обменивающиеся деятельностью или ее продуктами. Общество – это также «вертикальная» структура, система неравенства, где есть высшие и низшие, престижные и непрестижные виды деятельности, а соответственно есть индивиды, осуществляющие эти виды деятельности, приобретающие высокий или низкий статус, престиж.

Социальные институты организуют деятельность индивидов в различных сферах с помощью различных норм, формальных и неформальных, а также санкций за выполнение или невыполнение соответствующих норм. Социальными институтами являются, например, семья, школа, вуз, армия, парламент, государство в целом, фирма, биржа.

2. Социальная структура – это «устройство» общества, наличие в нем устойчивых социальных групп, а также связей между ними и внутри данных групп. В обществе происходит обособление и институционализация отдельных сфер социальной жизни. Формируются различные социальные группы и общности, внутри которых и между которыми складываются многообразные отношения: профессиональные, экономические, политические, религиозные. Такими социальными группами являются семья, производственный коллектив, студенческая группа, коллектив ученых, община верующих.

Необходимо отметить, что нормальное существование любого общества предполагает осуществление ряда важных функций: производства необходимых средств существования, защиты, поддержания общественного порядка. В современном обществе можно выделить пять основных сфер, которыми являются *материальное производство, духовное производство, коммуникативная, организационная и социальная деятельность*. Студентам предстоит провести анализ всех этих сфер: определить их структуру, задачи различных видов деятельности, выявить те сложные отношения координации и субординации, которые возникают как внутри каждой сферы, так и между ними, определить, какие сферы являются доминирующими в тот или иной исторический период.

В социальной структуре общества обычно выделяют профессиональную, демографическую, этническую, образовательную и поселенческую структуры, однако важнейшей характеристикой общества является его социально- классовая структура. Классы – большие социальные группы, отношения между которыми определяют различные аспекты функционирования и развития общества. Между классами складываются производственно-экономические, политические, идеологические отношения. У них формируются различные интересы, которые ведут к сотрудничеству или конфликту.

Следствием экономических отношений является формирование социальных групп, именуемых социальными классами. Важнейший признак класса – наличие или отсутствие собственности на средства производства. Классы различаются также своей ролью в процессе производства и размером получаемого общественного богатства. *Классы* – это такие социальные группы, одни из которых могут присваивать труд других. Важно подчеркнуть, что наряду с основными признаками класса можно выделить множество производных признаков, определяющих «качество» жизни: неравенство властных статусов; различия в уровне образования классов; различие их мироощущения и стереотипов жизненного поведения.

3.В социальной структуре современного общества исключительно важную роль играют наряду с классами также более подвижные, неустойчивые общности людей, представляющие как слои внутри того или иного класса, так и межклассовые социальные образования и даже социальные группы, не имеющие прямого отношения к классовому делению общества. Это – страты. Деление общества на подобные социальные структуры получило название «стратификация».

Стратификация **(**социальное неравенство) – неравный доступ различных социальных групп к таким социальным благам, как деньги, власть, престиж, образование. В связи с этим, студентам рекомендуется провести анализ одномерной и многомерной стратификации, а также рассмотреть понятие «социальной мобильности».

Студентам следует также сравнить классовый и стратификационный подходы к анализу социальной структуры общества и показать связь последнего с теми изменениями, которые происходят в современном постиндустриальном обществе.

Основная литература: [1, 2, 3, 5, 9]

Вопросы для самопроверки

1. Чем общественные связи и отношения отличаются от тех, которые складываются в животных сообществах?
2. Какие отношения являются определяющими в любом типе общества?
3. Что подразумевается под «горизонтальными» и «вертикальными» связями в обществе?
4. Что является основным классообразующим признаком?
5. Является ли классовое деление общества неизбежным?
6. Чем страты отличаются от классов? По каким критериям выделяются страты?

Тема 4. Типы общества по экономической основе и по социокультурным характеристикам

1.Типы общества по экономической основе
2.Общества Восточного и западного типа.
3.Гражданское общество, государство, демократия.

Методические рекомендации к изучению темы

1. Все многообразие обществ, существовавших прежде и существующих сейчас, можно разделить на определенные типы. Существует множество классификаций (типологий) обществ. Студентам рекомендуется проанализировать некоторые из них.

Согласно марксистской традиции, тип общества определяется способом производства, т.е. тем, как используются и контролируются экономические ресурсы, которыми оно владеет. В связи с этим различаются, например, рабовладельческое и феодальное, капиталистическое и социалистическое общества.

Классификация обществ может быть составлена также в соответствии с основными характерными для них способами получения средств существования, по их социокультурным характеристикам, по способам организации хозяйственной жизни (рыночная или административно – командная экономика) и.т.д.

Широкой известностью пользуется классификация обществ по способам получения средств существования (по их экономической основе). Данная типология делит все общества на: 1) общества, живущие охотой и собирательством; 2) аграрные общества; 3) промышленные (индустриальные) общества. Рассматривая данную классификацию, необходимо обратить внимание на то, какую роль при переходе от общества, живущего охотой и собирательством к аграрному обществу, а от него к промышленному, сыграли совершенствование технологий и использование новых источников энергии.

Наиболее распространенной в современной социальной философии является типология обществ по характеру труда. В ней выделяются доиндустриальные, индустриальные и постиндустриальные общества, которые студентам рекомендуется охарактеризовать.

2.Другой подход к классификации обществ основывается на различии его социокультурных характеристик. В таком случае выделяют общества «восточного» и «западного» типов. Следует подчеркнуть, что в данной типологии под Западом и Востоком имеются в виду не географические понятия, а различные способы мировосприятия, образы жизни и системы ценностей.

Рассматривая общества «восточного» типа, необходимо обратить внимание на их характерные признаки: 1) сосредоточенность на духовном; 2) коллективизм, 3) традиционализм; 4) абсолютное преобладание государства над обществом; 5) гармония с природой; 6) стабильность и др.

Запад – это символ «прогрессивного развития», при котором возникла самоподдерживающаяся (рыночная) экономика, светское правовое государств, демократическое общественное устройство, развитые системы жизнеобеспечения. Для западного общества характерны: 1) ориентация на материальные ценности; 2) индивидуализм; 3) стремление к новизне; 4) потребительское отношение к природе; 5) склонность к риску и др.

Важно подчеркнуть ценность каждой из культур и активное взаимообогащение разных типов общественного развития на современном этапе.

3. Рассмотрение данного вопроса целесообразно начать с характеристики основных черт капиталистического общества. Центральными в его описании являются понятия: гражданское общество, правовое государство,

многопартийность, демократия, разделение властей, частная собственность, рынок, автономия и суверенитет личности.

Гражданское общество – это совокупность добровольных ассоциаций и организаций, огражденная законами от прямого вмешательства со стороны органов государственной власти. В капиталистическом обществе государство не вмешивается в частную жизнь людей, не навязывает им единую идеологию и единую систему ценностей. Многообразные интересы людей реализуются через их совместные действия, для организации которых люди вступают в добровольные, неподотчетные государству объединения и ассоциации. Негосударственные, неправительственные организации, отражающие интересы людей, составляют гражданское общество.

Следует отметить то, что государство всегда стремится взять под свой контроль все стороны жизни граждан, сузить сферу их нерегламентированной деятельности. Поглощение гражданского общества государством составляет одну из характерных черт тоталитаризма (государственного режима, при котором осуществляется полный (тотальный) контроль государства над всеми сферами жизни общества). Гражданское общество не позволяет государству вторгаться в частную жизнь граждан, с одной стороны, а с другой – доносит до власти имеющиеся проблемы. Развитие гражданского общества является важнейшим условием и гарантией стабильности демократии.

Студентам следует рассмотреть основные условия, обеспечивающие успешное функционирование демократии. К ним относятся: устойчивое гражданское общество, правовое государство, свобода слова, достаточно высокий уровень социально-экономического развития общества.

Основная литература: [1, 2, 3, 5, 9]

Вопросы для самопроверки

1. Для чего нужна типология обществ?
2. Чем обусловлена разница восточного и западного типов общества?
3. К какому типу общества относится Россия?
4. В чем заключается необходимость развития гражданского общества?
5. Возможно ли развитие демократии без гражданского общества?

Тема 5. Содержание и основные направления философии истории. Формационная концепция социального развития

1.Содержание, задачи, основные направления философии истории.
2. Формационная концепция общественного развития.

Методические рекомендации к изучению темы

1. Философия истории представляет собой философскую интерпретацию исторического процесса. Элементы философского осмысления истории содержались еще в античной философии. Философия истории как особый раздел философии сложилась только в XVIII в. Название этого раздела впервые предложил Ф. Вольтер.

Содержание и проблематика философии истории существенно изменялась с течением времени. Студентам предстоит проанализировать круг основных задач современной философии истории. Важно также подчеркнуть принципиальные различия исторического и философского подходов к реальной истории.

Философия истории отличается от истории как науки. Если история исследует прошлое и пытается восстановить картину прошлого, то философия истории рассматривает следующие проблемы:

1) периодизация исторического процесса;
2) характер исторического процесса (является ли история прогрессивным изменением или ее развитие циклично);
3) движущие силы исторического процесса (что определяет развитие истории, соотношение в ней свободы и необходимости);
4) смысл истории, ее направленность и цель;
5) возможное будущее.

В философии истории можно выделить два основных направления: *социальный эволюционизм*, рассматривающий исторический процесс как единую линию развития от низшего к высшему, и *цивилизационный подход*, отрицающий единство исторического процесса и прогрессивное развитие истории.

Социальный эволюционизм представляет собой попытку научно-философского осмысления исторического процесса как части общего, бесконечно разнообразного и линейного процесса эволюции Космоса, Земли, культуры. К разряду линейных относится учение К. Маркса, постулировавшее одно-единственное направление истории – к коммунизму как конечной стадии социального развития человечества.

2. *Формационная концепция* Маркса является примером социально-эволюционного подхода.

До Маркса, господствовал идеалистический взгляд на историю. Разум считался главной движущей силой исторического процесса. Маркс же полагал, что прежде чем заниматься наукой, искусством, политикой, человек должен пить, есть, одеваться, иметь жилье и т.д. Он рассматривал производство как главную сферу общественной жизни, основу общества, базис. Все другие сферы зависят от базиса. С точки зрения Маркса, не сознание определяет бытие, а, наоборот общественное бытие определяет сознание.

Весь строй жизни людей зависит, в конечном счете, от способа, каким люди в каждую историческую эпоху производят материальные блага.

Для изложения своей концепции Маркс использовал такие категории как «способ производства», «общественно–экономическая формация», «про-

изводительные силы», «производственные отношения», «базис», «надстройка», «общественное бытие», «общественное сознание» и др. Взятые во взаимосвязи эти категории позволяют интерпретировать различные аспекты материалистического понимания истории.

Основа формации (базис) – способ производства, от него зависит политическая, юридическая надстройка. Смена формации происходит вследствие социальной революции. Причина социальной революции – противоречие производительных сил и производственных отношений. Формационная концепция Маркса господствовала в социальной философии до середины XX в.

Маркс выделяет в истории человечества несколько способов производства: первобытнообщинный, рабовладельческий, феодальный, капиталистический и коммунистический, на основе которых формируется и определенный тип общества, названный им «общественно-экономическая формация».

Маркс представил развитие общества как объективный естественноисторический процесс, последовательную смену общественно-исторических формаций. Студентам следует назвать их и кратко охарактеризовать, отметив, что итогом этого процесса является формирование коммунистического общества, в котором экономика находится под контролем ассоциированных производителей.

Следует заметить, что, по Марксу, исторический процесс может развиваться только в направлении освобождения человека от стихийных природных и социальных сил, однако прогрессистское, линейное видение истории Марксом не оправдалось. Важно также обратить внимание на те новые явления капиталистического развития в эпоху НТР – создание механизмов регулирования экономических и политических противоречий, формирование так называемого социального государства, которые изменили расстановку сил и накал противостояния в буржуазном обществе и которые Маркс не мог предвидеть.

Основная литература: [1, 2, 3, 4, 5, 9]

Вопросы для самопроверки

1. Чем философия истории отличается от философии, с одной стороны, и от исторической науки, с другой?
2. Какие подходы в осмыслении исторического развития человечества выработаны в философии истории?
3. Каковы основные черты формационного подхода?
4. Что, по Марксу, лежит в основе развития общества?
5. В чем смысл истории, по Марксу?

Тема 6. Концепции «индустриализма» и «постиндустриализма»

1. Концепции «Стадий экономического роста» и индустриального общества.

2. Концепции постиндустриального общества.
3. Концепция информационного общества.

Методические рекомендации к изучению темы

1.Во второй половине XX в. стали появляться концепции, которые продолжили линию социального эволюционизма в западной социальной философии. Студентам рекомендуется провести анализ данных концепций.

Серию социально философских исследований второй половины XX в., получивших известность под общим названием «индустриализма», открывает американский социолог, экономист и историк У. Ростоу. Следует обратить внимание на то, что всю историю человечества У. Ростоу делит «на пять стадий экономического роста», исходя из идеи о решающей роли технико-экономических показателей в развитии общества. Это такие стадии, как: 1) «традиционное общество»; 2) «переходное общество»; 3) «стадия сдвига»; 4) «стадия зрелости»; 5) «эра высокого массового потребления товаров длительного пользования». Ведущий сектор экономики – сфера услуг и производство предметов потребления.

Концепция «индустриального общества» была разработана известным французским философом и социологом Р. Ароном в 60-х гг. В данной концепции Р. Арон показывает, что социальный прогресс характеризуется переходом от прежнего отсталого «традиционного общества» к передовому, промышленно развитому «индустриальному» обществу. Следует обратить внимание на то, что указанное Р. Ароном сходство между индустриальным обществом западного типа и моделью социалистического общества, привело его к выводу о формировании «единого индустриального общества».

2. В 70-х годах появляется точка зрения, согласно которой научно-технический прогресс приводит к трансформации прежнего «индустриального» общества в некоторое качественно иное «постиндустриальное» общество. Одним из первых обратил внимание на эту тенденцию американский экономист, социолог З. Бжезинский. Он определил новое «постиндустриальное общество» как «технотронное». Студентам необходимо рассмотреть ряд качественных особенностей «технотронного общества», отмеченных З. Бжезинским, а также провести анализ концепции «постиндустриального общества» американского социолога Д. Белла.

В 80-е годы появляются новые идеи об основных тенденциях развития общества, которые в наиболее концентрированной форме были представлены американским социологом О. Тоффлером в работе «Третья волна» (1980 г.). В развитии общества Тоффлер выделяет три этапа: «сельскохозяйственную цивилизацию», именуемую «первой волной»; «индустриальную цивилизацию» («вторую волну»), на смену которой в конце XX в. приходит «третья волна» – грядущая цивилизация. В своей работе А. Тоффлер сопоставляет «вторую и третью» волны. Необходимо подчеркнуть, что это сопоставление проводится им на различных уровнях: экологическом, научно-техническом,

экономическом, социально-организационном, политическом, культурном. Тоффлер выделяет такие черты нового общества, как гармонизация отношений с природой и размывание границ между рабочим и свободным временем.

Важно подчеркнуть, что и философия истории марксизма, и социально–философские концепции «индустриализма» основаны на признании того, что источником прогресса человечества и его измерителем выступает развитие материального производства. Степень совершенства материального производства, развитость его форм и методов лежит в основе оценок того или иного общества как в марксизме, так и в современных западных «индустриальных» концепциях.

3.В рамках данного вопроса студентам необходимо рассмотреть концепцию «информационного общества», появившуюся в середине 80-х годов. Важно отметить заслугу создателей этой концепции Дж.Несбита и И.Масуды, которая состояла в предвидении того, что новый «информационный» этап научно-технического прогресса станет реальностью и изменит многие стороны социальной жизни.

«Информационное общество» характеризуется компьютеризацией самых различных областей социальной жизни, появлением новых информационных технологий и новых видов деятельности. Основным стратегическим ресурсом общества становится информация, что революционным образом воздействует на все стороны жизни общества. Правила, умение и талант, необходимые для раскрытия, овладения, производства, сохранения и использования информации являются ныне самым большим достоянием человечества. Именно интеллектуальный капитал определяет судьбу наций.

Следует заметить, что теории индустриального и постиндустриального общества находятся в рамках социального эволюционизма, поскольку они предполагают прохождение обществом определенных стадий на основе технических и технологических нововведений. Технологические перевороты влекут за собой перевороты в других сферах общественной жизни. Согласно этим концепциям история – это процесс движения от доиндустриального общества к обществу индустриальному и постиндустриальному. Социальные изменения имеют линейный характер.

Основная литература: [1, 2, 3, 5, 9]

Вопросы для самопроверки

1. Что общего и в чем разница между формационной концепцией Маркса и концепциями индустриализма и постиндустриализма?
2. Является ли общество «третьей волны», рисуемое Тоффлером, новым этапом в развитии общества или речь идет о постиндустриализме?
3. Какие социальные проблемы характерны для постиндустриального общества?

4. Какие качества работника будут особенно востребованы в информационном обществе?

Тема 7. Цивилизационный подход к истории человечества

1. Многозначность понятия «цивилизация». Основные черты цивилизационного подхода.
2. Теория локальных цивилизаций А.Тойнби.
3. Теории единой цивилизации и столкновения цивилизации.

Методические рекомендации к изучению темы

1. Рассмотрение данного вопроса целесообразно начать с определения понятия «цивилизация». Сам термин «цивилизация» (от лат. civilis – гражданский) был введен в XVIII в. французским экономистом В.Мирабо.

В XIX в. Л. Морган предложил схему истории человечества, в которой выделялись три этапа развития общества: дикость, варварство и цивилизация. Морган считал, что описанные им этапы развития универсальны и характерны для истории каждого народа.

Одним из широко распространенных пониманий термина «цивилизация» стало отождествление его с понятием «культура». При этом по мере изучения разнообразных культур сложился локально-исторический подход, представители которого рассматривают цивилизации как качественно различные локальные исторические образования, как особые социокультурные феномены, ограниченные пространственно-временными рамками.

В XIX в. этой точки зрения придерживался известный русский социальный мыслитель и культуролог Н.Я. Данилевский (1822-1885), который рассматривал цивилизации как определенные «культурно-исторические типы общества», существующие в рамках обособленных локальных образований. Каждая локальная цивилизация, считал он, проходит в своем развитии следующие этапы: становления самобытности, юности (формирование политических институтов), зрелости и упадка.

Цивилизационный подход, в отличие от социального эволюционизма, рассматривает историю как процесс возникновения и гибели отдельных цивилизаций; тем самым обращается внимание на цикличность и повторяемость в истории. Подчеркивается неповторимость и уникальность отдельных культур и цивилизаций.

2. Студентам предстоит проанализировать теорию локальных цивилизаций А. Тойнби (середина 70-х XX в.). Тойнби – религиозный мыслитель, что существенным образом сказывается на его видении истории.

Позицию Тойнби можно охарактеризовать как культурологический плюрализм – убеждение, что человеческая история представляет собой совокупность дискретных единиц социальной организации («цивилизаций»). Ка-

ждая из них проходит свой уникальный путь и имеет своеобразную систему ценностей, вокруг которой складывается вся ее жизнь.

Центральным в концепции Тойнби является понятие цивилизации, под которым понимается замкнутое общество, характеризующееся набором признаков: во-первых, религией и формой ее организации и, во-вторых, территориальностью, степенью удаленности от того места, где данное общество первоначально возникло.

Согласно Тойнби, рост цивилизации состоит в прогрессивном внутреннем ее самоопределении или самовыражении, в переходе от более грубой к более тонкой религии и культуре.

Развитие цивилизации определяется «законом вызова и ответа». Студентам следует выяснить, что понимается по «вызовом» и как их классифицирует Тойнби. Все вызовы делятся на вызовы природного мира и вызовы человеческой среды. Существует пять типов вызова: вызов сурового климата, вызов новых земель, вызов неожиданных ударов со стороны внешнего человеческого окружения, вызов постоянного внешнего давления и вызов ущемления, когда общество, утратив нечто жизненно важное, направляет свою энергию на выработку свойства, возмещающего потерю.

Рост цивилизаций – дело рук творческих личностей или творческих меньшинств. Надлом и распад цивилизации не являются следствием действия внешних сил. Цивилизация погибает, как правило, от собственных рук, когда правящее меньшинство утрачивает творческие силы и энергию, а с ними и способность отвечать на исторический вызов.

Необходимо отметить, что, разбивая историю на отдельные, локальные цивилизации, Тойнби вместе с тем пытается восстановить идею единства мировой истории, придавая этому единству религиозный смысл. Иными словами, прогресс в истории – это все большее приближение человека к богу. Образование мировых религий – высший результат исторического развития. Это то, в чем яснее всего воплотились культурная преемственность и духовное единство человечества, пробившие себе дорогу через замкнутость отдельных цивилизаций.

3. В рамках данного вопроса студентам предстоит рассмотреть теории единой цивилизации и столкновения цивилизаций. В основе социокультурной теории единой цивилизации лежит либеральная идея о постепенном движении всех стран мира к единому политическому, социальному и экономическому строю – либеральной демократии. Данная теория обращает внимание на все более могучие силы, которые объединяют страны мира в единое сообщество, усиливают их взаимозависимость, стирают границы между ними. Это силы интернационализации экономики, перерастающие в ее глобализацию. В результате подобной модернизации черты современного индустриального и постиндустриального общества, которые впервые появились на западе, становятся чертами, присущими и остальным странам мира. Теория единой цивилизации предполагает, что модернизация идентична вестернизации (то есть копированию западной культуры другими странами). Однако,

следует отметить, что опыт России и Японии в предыдущие века и стран Восточной и Юго-Восточной Азии в XX в. говорит о том, что модернизация может идти без вестернизации, и страны в ходе ее могут сохранять свою принадлежность к другой, не западной цивилизации.

Другое, не менее интересное видение современности и ближайших перспектив развития человечества предложил американский ученый С.Хантингтон. В 1996 г. он опубликовал книгу «Схватка цивилизаций и переустройство мирового порядка». Центральная мысль этой книги состоит в том, что после периода «холодной войны» будущее человечества будет определять конфронтация цивилизаций.

Основная литература: [1, 2, 3, 4, 5, 9]

Вопросы для самопроверки

1. Каковы основные черты цивилизационного подхода?
2. Что понимается под цивилизацией?
3. Как соотносятся понятия «этнос» и «цивилизация»?
4. Каковы критерии развитости цивилизации, по Тойнби?
5. Какой вызов сыграл решающую роль в становлении нашей цивилизации, по Тойнби?
6. На какой основе, по мнению Ф.Фукуямы, возможно создание единой цивилизации?
7. Согласны ли Вы с прогнозом С.Хантингтона о неизбежности «схватки» цивилизаций?

Тема 8. Проблема человека в философии

1.Образ человека в современной культуре.
2. Биологическое и социальное в человеке.
3. Смысл человеческого бытия.

Методические рекомендации к изучению темы

1. Человек – главный предмет философского исследования. Раздел философии, изучающий проблему человека, называется философской антропологией. Антропология рассматривает человека с точки зрения его «природы», «сущности». Необходимо отметить, что попытки найти такую «сущность» или «природу» человека предпринимались постоянно, но только теперь становится ясно, что человек – это существо, которое сам себя определяет, и у него нет какой-то одной, раз и навсегда данной ему «природы», так же как и неизменной «сущности». И то и другое – исторически изменяющиеся характеристики человека.

Важно подчеркнуть, что человек – существо деятельностное. Деятельностная сущность человека была осознана немецкой классической филосо-

фией. Затем данный подход к человеку был развит марксизмом. Такая важнейшая характеристика человека как свобода, была проанализирована в различных философских учениях, в частности в экзистенциализме.

Следует отметить, что частные проявления человека изучают многие науки и только философия пытается создать некий целостный, интегрированный образ человека, найти его главные бытийные характеристики. В связи с этим студентам предстоит, как бы подвести итог, выделить основное в современных представлениях о человеке, где, прежде всего, необходимо сосредоточиться на таких философских направлениях как экзистенциализм и постмодернизм.

2.В понимании сущности человека очень важным является вопрос о соотношении в нем биологической и социальной сторон. Человек есть продукт длительного развития живой природы, в то же время человек рождается и живет в обществе и поэтому он существо социальное.

В настоящее время можно выделить 3 основных подхода в решении проблемы человека.

– *Натурализаторский (*или биологизаторский*),* рассматривающий человека как биологическое существо, поступки и поведение которого определяются природными качествами и задатками.

– *Социологизаторский,* в котором человек рассматривается как часть общества, а его поведение – как зависимое от характера социальных отношений, от места человека в обществе, его принадлежности к определенному социальному слою. (Маркс определял человека как совокупность общественных отношений).

– Экзистенциально-философский – человек рассматривается в его глубинных характеристиках.

Первые два направления – попытка дать научное объяснение человеческим поступкам, т.е. найти их причинную обусловленность.

Третье направление рассматривает человека как существо свободное. Уже Кант рассматривал человека как существо, принадлежащее двум мирам: миру природы и миру свободы. Экзистенциализм рассматривает человека как существо неопределенное, не имеющее собственной сущности. Человек всегда выходит за пределы, он всегда стремится к чему-то другому. Он определяется будущим: ставит перед собой цели и изменяет себя в процессе их достижения. Человек является проектом. Он обречен на свободу, т.е. перед ним всегда существует выбор, но что бы он ни выбрал, он всегда будет не удовлетворен.

В последние десятилетия XX в. значительно усилилась биологическая направленность в осмыслении проблемы человека. Появилось новое научное направление – социобиология. Его основоположником считается Э.Уилсон.

В своих исследованиях Уилсон пришел к выводам:

1) Воспитание в социальной среде – попытка «обуздать» нашу биологическую природу

2) Существует зависимость форм социального поведения человека от его генетических основ.

С такой точки зрения, человек является объектом биологического знания. Не отвергая многие достижения современной социологии, следует иметь ввиду, что биологическое и социальное в человеке находится между собой в тесной взаимосвязи. Именно в процессе социального взаимодействия формируются важнейшие качества человека – его сознание и самосознание, профессиональные и личные характеристики.

3.Следует отметить, что проблема смысла человеческого бытия, человеческой жизни – одна из самых сложных и неоднозначно решаемых. Речь идет, прежде всего, о личностной характеристике отношения к жизни, об осознании целей и задач человеческого существования, о соотнесении индивидом своей позиции с выработанной в обществе системой ценностей, о степени включения себя в социальную жизнь.

При рассмотрении всех этих вопросов наряду с позицией, признающей тот или иной вариант смысла человеческого бытия, можно выделить и позицию, отрицающую его существование вообще. Студентам рекомендуется рассмотреть отпечаток этой позиции в философии экзистенциализма.

Смысл жизни – цель, которой человек подчиняет свою жизнь, причем цель главная, для которой все другие цели выступают в качестве средств. Собственно проблема смысла жизни – проблема ценностей, проблема того, что есть высшее благо для человека.

Особое внимание необходимо уделить рассмотрению данной проблемы в религиозных воззрениях. Религиозное понимание смысла человеческого бытия связано не с эмпирической внешней жизнью мира, а с областью внутренней, духовной жизни человека. Важно отметить, что упор на нравственное содержание смысла человеческой жизни был сделан В. С. Соловьевым, который соединил религиозный подход к рассматриваемой проблеме с философскими исканиями.

Проводя анализ проблемы смысла человеческого бытия, нельзя не обратиться к творчеству Л. Н. Толстого. Облекая свой ответ в религиозные, христианские формы, Толстой выделил особую в решении данного вопроса роль любви в высоком смысле этого слова, любви, присущей верующему человеку.

Принципиально иное понимание этой проблемы было предложено в марксистской философии. Марксизм видит подлинный смысл человеческого бытия в максимальном содействии решению задач общественного развития, в созидательном труде, формирующем предпосылки всестороннего развития личности.

Следует отметить, что каждый человек обладает индивидуальным пониманием смысла жизни вместе с тем можно выделить некоторые общие компоненты понимания смысла жизни. К ним, например, относят любовь и семью, любимое дело, труд, борьбу, служение на благо другим, поиск удовольствия, познание мира и.т.д. Необходимо учитывать и то, что общество

всегда «программирует» индивида на достижение тех целей, которые представляются наиболее важными. Внутри общества всегда сосуществовали различные представления о смысле жизни. Они были различны у разных сословий, социальных групп, классов и.т.д.

Важно подчеркнуть то, что большинство людей не задумываются о смысле жизни, а живут теми «программами», которые вложило в них общество. Таким образом, смысл жизни каждого человека состоит в том, чтобы «проснуться», перестать быть роботом, выполняющим то, что «принято», осознать себя и стать творцом, навести порядок в своей душе и мире.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. В чем заключается особенность философского подхода к изучению человека в отличие от частных наук?
2. Какое из философских учений в качестве важнейшей характеристики человека выделяет свободу?
3. Можно ли рассматривать социобиологию Уилсона как попытку «примирения» социологизаторского и биологизаторского подходов в понимании сущности человека?
4. В каком из философских направлений вопрос о смысле жизни считается основным вопросом философии?
5. Задумывались ли Вы о смысле собственной жизни?

Тема 9. Глобальные проблемы современности

1.Многообразие глобальных проблем, их общие черты и классификация.
2.Пути преодоления глобальных кризисных ситуаций.

Методические рекомендации к изучению темы

1. Во второй половине XX в. на нашей планете возникли такие условия, которые поставили человечество перед угрозой подрыва самих основ его существования. Род человеческий впервые в своей истории столкнулся с возможностью его общей гибели. Под вопросом оказалось само существование жизни на Земле, ибо уничтожение биосферы стало технически возможным.

Важно отметить, что глобальные проблемы, (при всем их многообразии и внутренних различиях) обладают общими чертами, которые студентам и предстоит выяснить.

Далее необходимо провести классификацию глобальных проблем по следующим признакам: 1) группа актуальных политических проблем; 2) экологическая группа глобальных проблем; 3) проблемы ресурсосбережения; 4) группа актуальных социальных и медико-биологических проблем; 5) непре-

кращающийся разрыв между экономически развитыми странами и между странами «третьего мира»; 6) международный терроризм.

2.Важно отметить, что во второй половине XX в. обострился интерес исследователей к проблемам будущего человечества. В связи с этим все возрастающую роль в сфере социальных наук стала играть футурология, то есть совокупность научных представлений о перспективах развития человеческой цивилизации. Так, например, с начала 70-х годов широкую известность получили глобальные прогнозы, оформлявшиеся в виде докладов Римскому клубу. Но основоположником и «идейным отцом» глобального прогнозирования с использованием математических методов и компьютерного моделирования по праву считается Дж. Форрестер. Следует также отметить деятельность стран мирового сообщества в рамках Программы ООН по окружающей среде (ЮНЕП), которая помогает укреплению международного сотрудничества.

Чрезвычайно важно рассмотреть, так называемую, концепцию устойчивого развития, появившуюся в конце XX столетия, которая поставила проблему перехода к глобальному управлению отношениями между обществом и природой, а так же отношениями между людьми и их сообществами. Эта концепция опирается на достижения всех наук о природе и обществе и является в силу это интегральной.

Основная литература: [1, 2, 3, 5, 9]

Вопросы для самопроверки

1. Каковы критерии выделения глобальных проблем современности?
2. Почему отсталость стран «третьего мира» рассматривается как глобальная проблема, а не региональная или локальная?
3. Можно ли выделить «главную» глобальную проблему современности?
4. Что является предметом исследования в футурологии?
5. Каковы пути решения глобальных кризисных ситуаций?

ТЕМЫ ЗАНЯТИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Тема 1. Изменение предмета философии в ходе ее исторического развития

Методические рекомендации к изучению темы

Рассматривая данный вопрос, следует отметить, что определение предмета философии – одна из основных проблем философии, что свидетельствует о ее непрерывном развитии.

Представления о предмете философии исторически изменялись. Аристотель, например, полагал, что философия – это высшая умозрительная наука, «мать» всех наук. Все частные науки выделяют какую-либо часть сущего, а философия изучает сущее само по себе, это учение о первых началах и причинах мира. В средневековье философия – служанка богословия, ее методы и средства использовались для создания и обоснования христианского вероучения. В Новое время философия – «царица наук», сводящая воедино результаты, полученные другими науками.

Студентам рекомендуется обратить внимание на то, что в советскую эпоху марксистско-ленинская философия трактовалась как наука о наиболее общих законах развития мира. Такое понимание предмета философии отражало особенности советской системы, и универсальным его назвать нельзя. Дело в том, что далеко не все философы, философские школы и течения занимаются изучением этих законов, однако, не перестают при этом быть философскими.

Разнообразие философских течений, как существовавших в прошлом так и современных, заставляет переосмысливать ее предмет. В отличие от науки, где каждый ученый продолжает идеи своих предшественников, опираясь на их достижения, развитие философии происходит по-другому. Каждый философ вместо того, чтобы «достраивать здание», разваливает его стены и начинает закладывать фундамент вновь. Каждый мыслитель в своем учении выражает свое видение мира, строит свою систему мироздания, и это – единственно общее, что объединяет философию Аристотеля, Платона, Гегеля, Маркса или Сартра. Не случайно, развитие философских школ и направлений представляют в образе дерева, ветви которого множатся и разделяются.

Важно подчеркнуть, что любое философское учение создает особое целостное представление о мире и человеке, о месте человека в мире. Философия – это, прежде всего, определенный тип мировоззрения, отличающийся от других рядом особенностей. Мировоззренческий статус философии утвердился в XX в.

Далее студентам следует уяснить, что философия, изучая всеобщее в системе отношений «человек-мир», представляет собой огромный массив знания со сложной структурой.

В зависимости от того, какие именно отношения и проблемы исследуются, как правило, выделяют следующие разделы философского знания:

фундаментальные (что есть мир и как он устроен?) – онтология;

познавательные (как познается мир?) – гносеология;

ценностные (что такое хорошо и что такое плохо?) – аксиология;

этические (что такое добро и зло?) – этика;

эстетические (что такое красивое, безобразное?) – эстетика;

практические (как с этим миром взаимодействовать?) – праксеология.

Философская антропология сосредотачивается на специфике человеческого бытия; философия истории пытается постичь смысл и направление исторического процесса; социальная философия исследует закономерности развития общества; логика представляет собой учение о формах и законах правильного мышления.

Перечень философских дисциплин увеличивается, появляются новые разделы – философия науки, философия техники, философия искусства и т.д., что свидетельствует о непрерывном росте философского знания.

Функции философии многообразны. Студентам следует сосредоточиться на наиболее важных – мировоззренческой, гносеологической, методологической, аксиологической, интегрирующей, культурной и выяснить, в чем они заключаются.

Основная литература: [1, 2, 3, 9]

Вопросы для самопроверки

1. Как изменялся предмет философии?
2. Является ли философия наукой?
3. Назовите основные разделы философского знания.
4. Что возникло раньше – философия или наука?

Тема 2. Натурфилософия античности. Учение о первоначалах мира. Античная атомистика. Учение о стихийной диалектике в античной философии

Методические рекомендации к изучению темы

Античная философия представляет собой совокупность философских учений, разрабатывавшихся в древнегреческом и древнеримском обществах с конца VII в. до н.э. до VI в. н.э.

Рассматривая данную тему, студентам следует обратить внимание на то, что первые греческие философы в основном размышляли о природе, поэтому начальный этап античной философии (приблизительно с VII по V вв. до н.э.) является натурфилософским (от лат. natura - природа). Важно подчеркнуть своеобразие древнегреческой натурфилософии – ее созерцательный

характер, космоцентричность, синкретичность, зарождение диалектических воззрений и др.

Основная проблема античности – поиски первоначала, источника происхождения и изменения всех вещей. Постановка этого вопроса означала поворот в истории мысли, так как признавалось, что не боги определяют ход событий, а безличная причина, внутренне присущая самому миру. Античные философы усматривали это первоначало в чувственно воспринимаемых вещах. Студентам следует выяснить специфику понимания первовещества Фалесом, Анаксименом, Анаксимандром, Эмпедоклом, Демокритом.

Важно обратить внимание на противоположность материалистического и идеалистического подходов в трактовке первоначала, проявившихся на раннем этапе становления философии. Идеалистические воззрения обнаруживаются у Пифагора, по мнению которого сущность вещей определяется числом – абстрактным, идеальным понятием. Демокрит, напротив, в атомистической концепции дает материалистическую трактовку бытия.

Следует отметить, что древнегреческая философия была *первой формой диалектики* – наивной и стихийной, зафиксировавшей развитие и всеобщую связь явлений на уровне созерцания.

Основы диалектики как учения о развитии заложил Гераклит (530 - 470 гг. до н.э.). В качестве первовещества он полагал огонь – наиболее изменчивое и подвижное из всех наблюдаемых явлений природы.

Основные положения учения Гераклита:

Все существующее изменяется: «Нельзя в одну и ту же реку войти дважды», «Все течет и ничто не становится».

Всякая вещь и всякое свойство не просто изменяются, а переходят в свою противоположность: «В мире нет ничего неподвижного: холодное теплеет, теплое холодеет, влажное высыхает, сухое увлажняется».

Эта борьба противоположных начал и есть причина изменения. Она – «отец всего, царь всего». В борьбе противоположностей обнаруживается их внутреннее тождество: «Расходящееся согласуется с собой. Путь вверх и вниз – это один путь».

Переход явления из одного состояния в другое через борьбу противоположностей, в основе тождественных друг другу, Гераклит назвал единым, общим для всего существующего законом, Логосом. Огонь и Логос суть одно и то же. «Огонь разумен и является причиной управления всем». «Мир, единый…не создан никем из богов и никем из людей, а был, есть и будет вечно живым огнем, закономерно воспламеняющимся и закономерно угасающим».

Если Гераклит подчеркивал изменчивость бытия, то Парменид, наоборот, подчеркивал неизменность и постоянство бытия, постижимого только разумом.

Следует подчеркнуть, что натурфилософские концепции первых греческих философов во многом были умозрительны, не свободны от фантастических представлений и основывались на антропоморфных и социоморфных аналогиях. Как правило, они опирались на простые наблюдения и жизненный

опыт. Но, главное, это была попытка объяснить мир из него самого, не прибегая к помощи богов.

Тема 2.1 Классическая древнегреческая философия. Сократ. Платон. Аристотель.

Методические рекомендации к изучению темы

Классический этап – расцвет древнегреческой философии – охватывает период приблизительно с середины V в. до н.э. до конца IV в. до н.э. Он связан с творческой деятельностью таких крупнейших мыслителей как Сократ, Платон и Аристотель.

Студентам следует кратко охарактеризовать историческую ситуацию в Афинах в V – IV вв. до н.э.

Сократ – одна из ярчайших фигур античности, родоначальник антропологии. Сократа не интересовало устройство природы, его учение имеет этическую направленность и сосредоточено на человеке. Философия для Сократа – основание определенного образа жизни, мудрость, позволяющая понять человеку, как жить, как отличить истинные ценности от ложных, познать суть добродетели.

Студентам необходимо выяснить, в чем заключался этический рационализм Сократа, проанализировать особенности его диалогического метода. Основная заслуга Сократа состоит в том, что он совершил переворот в традиционной системе ценностей, показав, что истинные ценности не те, которым обычно поклоняются люди – богатство, слава, красота, а разум, знание, умеренность и справедливость.

Платон – величайший философ античности, автор первой рационализированной системы объективного идеализма.

Рекомендуется раскрыть особенности онтологических воззрений Платона – удвоение мира на видимый мир изменчивых предметов и невидимый мир идей. Необходимо отметить связь учения Платона с реальными проблемами жизни полиса. Учение об идеях создается Платоном для обоснования того, что у мира есть прочная основа, его текучесть и изменчивость – иллюзия. Человек должен познать это вечное и неизменное бытие и строить свою жизнь, опираясь на это знание. Тогда можно преодолеть противоречивость мнений и прийти к согласию и гармонии в жизни.

Следует рассмотреть представления Платона о душе, его гносеологические воззрения и теорию идеального государства.

Учеником Платона, его критиком и продолжателем традиций является выдающийся античный мыслитель Аристотель.

Немаловажно обратить внимание на энциклопедичность таланта Аристотеля, систематизировавшего накопленное в древнем мире знание и предложившего первую классификацию наук. Необходимо охарактеризовать вклад Аристотеля в развитие логики, его геоцентристскую космологию.

Аристотель подверг критике платоновскую теорию «идей», подметив в ней целый ряд противоречий, и пришел к выводу, что Платон исследовал не подлинные сущности. Согласно Аристотелю, ими являются отдельные предметы, единичное бытие. Каждая вещь – единство нематериальной формы и материи. Материя – это некоторый субстрат, возможность вещи, а форма – сущность вещи, которая переводит ее из возможности в действительность. Материя оформляется, и таким образом возникают качественно отличные друг от друга вещи.

Всякая вещь имеет четыре причины: сущность (форму), материю (субстрат), действие (начало движения), и цель (то, ради чего). Последние две составляющие определяются формой, она определяет переход материи-субстрата, возможности к действительно существующим вещам. Различные формы находятся в отношении соподчинения, существует форма форм – перводвигатель, Бог, который является причиной всех изменений.

Студентам следует обратить внимание на то, что начиная с Платона и Аристотеля, философия становится развернутым, последовательным учением, создаваемым в понятийной форме с помощью логических средств.

Тема 2.2 Философия эпохи эллинизма. Скептицизм. Эпикуреизм. Стоицизм. Скептическая установка в теории познания

Методические рекомендации к изучению темы

Раскрывая данный вопрос, важно отметить, что в эллинистический период важное место занимает проблема смысла человеческого существования. Ее решением заняты эпикурейцы и стоики.

Студентам следует выделить составные части учения Эпикура – логику, физику и этику и показать их роль в достижении счастливой жизни.

Эпикур полагал, что человек по природе своей стремится к удовольствию, поэтому задача философии – подвергнуть удовольствие рациональному исследованию. Студентам предстоит выяснить, как Эпикур понимал удовольствие, что считал необходимым для его достижения.

С точки зрения стоиков (Зенон из Китиона, Клеанф, Хрисипп, в римский период – Сенека, Эпиктет, Марк Аврелий), положение человека в мире трагично, ибо он не в силах противостоять необходимости, судьбе. Стоики видели выход из этой ситуации в невозмутимости духа, спокойствии, мужестве, проистекающим из принятия мира таким, каков он есть. Обстоятельства внешней жизни, по мнению стоиков, ничего не значат для внутреннего состояния человека: безразличие, бесстрастие – их этический идеал. Счастье человека зависит не от его конкретного положения в мире, а от того, как он воспринимает и оценивает свою включенность в мир.

Следует обратить внимание на методы изменения сознания – «духовные упражнения», которые применялись в стоицизме для того, чтобы по-другому относиться к окружающему миру, освобождаться от мелочности,

корысти и эгоизма. В этике поздних стоиков появляются новые мотивы, сближающие их с христианами – смирение, покорность, поиск спасения свыше.

Скептики (Пиррон, Карнеад, Клеанф) выражали сомнение во всем. С их точки зрения, подлинное знание в принципе недостижимо. Все в мире текуче, относительно, устойчивый смысл отсутствует, поэтому каждый волен жить, как он хочет. Знающий многое не придерживается однозначного мнения. Мудрецу-скептику лучше молчать, молчание и есть ответ на поставленные вопросы.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Каковы главные черты античной философии?
2. Чем античная философия отличается от предшествующей – мифологической – формы мировоззрения?
3. Что такое диалектика?
4. Каковы отличительные черты античной диалектики?
5. Какие проблемы стали центральными в классический период?
6. Основателем какого философского направления считается Сократ?
7. Как можно охарактеризовать философскую позицию Платона?
8. Каков вклад Аристотеля в развитие научного знания?
9. С чем связано усиление внимания к смысложизненной проблематике в эллинистической философии?
10. В чем суть скептической установки в теории познания?

Тема 3. Средневековая философия, ее характерные черты. Патристика. Основные проблемы средневековой схоластики

Методические рекомендации к изучению темы

При рассмотрении данной темы следует подчеркнуть тесную связь средневековой философии с христианской религией и кратко охарактеризовать условия зарождения и распространения христианства.

Христианство противопоставляет себя предшествующей духовной культуре и создает совершенно новые представления о человеке и мире.

Студенты должны сравнить созданные античной философией представления о мире, человеке и его ценностях с христианскими.

В античной философии мир понимался как гармонично устроенный космос, бог – безличное упорядочивающее начало, а человек – часть мирового целого, разумное существо.

В центре христианского мировоззрения находится всемогущий и всевидящий Бог, творящий мир из ничего и создающий человека по своему об-

разу и подобию. Человек рассматривается как существо двойственное: с одной стороны, духовное, стремящееся к единению с Богом, с другой – телесное, наделенное природными потребностями. Воля ставится выше разума и выходит на первый план в характеристике Бога и человека: благодаря воле Бога создан и существует мир, человек тоже обладает свободной волей, может выбирать между долгом и удовольствиями. Христианство утверждает новые ценности – терпение, сострадание, самопожертвование, вера, надежда, любовь. Не познание и созерцание составляют смысл человеческого существования, а преодоление греховности, очищение, воплощение заветов Бога в жизнь.

Далее студентам рекомендуется раскрыть суть основополагающих принципов средневековой философии – теоцентризма, креационизма и провиденциализма.

В развитии средневековой философии выделяют два этапа – патристику (приблизительно с IV – V вв. до VIII в.) и схоластику (с VIII в. по XIV – XV вв.).

Патристика – это направление христианской мысли, в котором использовались достижения античной философии для защиты и обоснования христианского вероучения. Она начала свое развитие в пору распада античного мира и завершила его уже в рамках философии феодального общества. Особенно велико было влияние на раннюю патристику неоплатонических идей. Крупнейший представитель этого периода – Аврелий Августин, один из первых великих теологов, связавших неоплатонизм и христианство.

Студентам следует обратить внимание на личный характер истолкования Бога у Августина, его взгляды на взаимоотношение души и тела, воли и разума, превосходство веры над знанием, субъективистскую трактовку времени.

Если перед патристикой стояла задача создать систематизированное учение, то схоластика этой конструкцией уже располагала, и ее цель заключалась в том, чтобы ее упорядочить и сделать доступной для широких слоев необразованных масс. Схоласты занимались систематизацией христианской философии, разъяснением, интерпретацией религиозных догматов и их популяризацией. Метод схоластики заключался в том, чтобы доказать истину, данную в Откровении, посредством разума.

Схоластическое философское мышление было сосредоточено в основном на трех проблемах: на споре о природе общих понятий – «универсалий», на поиске способов рационального обоснования существования Бога и на проблеме соотношения веры и разума.

Студентам следует охарактеризовать позиции реалистов и номиналистов в полемике о соотношении общего и единичного, назвать их представителей.

Большинство схоластов полагало, что можно логически обосновать существование Бога. Студентам рекомендуется рассмотреть аргументы Ансельма Кентерберийского и Фомы Аквинского в пользу существования Бога.

Тема 3.1 Проблема соотношения веры и разума в средневековой философии

Методические рекомендации к изучению темы

Вопросы о взаимоотношении веры и разума, христианского откровения и греческого мышления, религиозности и мирской мудрости занимали большое место в христианской теологии и философии.

Некоторые христианские теологи, в частности, Тертуллиан полагали, что христианская вера и греческое мышление несовместимы: «Верую потому, что абсурдно».

Другие допускали наличие общей сферы для разума и веры, подчеркивая, однако, преимущества веры над разумом. Августин, например, считал, что благодаря вере становится возможным мышление. Он формулирует принцип «Верую, и потому понимаю» (V в.).

В XIII в. Фома Аквинский предлагает концепцию двух истин как попытку соотнести знание (философию и конкретные научные дисциплины) и веру (христианско-католические догматы). Наука получает истины из опыта и разума, ее истины требуют рационального обоснования, разум может ошибаться и поэтому должен прислушиваться к религиозному вероучению. Религия, основанная на вере, источником знания имеет откровения Бога и священное писание. Некоторые догматы доказуемы (например, существование Бога), некоторые требуют только веры, т.к. сверхразумны (творение мира из ничего, триединство, последний суд). Это знание более достоверное, представляет собой высшую мудрость, «подвластную только уму бога, а не человека», т.к. вера опирается на абсолютную достоверность Бога. Поэтому наука должна согласовывать свои выводы с вероучением, по крайней мере, не вторгаться в область веры.

Важно подчеркнуть, что объективно теория двух истин способствовала развитию науки, разрешив ей существовать, хотя и под эгидой веры.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Какова главная особенность средневековой философии?
2. В чем отличие схоластики от патристики?
3. Чем обусловлен исключительно логический характер схоластических рассуждений?
4. Можно ли с помощью рациональных аргументов обратить человека в религиозную веру?

Тема 4. Философия и наука эпохи Возрождения

Методические рекомендации к изучению темы

При рассмотрении данной темы важно отметить переходный характер эпохи Возрождения – от средневековья к Новому времени, кратко охарактеризовать изменения в социальной, экономической и духовной жизни Западной Европы, произошедшие в XIV-XV вв. Имеет смысл назвать крупнейшие естественнонаучные достижения эпохи Ренессанса и выделить ее наиболее видных представителей в философии, науке, литературе и искусстве.

Следует заметить главную особенность эпохи Возрождения – обращенность к человеку, признание прав и интересов личности, утверждение ее творческой самостоятельности, принципов гуманизма и антропоцентризма.

Студентам рекомендуется рассмотреть основные идеи о человеке и мире, сформулированные мыслителями этого периода: понимание человека как творца самого себя; признание его уникальности и индивидуальности; формирование идеала Возрождения – всесторонне развитой личности; новое истолкование Бога – пантеизм.

Представления об эпохе Возрождения будут более полными, если студенты познакомятся с социально-политическими воззрениями Т.Мора, Т.Кампанеллы и Н.Макиавелли, выразившими противоположные подходы в понимании справедливого социального устройства.

В заключение необходимо подчеркнуть значение эпохи Возрождения в процессе освобождения от религии всех областей общественной жизни, формирования светской культуры и знания и возникновения новой системы ценностей, где на первом месте стоят человек и природа.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. В чем основное значение эпохи Возрождения?
2. Культура какого общества «возрождается» в эту эпоху?
3. Какие новые принципы утверждаются в эпоху Возрождения?
4. Какие научные открытия этого периода изменили представления о мире?
5. В чем выражается противоположность политических воззрений Т.Мора и Н.Макиавелли?

Тема5. Философия Нового времени

Методические рекомендации к изучению темы

Изучение темы необходимо начать с характеристики социально-экономической ситуации в Западной Европе в XVI-XVII в. Развитие нового – буржуазного – общества порождает изменения во всех сферах жизни и в соз-

нании людей. Становление капиталистических отношений, появление промышленного производства обусловили рост научного знания.

Возникает классическое естествознание, которое формирует иное представление о мире – механистическое, объективистское. Студенты должны охарактеризовать вклад Коперника, Галилея, Ньютона в развитие науки.

Важно отметить, что философия Нового времени находилась под непосредственным влиянием нового научного знания. Она решает две задачи: вместе с наукой создает новую механистическую картину мира и исследует те методы познания, которые обеспечивают достижение истинного знания.

Ф.Бэкон – английский философ, который первым сознательно поставил вопрос о методах истинного знания и стал родоначальником эмпиризма – течения, признающего решающую роль чувственного опыта в познании.

Бэкон по-новому осмыслил взаимоотношения науки и общества: наука – средство для улучшения жизни людей, а не путь к просветленному разумом пониманию природы, как считали древние греки.

Студентам рекомендуется обратить внимание на критику Бэконом традиционной философии за ее созерцательный характер, рассмотреть теорию «идолов», в которой выделяются предрассудки, мешающие достижению истины, и особенности индуктивного метода, ведущего к достоверному знанию, по мнению Бэкона.

Нельзя не отметить, что английский философ сделал чрезмерный акцент на эмпирических методах исследования, недооценив при этом роль рационального начала в познании.

У истоков рационалистической традиции Нового времени стоял Р.Декарт – французский философ и математик. С позиций рационализма, основным источником знания является разум, а наиболее надежным методом – дедуктивный. Декарт считал, что философия должна быть построена как математика, т.е. основываться на абсолютно достоверных утверждениях, подобных аксиомам, из которых выводится все остальное знание. Эти предельно ясные и отчетливые основания знания даны человеку как врожденные идеи. Опираясь на врождённые идеи и используя правильные приёмы рассуждения, разум может достичь достоверного знания во всех областях. Так же, как Бэкон Декарт считал, что целью науки является достижение господства человека над природой.

Следует обратить внимание на использование Декартом метода радикального сомнения, приводящего, в конечном счете, к изменениям в самосознании европейского человека, утверждению значимости индивидуального разума и пониманию познания как работы с идеями.

Далее следует остановиться на воззрениях английских философов Т. Гоббса и Дж. Локка, основоположниках западного либерализма. Студентам рекомендуется подчеркнуть, что в их работах было выражено представление об индивиде нового буржуазного общества, сформулированы наиболее значимые принципы его жизни, гарантирующие свободу и процветание как личности, так и общества. К этим качествам относятся индивидуализм, рационализм, свобода, толерантность, отношение к государству как гаранту соблю-

дения закона, уважение к частной собственности, уверенность в том, что успех зависит, прежде всего, от собственных усилий, поэтому человек сам ответственен за свои успехи и неудачи.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Почему XVII в. принято называть Новым временем? Что впервые возникает в жизни европейского общества в этот период?
2. Какая проблема считается центральной в философии Нового времени?
3. Родоначальником какого направления в теории познания считается Ф.Бэкон?
4. Как называется философская позиция Р.Декарта?
5. В чем суть идеологии либерализма?

Тема 6. Антропологические идеи французского Просвещения. Социальные взгляды Ф.Вольтера и Ж.-Ж. Руссо

Методические рекомендации к изучению темы

Просвещение – интеллектуальное и духовное движение, начавшееся в XVII в. в Европе. Оно явилось естественным продолжением гуманизма Возрождения и рационализма Нового времени, заложивших основы просветительского мировоззрения. Главные идеи эпохи Просвещения – отказ от религиозного миропонимания, освобождение от суеверий и невежества, обращение к разуму как к единственному критерию познания человека и общества.

В XVIII в. центром просветительского движения становится Франция. Его наиболее видными представителями были Ш. Монтескье, Ф. Вольтер, Д. Дидро, П. Г. Д. Гольбах, Ж. Ламетри, К.Гельвеций, Ж. Ж. Руссо.

Следует отметить, что разум является основным понятием в философии Просвещения. Он дает человеку понимание как общественного устройства, так и самого себя. И то и другое можно изменить к лучшему, усовершенствовать. Таким образом обосновывалась идея прогресса, который мыслился как необратимый ход истории от неведения к торжеству разума. Наивысшей и самой продуктивной формой деятельности разума считалось научное познание. Характерной особенностью эпохи Просвещения была ориентация на практическое использование достижений науки в интересах промышленного и общественного развития.

Проблема человека также занимает одно из центральных мест в философии Просвещения, провозгласившей равенство всех граждан перед законом, неотъемлемость прав каждого человека на жизнь, свободу и собственность.

Студентам целесообразно остановиться на анализе социальных взглядов Вольтера и Руссо. Основы социального равенства трактовались ими поразному. Вольтер понимал его как равенство политическое, равенство перед законом и правом. В отличие от Вольтера, Ж. Ж. Руссо считал несправедливым имущественное неравенство. По его мнению, оно является основой всякого другого неравенства и не соответствует «естественному состоянию человека», а представляет собой продукт развития цивилизации, результат разделения труда. Руссо выступал с критикой цивилизации, которая искажает природу человека, формирует «искусственные потребности». Поэтому, веря в прогресс, Руссо рассматривал будущее как абсолютизированное прошлое: будущее справедливое и разумное общество должно быть возвратом к «естественному состоянию человека». В этом состоянии все люди равны, нравственно не испорчены.

В целом следует отметить, что французское Просвещение представляет собой специфическую антропологию, утверждающую суверенитет личности, наделение ее правом мужественно пользоваться своим разумом и занимать активную позицию в мире. Их уверенность в силе человеческого ума достойна восхищения, но это не единственный путь решения проблем человеческого сознания и познания. Альтернативой их взглядам является немецкая классическая философия.

Тема 6.1 Этическое учение И.Канта. Категорический императив и нравственный долг человека

Методические рекомендации к изучению темы

В начале рассмотрения данной темы, следует обратить внимание на то, что немецкое Просвещение принято называть немецкой классической философией. Она развивается с 80-х годов XVIII в. до середины XIX в. Ее представители – *Иммануил Кант, Иоган Готлиб Фихте, Фридрих Вильгельм Шеллинг, Георг Вильгельм Фридрих Гегель, Людвиг Фейербах.*

Студентам рекомендуется выделить главные темы немецкой классической философии – проблемы человеческой деятельности и нравственности, проблемы диалектического метода, развития человеческого сознания, философские вопросы истории и естествознания. Немецкая классика довела до логического конца основные идеи новоевропейской рациональной философии. Она сохранила культ разума, но по-новому поставила гносеологические и этические проблемы.

Главным предметом своей философии Кант считал человека. В своем этическом учении Кант исходил из того, что человек – существо двойственное: и разумное, и природное. Как разумное существо, человек подчиняется долгу, моральным обязательствам, а как природное он следует естественным склонностям.

Важно подчеркнуть, что, по Канту, посредником между этими двумя сферами является воля. Любой поступок требует концентрации воли. Благо-

даря воле человек способен не подчиняться чувственным мотивам, а действовать вопреки им, сообразно долгу. Человек способен к свободной автономной мотивации, его поведение не определяется внешними причинами, как у животного.

Каждый человек, согласно Канту, имеет внутри нравственный закон – *категорический императив*. Кант считал, что это всеобщее правило поведения, обязательное для всех людей, и оно может быть построено подобно правилам поведения на формальной основе. Кант приводит несколько формулировок категорического императива, которые носят взаимодополнительный характер. Одна из них гласит: «Поступай лишь согласно тому правилу, следуя которому, ты можешь вместе с тем хотеть, чтобы оно стало всеобщим законом». Если намерение человека не может быть возведено в закон, то ему следует отказаться от такого поведения.

Другая формулировка категорического императива: «Поступай так, чтобы использовать другого человека всегда как цель и никогда лишь как средство» отражает влияние на Канта Руссо, который считал, что всякая личность – самоцель и ни в коем случае не должна рассматриваться как средство.

Следует подчеркнуть, что принципы кантовской этики делают возможным гражданское сообщество, жизнь по нормам права.

Тема 6.2 Абсолютная идея Г. Гегеля как смысл исторического процесса

Методические рекомендации к изучению темы

Г.Гегель – вершина немецкой классической философии, создатель системы диалектического идеализма – не согласился с субъективизмом Канта и явился продолжателем традиции рационализма. Он абсолютизировал Разум, подчинив ему весь мир.

Студентам следует охарактеризовать исходный пункт философских воззрений Гегеля – принцип тождества мышления и бытия, основные положения его объективно-идеалистической системы и ее составные части – логику, философию природы и философию духа.

Основной тезис гегелевской философии заключается в следующем: объективное абсолютное мышление составляет первооснову и движущую силу всего сущего. Это объективное мышление Гегель называл абсолютной идеей. В процессе саморазвития абсолютная идея порождает всю действительность. Происходит это в результате отчуждения идеи от самой себя и перехода в другую форму бытия. Таким образом, реальная действительность понимается Гегелем как инобытие абсолютной идеи.

Диалектика Гегеля имеет идеалистический характер: развитие характерно только для логических категорий, а весь мир не развивается во времени, а только лишь разворачивается в пространстве.

Высшим природным воплощением абсолютной идеи является человек, имеющий сознание, разум. С помощью субъективного духа, человеческого разума идея познаёт саму себя и своё инобытие. Учение Гегеля о субъективном духе – это учение о человеке и закономерностях его общественной жизни. Философия духа включает учение о трёх формах духа:

1) субъективный дух – индивидуальное человеческое сознание;

2) объективный дух – учение об обществе, включающее в себя право, нравственность и государство;

3) абсолютный дух, который включает в себя искусство, религию и философию.

Дух постоянно развивается от низшего к высшему, достигая своей вершины в философском учении самого Гегеля. В этом утверждении проявляется одно из противоречий между системой и методом Гегеля, суть которого студентам необходимо обнаружить.

В философии истории Гегель занят отысканием разумного начала в общественном развитии. Рекомендуется изложить представления Гегеля о социальном процессе, «хитрости» исторического разума, движущей силе истории, ее цели, критерии общественного прогресса.

Историю общества Гегель периодизирует по степени осуществленности свободы в обществе. Он выделяет несколько эпох в развитии общества: восточную, в котором свободен лишь один деспот, греческую и римскую, в которых свободны лишь немногие, и германскую, в которой немецкий народ и прусское государство достигли полной свободы. Германское же общество Гегель считал вершиной всемирной истории, в чем также обнаруживается противоречие между методом и системой гегелевской философии.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1.Каковы основные идеи философии эпохи Просвещения?

2. В чем проявляется наивность просветительской философии и в чем ее сильные стороны?

3. В чем проявляется разница социальных воззрений Вольтера и Руссо?

4. Что Кант понимал под категорическим императивом?

5.Что лежит в основе истории, по Гегелю?

6.Что Гегель понимал под «хитростью» исторического разума?

Тема 7. Философские аспекты труда, частной собственности и отчуждения, идея коммунизма в философии К.Маркса

Методические рекомендации к изучению темы

Важную роль в философии Маркса играет понятие отчуждения. *Отчуждение* – это такая ситуация в обществе, когда силы, созданные человеком

и контролируемые им, пока они создаются, отделяются от него и начинают над ним господствовать, определяя его деятельность.

Маркс проанализировал феномен экономического отчуждения при капитализме и выделил четыре его формы:

1) отчуждение предметов и результатов труда от производителя. Рабочий использует природные материалы, которые ему не принадлежат, и получает продукты, которые также ему не принадлежат. В конечном счете, предметы и результаты труда подчиняют себе рабочего, он от них полностью зависит.

2) отчуждение трудовой деятельности от человека. При капитализме труд принудителен, человек не может выбирать, работать ему или не работать. Человек в процессе труда выступает винтиком, а сам труд является лишь средством для удовлетворения других потребностей. Свободным человек является вне труда, в сфере потребления: в еде, питье, отчасти бытовых условиях.

3) отчуждение человека от его родовой сущности. Сущностная черта человека – преобразование внешнего мира по меркам человека, трудовая деятельность. Но здесь человек как раз и не свободен, труд ему представляется внешней и нежеланной силой, а в сфере потребления человек обладает некоторой долей свободы, имея возможность выбирать между товарами.

4) отчуждение между людьми, которые в обществе выступают как конкуренты.

Феномен отчуждения человека Маркс связывал с наличием в обществе частной собственности. Борьба с отчуждением предполагает ликвидацию частной собственности.

Безусловным вкладом Маркса является материалистическая концепция практики. Человек не просто часть природы, он преобразует, изменяет ее, и в процессе преобразования природы изменяет себя и отношения в обществе.

Практика – чувственно-предметная деятельность человека по преобразованию природы в своих целях (производство вещей), в процессе которой люди изменяют и самих себя.

Студентам следует раскрыть основные черты практики – ее предметность, общественный и целеполагающий характер, историчность и первичность по отношению ко всем другим видам деятельности, в том числе культуре, искусству, религии и т.д.

Проблема человека рассматривается Марксом и в связи с явлением отчуждения. Жизнь людей в условиях экономического отчуждения искажает сущность людей, делает их «частичными» людьми. Человек, производящий подневольно, ради куска хлеба, из-за страха быть наказанным, производит «не по-человечески». Ликвидация отчуждения позволит человеку производить «как человеку», соответственно его родовой сущности. Тогда труд превратится в средство, развивающее человека, в процесс, в котором человек сможет реализовать свои лучшие личностные качества. В обществе будущего труд как свободная самореализация перевернет и человека, и его отношения к природе и другим людям. Сформируется «универсальный человек». Он

преобразует природу не из выгоды и пользы, а по законам красоты, в соответствии с законами самой природы. Изменится и сущность самого человека: он будет стремиться не только к удовлетворению своих материальных потребностей, а, прежде всего, к развитию своих многочисленных духовных потенций.

Универсально развитый человек, живущий в гармонии с природой и другими людьми,– идеал. Выявившиеся пороки капиталистического общества и справедливая критика капитализма Марксом подтолкнули его к мысли о необходимости изменения существующего строя и создании нового коммунистического общества.

Тема 7.1 Проблема культурного самоопределения России в отечественной философии. Философия русского зарубежья

Методические рекомендации к изучению темы

При рассмотрении данной темы важно подчеркнуть, что русская философия в основном разрабатывала вопросы о сущности человека, его религиозно-нравственной жизни, о смысле и целях истории, субъекте исторического действия. Идеи русских философов имели нравственно-практическую или социально-политическую направленность. В историю культуры отечественные философы внесли целый ряд оригинальных идей, порожденных особенностями нашей истории и национального характера.

Проблема культурного самоопределения России наметилась в XVIII в. в противоборстве двух тенденций: одна акцентировала внимание на самобытности русской мысли и жизни, другая пыталась вписать Россию в процесс развития европейской культуры.

Следует заметить, что родоначальником западничества был П.Я. Чаадаев. Студентам следует охарактеризовать взгляды Чаадаева на российскую историю. Русская культура трактовалась им как находящаяся в отрыве от цивилизованной Европы. Безрадостное существование своей страны Чаадаев объясняет ее печальным прошлым: сначала она находилась в состоянии дикого варварства, потом грубого невежества, затем – чужеземного владычества. Освободившись от татарского ига, русские попали в новое рабство – крепостничество. Немаловажно отметить, что негативная оценка П.Я. Чаадаевым русской истории обусловлена его категорическим неприятием самодержавия и крепостной зависимости.

Необходимо подчеркнуть, что взгляды П.Я. Чаадаева послужили катализатором для дискуссии в русской обществоведческой мысли между славянофилами и западниками о путях дальнейшего развития России.

К.Д. Кавелин, Т.Н. Грановский, А.И. Герцен, Н. Огарев, В.П. Боткин, Б.Н. Чичерин, отчасти В. Белинский, Н.Г. Чернышевский, Н.А. Добролюбов были сторонниками западноевропейского пути развития. Социальный идеал России они видели в приближении к европейской цивилизации, для чего не-

обходимо отменить крепостное право, способствовать учреждению демократических принципов управления, свободы слова, печати, развивать образование и науку.

Славянофилы (А.С. Хомяков, И.В. Киреевский, К.С. Аксаков, Ю.Ф Самарин и др.) настаивали на самобытности исторического развития России.

Студентам необходимо проанализировать аргументы славянофилов, приводимые ими в обоснование своей позиции.

В частности, особую роль в формировании «русского духа», «исконно русских начал» А.С. Хомяков отводил православию. Славянофилы своеобразно оценивали самодержавие, которое считали лучшей формой правления для России. Важным фактором, определяющим особый путь России, славянофилы считали также общинное устройство, «мир». Идеализируя сельскую общину и представляя ее деятельность как гармоничное сочетание личных и общественных интересов, славянофилы предлагали сделать общинный принцип всеобъемлющим. По мере его распространения, по их мнению, в российском обществе будет укрепляться дух соборности, а ведущим началом социальных отношений станет «самоотречение каждого в пользу всех».

Крупнейшей фигурой в философии «русского зарубежья» был Н. А. Бердяев. Духовная эволюция Бердяева шла от марксизма – к неокантианству, затем к персонализму и завершилась религиозным экзистенциализмом.

Отношения Духа и Природы рассматривались Бердяевым дуалистически: они представляют собой два обособленных мира, имеющих единый источник, коим является абсолютная свобода, отождествлённая с хаосом. Свобода может быть рациональной (следование моральным законам) и иррациональной. Иррациональная свобода выступает носителем потенциального зла. Зло объективно, «мир во зле лежит». Истинная свобода, которая ведёт человека к добру, понимается мыслителем как любовь к Богу.

Центральной проблемой в философии Бердяева была проблема человека, которую он ставил как проблему свободы человеческого творчества. Мир – это результат свободного сотворчества Бога и человека.

Согласно взглядам Бердяева, общественная жизнь по существу своему духовна, а не материальна. Духовность выступает в форме соборности, в которой реализуется богоустановленный миропорядок. Этот порядок устроен на началах иерархии и послушания. Именно с иерархизмом Бердяев связывал формирование личности.

Следует заметить, что идеи Бердяева оказали огромное влияние не только на русскую философию, но и на развитие мировой философской мысли.

Изучая вопрос о философии русского зарубежья, следует рассмотреть взгляды русских философов – Н. Бердяева, С. Булгакова, С. Франка, Н. Лосского, Л. Карсавина, П. Сорокина, после социалистической революции 1917 года высланных за пределы России. Немаловажно заметить, что в эмиграции русские философы сформировали новые, теперь уже западные течения. Корни таких философских направлений, как феноменология, экзистенциализм, персонализм, выросли из творчества русских мыслителей.

Основная литература: [1, 2, 3, 4, 5, 6, 8, 9]

Вопросы для самопроверки

1. Что такое отчуждение, по Марксу?
2. Чем марксистская трактовка практики отличалась от предшествующих подходов в ее понимании?
3. Назовите основные черты практики ?
4. В чем заключается особенность русской философии?
5. Какие особенности русской истории выделяли славянофилы, обосновывая ее своеобразие?
6. Актуален ли спор между западниками и славянофилами в наши дни?

Тема 8. Современная западная философия. Экзистенциализм

Методические рекомендации к изучению темы

Рассматривая данную тему, необходимо отметить, что для современной западной философии характерно огромное количество школ, направлений, концепций, в которых был поставлен ряд новых вопросов, не исследовавшихся в предшествующий период. К ним относятся, например, такие проблемы как существование человека в мире, анализ языка, смысла терминов, история науки и специфика развития научного знания, исторический процесс, проблема понимания и др.

На рубеже XIX и XX вв. совершился переход от традиционной классической философии к неклассической. Критическому пересмотру подвергаются принципы классической философии. Последняя верила в могущество разума, считая его единственным эффективным средством познания и преобразования действительности. Знание предполагалось только как ясное, доказательное, логически стройное, соответствующее внешнему миру. В XX в. философы обратили внимание на то, что в состав духа входят нерациональные моменты – инстинкты, интуиция, эмоциональные и волевые акты, которые также могут служить средством постижения мира. Возникают иррационалистические направления в философии: фрейдизм, интуитивизм, герменевтика, философия жизни.

Современных философов классические системы не удовлетворяют ввиду потери в них конкретного человека. В человеке рассматривалась прежде всего его сущность, а многообразные субъективные проявления оставались вне анализа. Неклассическая современная философия в качестве основы берет жизнь в ее многообразных проявлениях (философия жизни), существование отдельного человека (экзистенциализм). Философия сущностей заменяется на философию существования.

Экзистенциализм, или философия существования, – одно из крупнейших направлений иррациональной философии XX в., отразившее духовную ситуацию кризисного общества.

Студентам следует обратить внимание на специфику формирования экзистенциализма, которое шло волнами: накануне первой мировой войны в России в творчестве Н. Бердяева и Л. Шестова, после первой мировой войны в Германии в работах М. Хадеггера, К. Ясперса, М. Бубера и в период второй мировой войны во Франции в произведениях Ж-П. Сартра, А. Камю, Ф. Кафки, С. де Бовуар, Г. Марселя и др. Различают религиозный и атеистический экзистенциализм, разницу между которыми студентам следует выяснить.

Предмет философии, с точки зрения экзистенциализма, бытие человеческой личности, способ существования человека в мире. У человека существование предшествует сущности: человек творит сам себя, он – самореализующийся проект, и процесс обретения собственной сущности, «делания себя» длится всю жизнь. Смысл человеческого существования – в реализации себя как свободного человека.

Внешний мир в экзистенциализме рассматривается как безусловно враждебный и чуждый человеку, он жесток, бессмыслен, наполнен страданиями. Жизнь состоит в «трудности бытия», ее единственное достояние – обязанности. Выявить сущность человека в таких условиях трудно. Но такое возможно в так называемой «пограничной ситуации», которая понимается как существование на грани жизни и смерти. В своем реальном существовании, будучи «заброшен» в этот мир, человек постоянно испытывает страх, тревогу, ожидание. В то же время он должен самоопределиться в отношении себя, своей судьбы. Философия, по мнению экзистенциалистов, обязана помочь отчаявшемуся человеку преодолеть страх и искать свое подлинное «Я» в самых абсурдных ситуациях.

Строя свои отношения с миром, человек осуществляет выбор, и этот выбор свободен. Студентам необходимо уяснить специфику понимания свободы в экзистенциализме: свобода – это тяжелое бремя, огромная ответственность за последствия своего выбора. Человек «обречен» быть свободным, извиняющих обстоятельств за ошибку у него нет.

Важное место в экзистенциализме занимают проблемы отчуждения, ответственности, смысла жизни, уникальности человеческого существования, целостности личности, которые студентам следует рассмотреть. Необходимо остановиться на причинах пессимистических мотивов этого философского направления, показать их связь с духовной ситуацией кризисных обществ.

Тема 8.1 Философия позитивизма и его исторические формы

Методические рекомендации к изучению темы

Неопозитивизм – одно из основных направлений мировой философии, по-новому осмысливающее ее взаимоотношения с наукой. В связи с тем, что

неопозитивизм представляет собой современную, третью форму позитивизма целесообразно рассмотреть историю его становления.

Важно отметить социально-исторические условия возникновения позитивизма, связанные с индустриализацией Европы в 40-х гг. XIX в., с развитием науки, расширяющимся применением ее достижений, преобразовавшим весь образ жизни людей.

Первый позитивизм, представителями которого были О. Конт, Д.С. Милль и Г. Спенсер, проникнут верой в науку как единственное средство решения всех проблем и настроен на борьбу с метафизикой. Студентам следует выделить два различных понимания термина «метафизика»: 1) как противоположный диалектике метод и 2) как учение об умозрительно постигаемых началах бытия.

По мнению О. Конта, философия должна отказаться от попыток объяснить окружающий мир, т.е. от метафизической проблематики, и обратиться к исследованию положительного знания. В позитивистском понимании философия должна стать методологией науки, служить обобщающей сводкой результатов эмпирических наук, изучать их взаимосвязи.

Вторая форма позитивизма – эмпириокритицизм – сформировался в конце XIX – начале XX вв. Его основателями были Р. Авенариус и Э. Мах. Наследуя антиметафизическую установку «первого позитивизма», они пытались обнаружить источники метафизических заблуждений в механизмах реального познавательного процесса. Убежденность в том, что «не тела вызывают ощущения, а комплексы элементов образуют тела» сближала эмпириокритиков с субъективным идеализмом Д. Беркли.

Неопозитивизм, возникший в 20-е гг. XX в., стремился поставить философию на службу науке. Большую роль в его формировании сыграл «Венский кружок», в который входили во главе со М. Шликом Р. Карнап, К. Гёдель, О. Нейрат, Ф. Вайсман и др. Представители «Венского кружка» предложили программу обновления научного и философского знания, заключавшуюся в отделении науки от метафизики. Инструментом демаркации они считали процедуру верификации, суть которой – в соотнесении языковых высказываний с эмпирическими фактами.

Студентам следует рассмотреть те трудности, с которыми столкнулся неопозитивизм в попытке реализации своей программы, а также подчеркнуть неправомерность сведения философии только к логическому анализу языка, зачеркивая ее право на анализ коренных мировоззренческих проблем.

Завершая изучение данного вопроса , целесообразно остановиться на постпозитивистском понимании развития науки и рассмотреть основные идеи К. Поппера: его представления о развитии научного знания, принцип фальсификации, теорию рациональности.

Тема 8.2 Психоанализ и его эволюция

Методические рекомендации к изучению темы

При рассмотрении данной темы важно подчеркнуть, что фрейдизм и психоанализ относятся к иррациональным направлениям современной западной философии. Под психоанализом понимают такое направление в мысли, которое идеи Фрейда распространило на исследование проблем истории, культурологии, философии, эстетики, религии.

Учение З. Фрейда основано на представлении о доминирующей роли бессознательного в человеческой жизни. Фрейд разрывает с декартовской традицией, которая отождествляла психику и сознание. Фрейд усложнил структуру сознания, выделив в ней три уровня – бессознательное, предсознание и сознание, которые студентам следует охарактеризовать.

Важно отметить, что в этой модели психики Фрейд обозначил очень важную проблему неосознаваемых побуждений, которые реально движут людьми, однако в сознании не отражаются. С многочисленными проявлениями бессознательного человек сталкивается постоянно: оговорки, описки, неожиданные воспоминания давно забытого, сновидения, фантазии и т.д.

Далее следует проанализировать новую модель психики, предложенную Фрейдом в 20-е гг. XX в., в которой выделил три слоя: Оно (Ид), Я (Эго) и Сверх-Я (Супер-Эго).

«Оно» – это область влечений двух основных типов: 1) врожденных и наследуемых (инстинкты и рефлексы) и 2) вытесненных и приобретенных (запретные желания и «автоматизмы»). В Оно взаимодействуют и противоборствуют два механизма: Эрос и Танатос.

«Я» – система, в которой собирается информация о внешнем и внутреннем мире человека. Я непосредственно контактирует с внешним миром, приспосабливая человека к нему, это сфера принятия решений. Я ориентировано на принцип реальности, который подчиняет себе принцип удовольствия.

«Сверх-Я» – система, отвечающая за нравственное сознание личности, за совесть. Сверх-Я по отношению к Я выполняет роль цензора, судьи и даже палача. Формируется эта структура в процессе социализации, в общении с другими людьми под действием различных социальных институтов.

В целом, по Фрейду, духовная жизнь человека конфликтна: энергия Оно стремится к удовольствию, но испытывает давление со стороны Сверх-Я в виде запретов общества. Тогда она ищет обходные пути для своей актуализации, и в Я возникает напряженность. В свою очередь, Я спасается с помощью «защитных механизмов» – вытеснения, рационализации, сублимации. Задача психоанализа и есть освободить Я от этого дискомфорта.

Позиция Фрейда является *иррациональной,* т.к. бессознательное рассматривается существенной характеристикой человеческой природы. Но иррационализм Фрейда дополняется его верой в разум, он считает, что Я может поставить под свой контроль Оно. Кредо мыслителя – рациональный анализ иррациональных побуждений.

Идеи Фрейда развили его ученики А. Адлер и К. Юнг. Адлер, в отличие от Фрейда, считал, что человек – это цельный самосогласующийся организм. Целостность человеческой личности формируется в обществе. Первоначально у ребенка формируется чувство неполноценности, беспомощности, кото-

рую он пытается компенсировать стремлением к превосходству, совершенствованием себя. Культурные факторы как раз и предоставляют ему такую возможность. Таким образом, многое в судьбе человека зависит от него самого.

Юнг расширил понятие бессознательного Фрейда. Во-первых, оно больше, чем сексуальные влечения, как у Фрейда. Либидо Юнга – это жизненная сила с большим творчески потенциалом. Во-вторых, в бессознательном он выделил уровень коллективного бессознательного, архетипов. Этот слой хранит следы коллективной памяти человечества, предрасполагая людей воспринимать и реагировать на события определенным образом. Архетипы – мысли, чувства, образы, выработанные в опыте всего человечества, на которых строят свое поведение отдельные индивиды. К таким архетипам Юнг отнес образы матери, ребенка, героя, мудреца, Бога, смерти, обманщика и т.д. Эти образы зашифрованы в религиозных традициях и мифах, в эзотерической символике. Люди тянутся к ним как к образцам, но и боятся их, поэтому архетипы выражаются часто в символической форме.

Немаловажно подчеркнуть, что явление бессознательного, исследованное Фрейдом и его последователями, находит огромное применение в деятельности людей.

Основная литература: [1, 2, 3, 4, 5, 6, 8]

Вопросы для самопроверки

1. Что характерно для иррациональных направлений современной западной философии и в чем заключается их отличие от классической философии?

2.В чем выдели свою главную задачу неопозитивисты?

3. В чем заключалось новое понимание человека во фрейдизме?

4. Какие инстинкты Фрейд выделял в качестве основных у человека?

5. Можно ли рассматривать фрейдистскую модель психики человека как универсальную?

Тема 9. Глобальные научные революции в истории естествознания. Понятия «картина мира» и «парадигма»

Методические рекомендации к изучению темы

Изучение первого вопроса необходимо начать с определения понятий «картина мира» и «парадигма». Одним из первых термин «картина мира» использовал знаменитый физик Генрих Герц.

Студентам следует обратить внимание на различие понятий «картина мира», «естественнонаучная картина мира», «философская картина мира». Следует отметить, что отдельные естественные науки могут создавать собст-

венные картины исследуемой ими реальности. В таком случае говорят о частнонаучных (или локальных) картинах мира.

Философская картина мира представляет собой систему наиболее общих философских понятий (категорий), принципов, дающую на определенном историческом этапе представление о мире в целом. В зависимости от тех или иных решений определенных философских проблем могут строиться, например, материалистическая или идеалистическая картины мира, метафизические или диалектические представления о мире.

Следует отметить, что история научного познания сопровождалась периодической сменой картин мира, а, следовательно, и так называемых парадигм. Под парадигмой понимают определенную совокупность общепринятых в научном сообществе на данном историческом этапе идей, понятий, теорий, а так же методов научного исследования, которые дают модель постановки проблем и их решений научному сообществу.

Развитие естествознания характеризуется чередованием относительно спокойных периодов и периодов «крутой ломки» фундаментальных законов и принципов. Процесс, имеющий своим результатом смену научных теорий, преобразование стиля научного мышления, обозначается понятием «научная революция». Научная революция, которая приводит к формированию совершенно нового видения мира, вызывает появление принципиально новых представлений о его структуре и функционировании, а так же влечет за собой новые способы, методы его познания, называется *глобальной* научной революцией.

В результате *первой глобальной научной революции*, происходившей в XVI – XVII вв., радикально изменилось миропонимание. Это явилось следствием появления гелиоцентрического учения в космологии и последующего создания классической механики, ставшей основой своеобразного механистической картины мира.

Студентам следует обратить внимание, что первая научная революция считается началом формирования современного естествознания, базирующегося на экспериментальной методологии. Возникает так называемая классическая наука Нового времени. Ее становление связано с гелиоцентрическим учением Н. Коперника, открытиями Галилея, Кеплера и Ньютона. Особо следует отметить И.Ньютона, разработавшего классическую механику как целостную систему знаний о механическом движении тел. Ньютон сформулировал основные идеи, понятия, принципы, составившие механистическую картину мира.

Механистическая картина мира строилась на принципе жесткого детерминизма и образе мироздания как часового механизма. В целом, она сыграла положительную роль в развитии науки, так как давала естественнонаучное понимание многих явлений природы, освободив их от религиозных толкований. Механистическая картина мира оказала мощное влияние на развитее всех других наук на долгое время.

С середины XVIII в. начался «подрыв» механистической картины мира по двум направлениям: во-первых, со стороны самой физики и, во-вторых, со

стороны геологии и биологии, что послужило началом *второй глобальной научной революции*, завершившейся в конце XIX в. Указанный революционный период в развитии естествознания характеризовался переходом от метафизического миропонимания к диалектическому.

Следует проанализировать эволюционные идеи Ж. Б. Ламарка, Ж. Кювье и Ч. Лайеля, проникающие в биологию и геологию.

Диалектизации естествознания и «свержению» метафизико-механистического способа мышления в первые десятилетия XIX в. способствовали три великих открытия: создание клеточной теории, открытие закона сохранения и превращения энергии, и разработка Дарвином эволюционной теории. В результате этих и многих других открытий основополагающие принципы диалектики – принцип развития и принцип всеобщей взаимосвязи вошли в естествознание и стали основой формирования новой картины мира.

Следует отметить М.Фарадея и Д. Максвелла, чьи работы в области электромагнетизма положили начало крушению механистической картины мира.

Третья глобальная научная революция начинается с конца XIX в. и продолжается до середины XX столетия. В этот период были созданы принципиально новые, квантово-релятивистские представления о физической реальности. Произошли революционные перемены в различных областях знания: в физике (открытие сложного строения атома, становление релятивистской и квантовой теории), в космологии (концепция нестационарной Вселенной), в биологии (возникновение молекулярной биологии, становление генетики). В конце третьей глобальной научной революции возникает кибернетика.

Студентам следует обратить внимание на то, что третья глобальная научная революция вызвала появление неклассической науки и преобразование прежнего стиля научного мышления.

Особое значение имела теория относительности А.Эйнштейна, которая показала неразрывную связь между пространством, временем и материей.

Студентам рекомендуется обратить внимание на философско-методологические выводы из достижений неклассической науки, в частности, на возрастание роли философии в развитии естествознания; сближение объекта и субъекта познания; зависимость знания от применяемых субъектом методов и средств его получения. Идеалы и нормы неклассической науки характеризуются пониманием относительной истинности теорий, допускается истинность нескольких отличающихся друг от друга конкретных теоретических описаний одной и той же реальности, поскольку в каждом из них может содержаться момент объективно-истинного знания.

В 70-е гг. XX в. были совершены новые радикальные научные открытия. Эти достижения можно охарактеризовать как *четвертую глобальную научную революцию*, в ходе которой сформировалась постнеклассическая наука. Она характеризуется ориентацией на исследование сложных, исторически развивающихся систем, и новым уровнем интеграции научных исследований, требующих участия специалистов различных областей знания.

На современном постнеклассическом этапе познания материального мира чрезвычайно важную роль играет парадигма *самоорганизации.* Длительное время в науке доминировало представление об отсутствии явления самоорганизации в неживой природе. Считалось, что самоорганизующиеся процессы присущи только живым системам, а объекты неорганического мира способны изменяться только в направлении хаоса.

Постепенно в науке накапливалось все большее число фактов, свидетельствовавших о феномене самоорганизации в неживой природе. Указанные наблюдения и обобщения привели к возникновению *синергетики* – междисциплинарного научного направления, изучающего универсальные механизмы самоорганизации, т.е. механизмы самопроизвольного возникновения и относительно устойчивого существования макроскопических упорядоченных структур самой различной природы.

Важно отметить, что зарождение синергетики произошло в нашей стране благодаря работам отечественных ученых Б.Н. Белоусова и А.М. Жаботинского и активно развивается школами Г. Хакена и И. Пригожина.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Какой смысл вкладывается в понятия «научная картина мира» и «парадигма»?
2. Что подразумевается под глобальной научной революцией, в чем ее отличие от научной революции?
3. По какому критерию выделяются периоды в развитии науки?
4. Назовите основные принципы классического естествознания?
5. Выделите характерные черты неклассического стиля мышления?
6. Что является результатом четвертой глобальной научной революции?

Тема 10. Наука как особый вид деятельности

Методические рекомендации к изучению темы

Рассматривая данную тему, следует отметить, что существуют десятки определений науки, среди которых можно выделить две группы: 1) определения науки как системы знаний и 2) определения ее как вида человеческой деятельности. Это и есть два важнейших аспекта науки, которые необходимо учесть в ее определении.

Наука – это деятельность по производству объективно-истинного знания и результат этой деятельности – систематизированное, достоверное, практически проверенное знание.

Не всякое знание является наукой. В отличие от опытно-практического знания, наука есть знание причин. Научное знание отражает устойчивые, повторяющиеся связи явлений действительности, выражаемые в законах.

Сущность научного знания заключается в достоверном обобщении фактов, в том, что за случайным оно находит необходимое, закономерное, за единичным – общее и на этой основе осуществляет предвидение различных явлений и событий. Весь прогресс научного знания связан с возрастанием силы научного предвидения, которое дает возможность контролировать процессы и управлять ими.

Студентам следует выделить особенности научного знания – объективность, системность, непротиворечивость и охарактеризовать их.

В настоящее время научная деятельность институциализирована, т.е. приобрела устойчивые социальные формы, организована.

Как вид деятельности наука характеризуется:

1) Определенной системой ценностей, которая определяет деятельность ученого. Это ценности истины, разума, нового знания, которое и является результатом деятельности ученого. В целом наука в качестве своей основы имеет особый тип мышления, для которого характерны рационализм, стремление к знанию, независимость суждений, готовность признать свои ошибки, честность, коммуникабельность, готовность к сотрудничеству, творческие способности, бескорыстность.

2) Определенным набором «инструментов» – технических устройств, аппаратуры и т.д. – используемых в научной деятельности. В настоящее время оснащенность научного труда во многом определяет его результативность.

3) Совокупностью методов, используемых для получения нового знания.

4) Способом организации научной деятельности. Наука сейчас это сложнейший социальный институт, включающий в себя три основных составляющих: исследования (производство нового знания); приложения (доведения новых знаний до их практического использования); подготовку научных кадров. Все эти составляющие науки организованы в виде соответствующих учреждений: университетов, научно-исследовательских институтов, академий, конструкторских бюро, лабораторий и т.д.

Тема 10.1 Закономерности развития науки

Методические рекомендации к изучению темы

Важнейшими закономерностями развития науки считаются следующие:

1) Обусловленность развития науки потребностями общественно-исторической практики. Это – главная движущая сила, источник развития науки.

2) Относительная самостоятельность развития науки. Какие бы конкретные задачи ни ставила практика перед наукой, решение этих задач может быть осуществлено лишь по достижении определенных ступеней развития процесса познания мира.

3) Преемственность в развитии идей и принципов, теорий и понятий, методов и приемов науки, неразрывность всего познавательного процесса. Каждая ступень в развитии науки возникает на основе предшествующей ступени, с удержанием всего ценного, что было накоплено раньше.

4) Постепенность развития науки при чередовании периодов относительно спокойного (эволюционного) развития и бурной (революционной) ломки теоретических основ науки, системы ее понятий и представлений (картины мира). Эволюционное развитие всей науки – это процесс постепенного накопления новых фактов в рамках существующей парадигмы. Революция в науке наступает, когда начинается коренная ломка ранее установившихся воззрений, пересмотр фундаментальных законов и принципов в результате открытия новых явлений, не укладывающихся в рамки прежних воззрений.

5) Взаимодействие и взаимосвязанность всех составных отраслей науки, в результате чего предмет одной науки может и должен исследоваться приемами и методами других наук.

6) Свобода критики, беспрепятственное обсуждение спорных или неясных вопросов науки, открытое и свободное столкновение различных мнений. В результате такой борьбы преодолевается первоначальная неизбежная односторонность различных взглядов на объект исследования и вырабатывается единое воззрение, более адекватное самой действительности.

7) Дифференциация и интеграция научного знания. Дифференциация научного знания проявляется в выделении отдельных разделов науки в относительно самостоятельные дисциплины, интеграция научного знания – формирование наук, которые изучают свойства и отношения, общие для большого числа разнокачественных объектов.

8) Математизация науки. Современная наука характеризуется проникновением математики в различные области знания. Ныне доступ не только к глубоким проблемам естествознания, но и к области социальных исследований требует тончайших математических методов.

Существенной особенностью современной науки является то, что она стала такой силой, которая предопределяет практику, выступает в качестве предпосылки новых технических достижений, создает условия для гигантского развития производительных сил общества.

Тема 10.2 Синергетика как новое мировидение XXI в. Мир как самоорганизующаяся система

Методические рекомендации к изучению темы

На современном постнеклассическом этапе познания материального мира чрезвычайно важную роль играет парадигма самоорганизации. Длительное время в науке доминировало представление об отсутствии явления самоорганизации в неживой природе. Считалось, что самоорганизующиеся процессы присущи только живым системам, а объекты неорганического мира способны изменяться только в направлении хаоса.

Постепенно в науке накапливалось все большее число фактов, свидетельствовавших о феномене самоорганизации в неживой природе. Указанные наблюдения и обобщения привели к возникновению *синергетики* – междисциплинарного научного направления, изучающего универсальные механизмы самоорганизации, т.е. механизмы самопроизвольного возникновения и относительно устойчивого существования макроскопических упорядоченных структур самой различной природы. Синергетика стирает грани между физическими и химическими процессами, с одной стороны, и биологическими и социальными процессами – с другой, ибо исследует общие механизмы самоорганизации и тех, и других.

Важно отметить, что зарождение синергетики произошло в нашей стране благодаря работам отечественных ученых Б.Н. Белоусова и А.М. Жаботинского и активно развивается школами Г. Хакена, И. Пригожина и С. Курдюмова. Эти исследователи обнаружили, что в открытых диссипативных системах не действуют линейные зависимости. Открытые системы эволюционируют не постепенно, а скачкообразно. На траектории их эволюции всегда есть бифуркационные точки, где происходит выбор одного из множества путей следующего этапа развития системы. В точках бифуркации выбор системой дальнейшей траектории движения определяется случайным образом и не связан линейной или причинной зависимостью с ее предшествующими состояниями. Современное естествознание меняет свой концептуальный облик, переходя при описании своих объектов с языка линейных уравнений и причинно-следственных зависимостей на язык нелинейности и резонансных связей между объектами. По сути это означает новую революцию в развитии современной науки. Новая парадигма современного естествознания – синергетика – является выражением нелинейного мышления в науке, основанного на признании творческой, конструктивной роли случая в мире и вторичности порядка по отношению к хаосу.

Студентам нужно усвоить значение новых понятий, используемых в постнеклассической науке, таких как открытая система, нелинейность, флуктуация, бифуркация, стохастичность, хаосомность, аттрактор, куматоид и др., а также основные идеи синергетики – концепцию нестабильного неравновесного мира, идею возникновения порядка из хаоса, представления о неопределенности и многоальтернативности развития.

Важно подчеркнуть значимость глобального всестороннего взгляда на мир; стремление построить общенаучную картину мира на основе принципов универсального эволюционизма, объединяющих в единое целое идеи системного и эволюционного подходов.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Каковы отличительные черты научного знания?
2. Существует ли единственный критерий, позволяющий отделить научное знание от ненаучного?
3. Каковы основные закономерности развития науки?
4. Что изучает синергетика?
5. Как трактуется детерминизм в синергетической парадигме?
6. Каково соотношение между диалектикой и синергетикой?

Тема 11. Идеальность сознания. Концепции идеального в отечественной философии

Методические рекомендации к изучению темы

В начале рассмотрения данной темы целесообразно отметить, что понятие «идеального» в философии используется для объяснения специфики сознания. Идеальных образов как таковых нет ни в природе, ни в человеке как биологическом существе, они являются внешними и по отношению к природе, и по отношению к телесной организации человека.

Обычно сознание определяют как внутренний мир человека, его представления, чувства, идеи, настроения. Содержанием сознания выступает информация об окружающем мире и самом человеке, знания, а также оперирование ими, их преобразование и творчество. Сознание в этой связи предстает как особая субъективная реальность, отличающаяся от реальности объективной, его можно рассматривать как субъективный образ объективного мира.

Важно понять, что образ вещи в сознании и реальная вещь – это не одно и то же. Образ вещи в сознании – это, по сути, информация, которая выступает как набор данных для подготовки будущего действия. Использование информации, или ее раскодирование, предполагает перевод ее в соответствующее действие. Раскодирование – это понимание, а понимание – это правильное действие с данным предметом, но не реальное, а потенциальное. Например, образ стола или автомобиля предполагает умение обращаться с ними соответствующим образом. Любой предмет является материальным объектом и одновременно «сгустком» информации по его использованию. Материальным он является «сам по себе», а информацией – по отношению к человеку и его деятельности.

Об идеальном можно сказать, что его нет в реальности, и вместе с тем, что оно есть: идеальное не существует как реальная вещь, но оно существует в реальной вещи как способность человека действовать с ней.

Идеальное – это схемы деятельности, смыслы-функции, «запечатленные», «опредмеченные» в продуктах человеческой деятельности. Другими словами, идеальное – это общественно определенная форма вещи, но нахо-

дящаяся вне вещи в виде схемы активной человеческой деятельности. Эти схемы возникают и закрепляются в деятельности человека, создающего предметы для других людей. Сталкиваясь с человеческими предметами, формирующееся сознание «снимает» эти смыслы-функции, схемы деятельности, перерабатывает их в субъективную реальность человеческого сознания. Сознание – это отражение идеального в предметах, поэтому сознание не формируется вне общества, т.к. для его формирования требуется включение человеческого индивида в смысловое поле человечества в целом. Это включение осуществляется через общение человека с человеческими предметами на первом этапе, а позднее, через усвоение любых форм человеческой активности.

Российский философ Э.В. Ильенков попытался объяснить феномен идеального вообще и сознания как формы идеального отражения действительности, механизм их возникновения и функционирования. Он подчеркивал, что сознание – это не индивидуально- психологический феномен, его надо понимать как общественно-историческое явление. Это значит, что сознание рождается не как результат пассивного созерцания отдельного человека, а как результат и форма активной преобразовательной деятельности многих поколений. Сознание выступает моментом функционирования социальной системы.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Что такое сознание, какими функциями оно обладает?
2. Что понимается под «идеальностью сознания»?
3. Возможно ли формирование сознания вне общества?
4. Что понимается под термином «субъективная реальность», чем она отличается от объективной реальности?
5. В чем отличие между образом вещи и самой вещью?
6. Какой смысл вкладывается в понятия «опредмечивание» и «распредмечивание»?

Тема 12. Проблема познаваемости мира в истории европейской философии

Методические рекомендации к изучению темы

Познание исследует раздел философии, именуемый гносеологией (от греч. gnosis – знание, logos – учение) или теорией познания. Основной проблемой теории познания является проблема истины: способен ли человек достичь истины, правильно отразить окружающий мир. В зависимости от решения этого вопроса в философии выделяются позиции гносеологического оптимизма, скептицизма и агностицизма.

Студентам рекомендуется обратить внимание на истоки агностицизма и скептицизма, а также на то, что агностицизм противоречит самой практике познания, тому, что, например, ученым удается построить теории, подтверждающиеся на опыте.

Вплоть до середины XIX в. в теории познания господствовал взгляд, согласно которому познание есть результат воздействия внешнего мира на органы чувств человека и последующей переработки этих воздействий в его сознании.

Впервые связь между общественно-преобразовательной деятельностью и познанием была проанализирована в марксизме. Маркс рассмотрел познание в общественно-историческом контексте. Познает мир не абстрактный индивид, наделенный разумом и чувствами, а человек, деятельностные и познавательные способности которого сформированы в конкретном обществе.

В общем виде знания – это составная часть нашей деятельности, то, что позволяет нам поставить цель, наметить способы ее достижения и достичь желаемого результата. Знание практично, оно рождается в практике и живет в ней.

Студентам рекомендуется проанализировать следующие выводы, к которым пришла философия, исследуя процесс познания: 1) знание не является результатом пассивного отражения воздействия объекта на органы чувств человека (субъекта познания); познание – результат активного практического преобразования объекта человеком.

Тема 12.1 Проблема истины. Характеристики истины. Многообразие концепций истины и ее критерии

Методические рекомендации к изучению темы

Рассмотрение данной темы необходимо начать с характеристики истины. Со времен Аристотеля существует определение истины как знания, соответствующего действительности. Проблема заключается в том, как установить соответствие наших знаний действительности.

Истина – это адекватное отражение действительности, воспроизведение ее такой, какова она на самом деле. Достижение истины – не одномоментный акт, она представляет собой процесс движения знаний ко все более полному и адекватному воспроизведению действительности.

Процессуальность истины выражается с помощью понятий относительной и абсолютной истины. *Относительная истина* характеризует неполноту, приблизительность наших знаний, их ограниченность на данном этапе развития познания. *Абсолютная истина*, напротив, концентрирует в себе то в наших знаниях, что является безусловным, не может быть опровергнуто в будущем, что составляет элементы незыблемого, непреходящего знания. Всякая истина абсолютна и относительна одновременно.

Под объективностью истины понимается такое содержание наших знаний, которое не зависит от человека и человечества. Однако выражение ис-

тины субъективно, зависит от субъекта познания. Поэтому истина субъективна по форме выражения, но объективна по своему содержанию.

Отделить истину от заблуждения, неточностей, промахов познания и лжи позволяет ориентация на *критерий истинности знания*. В истории философии предпринимались многочисленные попытки выявить такой критерий. В Новое время сложилось два направления в теории познания – эмпиризм (Бэкон, Гоббс) и рационализм (Декарт, Спиноза). С точки зрения эмпиризма, истина подтверждается опытом. С точки зрения рационализма, критерии истины являются логическими – выводимость следствия из аксиом.

Студентам следует выделить несколько концепций истины.

Классическая (корреспондентская или аристотелевская) концепция трактует истину как соответствие знания объекту.

Согласно *теории когеренции,* критерием истины является внутренняя непротиворечивость, самосогласованность знания. Истинным считается то знание, которое встраивается в систему уже выработанного и проверенного знания, согласуется с ним.

Представители *конвенционализма* (А. Пуанкаре, Р.Карнап), считали, что в основе научных теорий лежит соглашение (конвенция) между учеными, их выбор обусловлен соображениями удобства, простоты и т.п.

С точки зрения *прагматизма,* истина понимается как польза, следовательно, критерий отделения истинного знания от ложного определяется тем, полезно оно или нет. Один из основоположников прагматизма американский философ У. Джеймс показатель истинности знаний видел в их способности приводить к эффективному результату, успешному действию.

Важно подчеркнуть, что эти концепции носят взаимодополнительный характер, выражают различные аспекты истинного знания.

С точки зрения современной науки, наиболее достоверным критерием истины является практика, однако и этот критерий не является абсолютным, т.к. практическая проверка не всегда возможна. В логике, математике, гуманитарном знании применяются другие способы оценки знания. Все более важную роль в научном познании играет формально-логический критерий истины.

Студентам рекомендуется проанализировать роль практики в познании, отметить, что практика – основа познания, цель познания и критерий истинности полученного знания. Проблемы, возникающие в практической деятельности, были и остаются важнейшим фактором, определяющим направление научного поиска

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Чем скептическая установка в познании отличается от агностицизма?
2. Каковы основания и аргументы скептицизма и агностицизма?

3. В чем заключается самое веское доказательство познаваемости мира?
4. Достижима ли абсолютная истина как полное и исчерпывающее знание о мире?
5. Можно ли согласиться с тем, что всякое знание относительно?
6. Существует ли единственный и неопровержимый критерий истины?

Тема 13. Основные категории морали, их ценностная характеристика. Общечеловеческие ценности

Методические рекомендации к изучению темы

Рассмотрение данной темы необходимо начать с выявления отличий морали от других форм регуляции поведения людей. Под моралью понимают реально существующие в обществе нравственные отношения, нормы поведения. Моральные нормы возникают в результате духовно-практического освоения действительности. Они отличаются от других средств социальной регуляции – права, традиции, обычая тем, что предполагают свободу выбора и основаны преимущественно на таких чувствах как стыд, угрызения совести, сознание собственного долга. В отличие от традиций, нормы морали содержат призыв к самосовершенствованию, преобразованию реальности, т. е. в них выражается противопоставление должного и сущего. В то время как нормы права в основном призваны регулировать уже развитые интересы людей, мораль также активно влияет на природу самих интересов, т.е. зовет человека к переделке собственной природы.

Студентам необходимо рассмотреть проблему возникновения морали, а также проанализировать процесс изменения нравственного самосознания личности от античной до современной эпохи.

В рамках данного вопроса, необходимо рассмотреть этику ненасилия, которая возникла еще в XIX в. Однако принято считать, что сам идеал ненасилия сформулирован в Нагорной проповеди (Новый завет). Заповеди непротивления злу насилием с большим трудом входили в сознание человека и поначалу казались просто невыполнимыми: они противоречили общепринятым нормам морали, принципам, природным инстинктам, традициям. В Нагорной проповеди непротивление злу рассматривалось как проявление морального совершенства, индивидуального нравственного превосходства над чужим грехом. Неумножение зла расценивалось как проявление добра.

Значительная роль в разработке принципов этики ненасилия принадлежит Л. Н. Толстому, который подводит нас к выводу: насилие бессильно, разрушительно, антигуманно.

Необходимо также отметить тот вклад, который внесли в развитие принципа этики ненасилия Мартин Лютер Кинг и лидер индийского освободительного движения М. Ганди.

Основная литература: [1, 2, 3, 4, 5, 8, 9]

Вопросы для самопроверки

1. Что такое мораль?
2. Что является источником моральных норм?
3. Чем моральные нормы отличаются от правовых норм?
4. Могут ли моральные нормы противоречить традициям?
5. Изменяются ли нормы морали?

Тема 14. Глобализация как ведущая тенденция развития современного мира: основные направления и проблемы

Методические рекомендации к изучению темы

Приступая к рассмотрению данной темы, следует отметить, что происходящие в настоящее время социальные изменения, охватывают не столько отдельные нации и народы, сколько цивилизацию в целом. Для их обозначения стали употреблять понятие глобализации.

Глобализация – формирование единого экономического и информационного пространства. Фундаментальной ее основой следует считать революцию в средствах коммуникации, связи и информатики, радикально изменившую характер интеллектуального, культурного и технологического взаимодействия между отдельными составными элементами всемирной цивилизации. В результате оказались фактически преодолены все национальные барьеры на пути распространения не только информации, но, что гораздо более существенно, инвестиций и технологических нововведений.

Глобализация предполагает резкое расширение масштабов мирохозяйственных и торговых отношений, рост научно-технических связей, новый качественный скачок в развитии мирового информационного сообщества в формировании глобальной экономики. Потенциально все это ведет к улучшению количественных и качественных показателей мировой экономики, к повышению благосостояния всего мирового сообщества. На деле же ситуация, складывающаяся в мировой экономике, далеко не так однозначна. Развивающийся ныне процесс глобализации по западному сценарию не только не способствует преодолению сложившихся в мире дисбалансов в экономическом развитии и условиях жизни людей, но и резко их усугубляет. Усиливается дифференциация различных стран, увеличивается пропасть между богатыми и бедными странами и регионами. Это закономерный результат глобального распространения правил игры, в которой выигрывают «сильные» – ведущие транснациональные корпорации и державы «золотого миллиарда», причем победители, разумеется, благоденствуют за счет «слабых», представляющих остальной мир. Такие «побочные» результаты глобализации вызывают все большее беспокойство в международных общественных кругах.

Студентам рекомендуется уделить особое внимание анализу социальных изменений, происходящих в России в результате модернизации и глоба-

лизации. В связи с этим, следует подчеркнуть, что наиболее часто типологическая принадлежность происходящих в России изменений определяется как «догоняющая модернизация».

С усилением процессов глобализации и реализацией политики однополярного мира возникли новые планетарные проблемы, и человечеству необходимо найти пути и средства их решения.

Основная литература: [1, 2, 3, 4, 5, 8,9]

Вопросы для самопроверки:

1. Что такое глобализация?
2. Какие основные направления глобализации можно выделить?
3. Каково соотношение понятий «глобализация», «вестернизация» и «модернизация»?
4. В чем проявляются негативные эффекты глобализации?
5. Возможно ли преодоление негативных тенденций глобализации?

Контрольные вопросы к зачёту

1. Мировоззрение, его исторические типы. Философия и наука. Философия и религия.
2. Предмет и специфика философии. Структура философского знания.
3. Особенности мифологического сознания.
4. Возникновение философии.
5. Основной вопрос философии. Разновидности идеализма. Формы материализма.
6. Натурфилософия античности. Учение о первоначалах мира. Первая историческая форма диалектики. Античная атомистика.
7. Классическая древнегреческая философия. Сократ. Платон. Аристотель.
8. Философия эпохи эллинизма. Скептицизм. Эпикуреизм. Стоицизм.
9. Средневековая философия, ее характерные черты. Патристика.
10. Основные проблемы средневековой схоластики.
11. Философия и наука эпохи Возрождения.
12. Философия Нового времени. Онтология и гносеология модерна. Обоснование либерализма и индивидуализма в воззрениях Т.Гоббса и Дж. Локка.
13. Немецкая классическая философия. И.Кант, его вклад в естествознание, агностицизм и этическое учение.
14. Объективный идеализм и диалектика Гегеля.
15. Русская философия XIX в. Славянофильство и западничество.
16. Западная философия XX в. Экзистенциализм.
17. Научные революции и смена типов миропонимания. Первая глобальная научная революция и создание механистической картины мира.
18. Вторая глобальная научная революция. Диалектизация естествознания. Изменение представлений о материи и движении.
19. Третья глобальная научная революция. Субстанциальная и реляционная концепции пространства и времени.
20. Четвертая глобальная научная революция. Синергетика как новое мировидение XXI в.
21. Проблема сознания в философии. Отражение и сознание. Формирование сознания, его социальная сущность.
22. Проблема познаваемости мира в истории европейской философии. Гносеологический оптимизм и пессимизм. Античный скептицизм. Агностицизм.
23. Чувственное и рациональное в познания.
24. Проблема истины. Характеристики истины. Многообразие концепций истины и ее критерии.
25. Природа как основа жизни и развития общества. Качественная специфика объектов живой природы. Понятие биосферы.
26. Социальная структура общества. Классы. Социальная стратификация.
27. Типы общества. Классификация обществ по их экономической основе. Общества «восточного» и «западного» типов.
28. Содержание, задачи, основные направления философии истории.
29. Формационная концепция общественного развития в философии истории марксизма.

30. Цивилизационный подход к истории человечества, его особенности.
31. Концепции «индустриализма» и «постиндустриализма» в западной философии истории второй половины XX в.
32. Проблема человека в философии. Образ человека в современной культуре.
33. Биологическое и социальное в человеке. Биологизаторские и социологизаторские направления в понимании человека. Человек как индивид и личность.
34. Глобальные проблемы современного общества. Их многообразие и общие черты. Пути преодоления глобальных кризисных ситуаций, стратегия дальнейшего развития человечества.

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная учебная литература

1. Голубинцев В.О. Философия : учебное пособие / Любченко В.С. - Новочеркасск : ЮРГПУ (НПИ), 2013. - 252 с. -Режим доступа: http://lib.npi-tu.ru/
2. Философия [PDF]: учебник/под В.П. Кохановского-23-е изд., стереотип. : учебник - М.: КНОРУС, 2014. - 368 с.- Режим доступа: http://lib.npi-tu.ru/

Дополнительная учебная литература

3. Голубинцев В.О. Философия : лекции, учебное пособие / Любченко В.С. - Новочеркасск : ЮРГТУ (НПИ), 2011. - 142 с.-Режим доступа: http://lib.npi-tu.ru/
4. Яновская С.П. Человек в социокультурном пространстве современной цивилизации : учебное пособие по спецкурсу / С. П. Яновская, В. О. Голубинцев, Н. О. Белоусова ; Юж.-Рос. Гос. Техн. Ун-т (НПИ). - 2-е изд. - Новочеркасск : ЮРГТУ (НПИ), 2012. - 120 с..- Режим доступа: http://lib.npi-tu.ru/
5. Гуревич П. С. Философия [Электронный ресурс]. – Директ-Медиа 2013 г. 539 с.- Режим доступа: http:// www.knigafund.ru/

- ***периодические издания***

6. «Вопросы философии» [Электронный ресурс]: журнал. - Доступ http: // www.elibrary.ru/
7. Вестник Московского университета. Серия 7: Философия [Электронный ресурс]: журнал. - Доступ http: // www.elibrary.ru/

Методические указания к семинарским занятиям

8. Белоусов А.А. Учебно- методическое пособие к изучению курса «Философия» /А.А.Белоусов, Н.О.Белоусова; ЮРГТУ(НПИ).- Новочеркасск: Изд-во ЮРГТУ(НПИ), 2013. – 247с.
9. Воронцова Т. Н. Учебно-методическое пособие к семинарским занятиям по базовому курсу философии. / Т. Н. Воронцова, В. С. Любченко, Л. В. Низовцева; ЮРГТУ(НПИ). - 2-е изд.. - Новочеркасск : Изд-во ЮРГТУ(НПИ), 2012. - 72 с.

Интернет-ресурсы.

12. Журнал «Философия и общество»: Доступ http: // www.elibrary.ru/

13. Вестник «Московского городского педагогического университета. Серия: Философские науки»: Доступ http: // www.elibrary.ru/
14. http://school-collection.edu.ru/ - федеральное хранилище Единая коллекция цифровых образовательных ресурсов
15. http://www.edu.ru/ - федеральный портал Российское образование
16. http://www.igumo.ru/ - интернет-портал Института гуманитарного образования и информационных технологий
17. http://elibrary.ru/defaultx.asp - научная электронная библиотека «Elibrary»
18. www.gumer.info – библиотека Гумер
19. http://anthropology.ru – электронный журнал «Философская антропология»
20. http://iph.ras.ru - Философский журнал Института Философии Российской Академии Наук
21. http://phenomen.ru - философия онлайн

Оглавление

ПЛАНЫ СЕМИНАРСКИХ ЗАНЯТИЙ ПО ФИЛОСОФИИ.....5

Тема 1. Мировоззрение, его виды. Философия как форма мировоззрения.....5

Тема 3. Общество как система. Социальная структура общества. Понятие классов и страт 10

Тема 4. Типы общества по экономической основе и по социокультурным характеристикам 12

Тема 5. Содержание и основные направления философии истории. Формационная концепция социального развития 14

Тема 6. Концепции «индустриализма» и «постиндустриализма».....16

Тема 7. Цивилизационный подход к истории человечества 19

Тема 8. Проблема человека в философии 21

Тема 9. Глобальные проблемы современности 24

ТЕМЫ ЗАНЯТИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ.....26

Тема 1. Изменение предмета философии в ходе ее исторического развития 26

Тема 2. Натурфилософия античности. Учение о первоначалах мира. Античная атомистика.

Учение о стихийной диалектике в античной философии.....27

Тема 2.1 Классическая древнегреческая философия.Сократ. Платон. Аристотель.....29

Тема 2.2 Философия эпохи эллинизма.Скептицизм. Эпикуреизм. Стоицизм. Скептическая установка в теории познания.....30

Тема 3. Средневековая философия, ее характерные черты. Патристика. Основные проблемы средневековой схоластики.....31

Тема 3.1 Проблема соотношения веры и разума в средневековой философии 33

Тема 4. Философия и наука эпохи Возрождения 33

Тема5. Философия Нового времени 34

Тема 6. Антропологические идеи французского Просвещения.Социальные взгляды Ф.Вольтера и Ж.-Ж. Руссо.....36

Тема 6.1 Этическое учение И.Канта.Категорический императив и нравственный долг человека.....37

Тема 6.2 Абсолютная идея Г. Гегеля как смысл исторического процесса 38

Тема 7. Философские аспекты труда, частной собственности и отчуждения, идея коммунизма в философии К.Маркса 39

Тема 7.1 Проблема культурного самоопределения России в отечественной философии. .....

Философия русского зарубежья.....41

Тема 8. Современная западная философия. Экзистенциализм.....43

Тема 8.1 Философия позитивизма и его исторические формы.....44

Тема 8.2 Психоанализ и его эволюция 45

Тема 9. Глобальные научные революции в истории естествознания. Понятия «картина мира» и «парадигма».....47

Тема 10. Наука как особый вид деятельности 50

Тема 10.1 Закономерности развития науки.....51

Тема 10.2 Синергетика как новое мировидение XXI в. Мир как самоорганизующаяся система.....52

Тема 11. Идеальность сознания.Концепции идеального в отечественной философии.....54

Тема 12. Проблема познаваемости мира в истории европейской философии 55

Тема 12.1 Проблема истины. Характеристики истины.Многообразие концепций истины и ее критерии.....56

Тема 13. Основные категории морали, их ценностная характеристика.Общечеловеческие ценности 58

Тема 14. Глобализация как ведущая тенденция развития современного мира: основные направления и проблемы 59

Контрольные вопросы к зачёту.....61

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ 62

Учебно-методическое издание

Воронцова Татьяна Николаевна

ФИЛОСОФИЯ

Учебно-методическое пособие
для подготовки к семинарским занятиям
и организации самостоятельной работы студентов

Подписано в печать 30.06.2017
Формат 60х84 1/16. Бумага офсетная. Печать цифровая.
Усл. печ. л. 3,95. Уч.-изд.л. 4,0 Тираж 50. Заказ

Южно-Российский государственный
политехнический университет
(НПИ) им. М.И.Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул Первомайская, 166
Idp-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

ИНФОРМАТИКА.
ОСНОВЫ АЛГОРИТМИЗАЦИИ

Учебно-методическое пособие
по изучению дисциплины «Информатика»
для направлений подготовки «Бизнес-информатика», «Информатика и вычислительная техника», «Математическое обеспечение и администрирование информационных систем», «Прикладная математика» и «Программная инженерия»

Новочеркасск
ЮРГПУ (НПИ)
2020

УДК 681.3 (075)
ББК 32.973.233
Г85

Рецензент – доктор технических наук, профессор,
заведующий кафедрой «Прикладная математика»
Ткачев Александр Николаевич

Гринченков Д.В., Кущий Д.Н.

Г85 **Информатика. Основы алгоритмизации** : учебно-методическое пособие по изучению дисциплины «Информатика» для направлений подготовки «Бизнес-информатика», «Информатика и вычислительная техника», «Математическое обеспечение и администрирование информационных систем», «Прикладная математика» и «Программная инженерия» / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2020. – 80 с.

Издание содержит разделы, посвященные введению в информатику, основам алгоритмизации и языкам программирования. Рассмотрены вопросы представления структуры информатики, определены предмет и задачи дисциплины. Описаны способы представления алгоритмов и базовые алгоритмические конструкции императивного программирования. Рассмотрена классификация языков программирования. Даны методические указания по началу работы с несколькими интегрированными средами разработки.

УДК 681.3 (075)
ББК 32.973.233

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ В ИНФОРМАТИКУ 5

1.1. Основные понятия 5

1.2. Структура информатики 6

1.3. Предмет и задачи информатики 11

1.4. Вопросы для самостоятельного изучения 12

2. ОСНОВЫ АЛГОРИТМИЗАЦИИ 12

2.1. Алгоритм и его свойства 12

2.2. Формальное определение алгоритма 14

2.2.1. Рекурсивные функции 15

2.2.2. Машина Тьюринга 19

2.2.3. Нормальные алгоритмы А.А. Маркова 22

2.3. Основные способы представления алгоритмов 24

2.4. Базовые алгоритмические конструкции 28

2.4.1. Алгоритмы линейной структуры 29

2.4.2. Разветвляющиеся алгоритмы 31

2.4.3. Циклические алгоритмы 35

2.5. Понятие сложности алгоритма 37

2.5.1. Расчет временной сложности 39

2.5.2. Асимптотическая сложность 40

2.5.3. Классы сложности алгоритмов 42

2.6. Задачи для самоконтроля 44

2.7. Вопросы для самостоятельного изучения 45

3. ЯЗЫКИ И СРЕДЫ ПРОГРАММИРОВАНИЯ 46

3.1. Задачи, языки и программы 46

3.2. Трансляция 51

3.3. Основные парадигмы программирования 52

3.4. Обзор сред разработки для языков C/C++ 55

3.4.1. Microsoft Visual Studio 56

3.4.2. Xcode 56

3.4.3. Eclipse 57

3.4.4. Qt Creator 57

3.4.5. Code::Blocks 58

3.4.6. Dev-C++ 58

3.5. Краткое практическое руководство 59

3.5.1. Создание консольного приложения в среде Dev-C++ 59

3.5.2. Создание консольного приложения в среде Code::Blocks 61

3.5.3. Создание консольного приложения в среде Visual Studio 64

3.6. Вопросы для самостоятельного изучения 67

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 68

ПРИЛОЖЕНИЕ 1 Служебные слова псевдокода 70

ПРИЛОЖЕНИЕ 2 Основные элементы блок-схем 72

ПРИЛОЖЕНИЕ 3 Примеры стилей программирования 76

1. ВВЕДЕНИЕ В ИНФОРМАТИКУ

1.1. Основные понятия

Информатика – это комплексная, техническая наука, основанная на использовании компьютерной техники, изучающая структуру и общие свойства информации, а также закономерности и методы ее создания, хранения, поиска, преобразования, передачи и применения в различных сферах человеческой деятельности.

ИНФОРМАТИКА = ИНФОРМАЦИЯ + АВТОМАТИКА

Термин «информатика» происходит от французских слов *information* (информация) и *automatique* (автоматика) и дословно означает информационная автоматика. Данное понятие появилось во Франции в середине 60-х годов XX века с началом массового использование вычислительной техники.

Информация – сведения о чем-либо, независимо от формы их представления.

Автоматика – отрасль науки и техники, которая разрабатывает технические средства и методы для осуществления технологических процессов без непосредственного участия человека.

Примечание. Несмотря на широкую распространенность, понятие информации остается одним из самых дискуссионных в науке, а термин может иметь различные значения в разных отраслях человеческой деятельности (рис.1.1).

Рис. 1.1. Виды информации

В англоязычных странах, особенно в США, вместо термина «*informatics*» обычно используют «*computer science*», то есть компьютерные науки. Данное понятие сформировалось в начале 40-х годов XX века.

КОМПЬЮТЕРНЫЕ НАУКИ = ЭВМ + ЛОГИКА + АЛГОРИТМЫ

Компьютерные науки представляли собой на тот момент соединение возможностей электронно-вычислительных машин (ЭВМ) того времени, математической логики и теории алгоритмов. В дальнейшем в области компьютерных наук появлялись новые направления. Сегодня, по разным классификациям, выделяют несколько десятков направлений в данной области.

На сегодняшний день многие специалисты считают термины «информатика» и «компьютерные науки» синонимами, хотя имеются и противники данной трактовки, считающие первое понятие более многогранным, нежели второе.

Слово «*computer*» появилось в английском литературном языке в начале XVII века, на тот момент оно означало «человек, занимающийся вычислениями». В конце XIX века у этого слова появилось второе значение «машина-вычислитель», но лишь в середине XX века первое значение было окончательно вытеснено новой трактовкой. На сегодняшний день «*computer*» означает в английском языке любую вычислительную машину: аналоговую, цифровую, гибридную и др.

Слово ЭВМ (точнее, ЭСМ, электронная счетная машина) появилось в СССР в 40-х годах XX века, т.е. в то же самое время, когда за словом «*computer*» в английском языке закрепилось значение машины-вычислителя. Однако с самого начала сокращение ЭВМ подразумевало не любую машину, а электронную.

В отличие от бытового языка, в современном научном, юридическом и техническом русском языке ЭВМ и компьютер – одно и то же.

1.2. Структура информатики

Комплексность информатики приводит к существованию различных способов представления ее структуры.

В узком смысле информатика может быть представлена в виде совокупности *технических, программных и алгоритмических средств преобразования информации* (рис. 1.2). Каждую из этих частей обычно рассматривают с разных позиций как: производственную отрасль, фундаментальную науку, прикладную дисциплину.

Рис. 1.2. Структура информатики. Вариант 1

Информатика как производственная отрасль состоит из однородной совокупности предприятий разных форм хозяйствования, которые занимаются производством компьютерной техники, программных продуктов и разработкой современной техноло-

гии переработки информации. Специфика и значение информатики как отрасли производства состоят в том, что от нее во многом зависит рост производительности труда в других отраслях народного хозяйства.

Информатика как фундаментальная наука занимается разработкой методологии создания информационного обеспечения процессов управления любыми объектами на базе компьютерных информационных систем (разработка сетевой структуры, компьютерно-интегрированные производства, экономическая и медицинская информатика, информатика социального страхования и окружающей среды, профессиональные информационные системы).

Информатика как ***прикладная дисциплина*** занимается:

- изучением закономерностей в информационных процессах (накопление, переработка, распространение);
- созданием информационных моделей коммуникаций в различных областях человеческой деятельности;
- разработкой информационных систем и технологий в конкретных областях и выработкой рекомендаций относительно их жизненного цикла: для этапов проектирования и разработки систем, их производства, функционирования и т.д.

Информатика в широком смысле представляет собой единство разнообразных отраслей науки, техники и производства, связанных с переработкой информации (рис. 1.3).

Теоретическая информатика **–** часть информатики, занимающаяся изучением структуры и общих свойств информации и информационных процессов, разработкой общих принципов построения информационной техники и технологии.

Искусственный интеллект – направление информатики, занимающееся созданием методов и технологий программирования разумности и передача этих умений различным техническим устройствам.

Прикладная информатика – высокотехнологичная, наукоемкая и инновационная сфера деятельности, использующая последние достижения в области информационно-коммуникационных технологий и смежных областях: АСНИ –

автоматизированные системы для научных исследований; САПР – системы автоматизированного проектирования; АИС – автоматизированные информационные системы; АСУ – автоматические системы управления; АОС – автоматизированные обучающие системы и др.

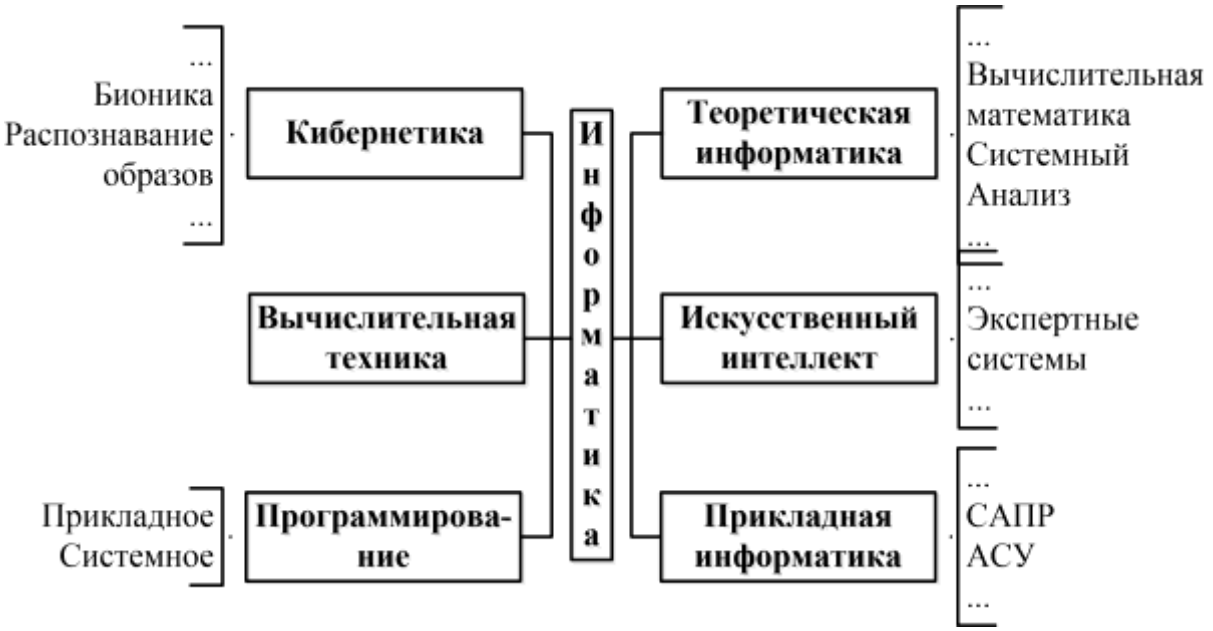


Рис. 1.3. Структура информатики. Вариант 2

Программирование как научное направление возникло с появлением вычислительных машин и только программное обеспечение определяет эффективность их использования.

Вычислительная техника – раздел, в котором разрабатываются общие принципы построения архитектуры вычислительных систем, определяющей состав, назначение, функциональные возможности и принципы взаимодействия устройств.

Кибернетика может рассматриваться как прикладная информатика в области создания и использования автоматических или автоматизированных систем управления разной степени сложности, от управления отдельным объектом (станком, промышленной установкой, автомобилем и т. п.) до сложнейших систем управления целыми отраслями промышленности, банковскими системами, системами связи и даже сообществами людей.

Путем обобщения вышеизложенных подходов к определению структуры информатики может быть получена двухуровневая схема, учитывающая такие составляющие, как *информационные процессы*, *аппаратное* и *программное обеспечение* (рис. 1.4).

Рис. 1.4. Структура информатики. Вариант 3

В рамках изучаемой дисциплины информатика будет рассматриваться как прикладная дисциплина и как отрасль производства.

1.3. Предмет и задачи информатики

Проанализировав различные подходы к определению структуры информатики, можно сказать, что ее предмет как науки составляют:

- аппаратное обеспечение средств вычислительной техники;
- программное обеспечение средств вычислительной техники;
- средства взаимодействия аппаратного и программного обеспечения;
- средства взаимодействия человека с аппаратными и программными средствами.

Основная задача информатики как науки – это систематизация приемов и методов работы с аппаратными и программными средствами вычислительной техники для разработки способов и средств преобразования информации.

Основные направления применения информатики:

- разработка вычислительных систем и программного обеспечения;
- теория информации, изучающая процессы, связанные с передачей, приемом, преобразованием и хранением информации;
- методы искусственного интеллекта, позволяющие создавать программы для решения задач, требующих определенных интеллектуальных усилий при выполнении их человеком (логический вывод, обучение, понимание речи и другие);
- системный анализ, заключающийся в анализе назначения проектируемой системы и в установлении требований, которым они должны отвечать;
- методы машинной графики, анимации, средств мультимедиа;
- средства телекоммуникации, в том числе, глобальные коммуникационные сети, объединяющие все человечество в единое информационное сообщество;
- разнообразные приложения, охватывающие производство, науку, образование, медицину, торговлю, сельское хозяйство и все другие виды хозяйственной деятельности.

На всех этапах технического обеспечения информационных процессов для информатики ключевым понятием является эф-

фективность. Для аппаратных средств под эффективностью понимают отношение производительности оборудования к его стоимости. Для программного обеспечения под эффективностью понимают производительность лиц, работающих с ними (пользователей).

1.4. Вопросы для самостоятельного изучения

1. Информационные процессы: хранение, обработка и передача информации в технических системах.
2. Представление чисел в памяти вычислительных машин: биты, байты, машинные слова, беззнаковые числа и числа со знаком.
3. Кодирование текстовых данных.
4. Представление звуковых данных памяти компьютера: дискретизация, квантование и кодирование звука.
5. Виды компьютерной графики: растровая, векторная и фрактальная графики.
6. Законы Грассмана. Цветовые модели *RGB* и *CMYK*.
7. Принципы оцифровки изображения.
8. Представление видеоданных в памяти компьютера.
9. Объем информации. Формулы Хартли и Шеннона.
10. Структура информационно-поисковой системы.
11. Сферы применения искусственного интеллекта.
12. Классификация вычислительной техники.
13. Архитектура вычислительных систем: состав, назначение, функциональные возможности и принципы взаимодействия устройств.
14. Прикладное и системное программное обеспечение.

2. ОСНОВЫ АЛГОРИТМИЗАЦИИ

2.1. Алгоритм и его свойства

На сегодняшний день понятие алгоритма является центральным понятием информатики. Термин «алгоритм» своим происхождением связан с именем узбекского математика Аль-Хоремзи,

который в IX веке сформулировал правила выполнения четырех арифметических действий.

Единого «истинного» определения понятия «алгоритм» на настоящий момент не существует. Наиболее известные варианты определения опираются на интуитивное понятие «задачи»:

Алгоритм – это конечный набор правил, который определяет последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, определенность, ввод, вывод, эффективность (Д.Э. Кнут).

Алгоритм – это всякая система вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи (А.Н. Колмогоров).

Алгоритм – конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи (Википедия).

Различные определения алгоритма в явной или неявной форме содержат следующий ряд общих требований:

1. *Дискретность* – разбиение процесса обработки информации на более простые этапы (шаги выполнения), выполнение которых компьютером или человеком не вызывает затруднений.
2. *Детерминированность* (*определенность*) – однозначность выполнения каждого отдельного шага преобразования информации. Алгоритм выдает один и тот же результат (ответ) для одних и тех же исходных данных
3. *Понятность* – алгоритм должен включать только команды из заранее оговоренной системы команд исполнителя.
4. Завершаемость (конечность, выполнимость) – при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
5. *Массовость* (*универсальность*) – пригодность алгоритма для решения определенного класса задач при разных наборах начальных данных.

Алгоритмизация – это метод описания систем или процессов путем составления алгоритмов их функционирования.

В алгоритме отражаются логика и способ формирования результатов решения с указанием необходимых расчетных формул, логических условий, соотношений для контроля достоверности выходных результатов. В алгоритме обязательно должны быть предусмотрены все ситуации, которые могут возникнуть в процессе решения комплекса задач.

Алгоритм решения комплекса задач и его программная реализация тесно взаимосвязаны. Специфика применяемых средств проектирования алгоритмов и используемых при этом инструментальных средств разработки программ может повлиять на форму представления и содержание алгоритма обработки данных.

2.2. Формальное определение алгоритма

Длительное время интуитивного понятия алгоритма было вполне достаточно для задач математики и логики. Пусть имеется некоторая алгоритмическая проблема – необходимо найти алгоритм для решения определенного класса задач. Если такой алгоритм найден, то предложенный способ для решения задачи признается всеми как алгоритм, исходя из их собственного интуитивного представления об алгоритме. В этом случае не требуется точного определения понятия алгоритма.

Однако ситуация становится противоположной, если после длительных, безуспешных попыток нахождения алгоритма возникает гипотеза, что такого алгоритма не существует. Для осуществления доказательства отсутствие алгоритма необходимо математически точно определить объект, существование которого будет опровергаться.

В 20-х годах XX века задача точного определения понятия алгоритма стала одной из центральных математических проблем. Она заключалась в попытке дать строгое математическое определение алгоритма, которое соответствовало бы имевшимся в то время интуитивным представлениям об алгоритме.

Решение данной проблемы было получено в первой половине XX века. Рассмотрим варианты формального определения алгоритма.

2.2.1. Рекурсивные функции

С точки зрения А. Черча и С. Клини, в алгоритмическом процессе вычисляется значение некоторой функции $f(x_1, x_2, ..., x_n)$, определенной на множестве натуральных чисел.

Строгое определение алгоритма, предложенное этими математиками, основано на понятии частично рекурсивной функции. Они предложили отождествить интуитивное понятие «алгоритм» со строгим математическим понятием «частично рекурсивная функция».

Функцию вида: $f:X \to Y$ будет называться *частичной*, если она определена не для всех $x \in X$. Совокупность тех элементов множества X, для которых определены элементы в Y, называется *областью определения функции*, а совокупность тех элементов Y, которые соответствуют элементам X, называют *областью значений функции*. Если область определения функции из X в Y совпадает с множеством X, то функция называют *всюду определенной*.

Функция $y = f(x_1, x_2, \ldots, x_n)$ ***называется эффективно вычислимой***, если существует алгоритм, позволяющий вычислить ее значение.

Понятие алгоритма в этом определении берется в интуитивном смысле, поэтому и понятие эффективно вычислимой функции оказывается интуитивным, однако при переходе от алгоритмов к вычислимым функциям возникает одно очень существенное обстоятельство, а именно: совокупность процессов, удовлетворяющих требованиям предъявляемым к алгоритмам, достаточно обширно и не поддается полному описанию, совокупность же вычислимых функций удовлетворяющих перечисленным условиям ограничена и легко описываемой в обычных математических терминах.

Частично рекурсивная функция – это числовая функция, которая строится следующим образом. Определяется конечное

число простейших функций и операций. Выполнимость функций должна быть очевидна, а с помощью операций из них можно порождать другие, более сложные функции.

Простейшие функции следующие:

1) $f(x) = x + 1$ – *оператор сдвига* или *функция следования*, позволяет получить любое число натурального ряда;
2) $O(x) = 0$ – *оператор аннулирования* или *константа* 0;
3) $I_n^m(x_1, x_2, ..., x_n) = x_m$, где $1 \le m \le n$. Выражение I_n^m – *называется оператором проектирования* или *селекторной функцией.*

Каждая из функций I_n^m равна одному из своих аргументов. Индекс *m* выделяет *m*-ю переменную.

Как указано в определении, перечисленные простейшие функции всюду определены и интуитивно вычислимы. Над ними можно выполнить ряд операций (часто используется термин *оператор*), которые обладают тем свойством, что их применение к функциям, вычислимым в интуитивном смысле, порождает новые функции, также заведомо вычислимые в интуитивном смысле.

Набор операторов, обеспечивающих преобразование простейших функций следующий:

1. ***Суперпозиция функций.*** Пусть задана функция $h(x_1, x_2, ..., x_m)$ и функции $q_1(x_1, x_2, ..., x_n)$, $q_2(x_1, x_2, ..., x_n)$, …, $q_m(x_1, x_2, ..., x_n)$, тогда говорят, что функция $\varphi(x_1, x_2, ..., x_n)$ получена суперпозицией из функций *h* и q_1, q_2, …, q_m, если имеет место следующее равенство:

$$\varphi(x_1, x_2, ..., x_n) = h\left(q_1(x_1, x_2, ..., x_n), q_2(x_1, x_2, ..., x_n), ..., q_m(x_1, x_2, ..., x_n)\right)$$

2. ***Оператор примитивной рекурсии.*** Говорят, что функция $f(x_1, x_2, ..., x_n, y)$ получена из функций $h(x_1, x_2, ..., x_n)$ и $\varphi(x_1, x_2, ..., x_n, y, z)$ с помощью оператора примитивной рекурсии, если она может быть задана схемой примитивной рекурсии:

$$\begin{cases} f(x_1, x_2, \ldots, x_n, 0) = h(x_1, x_2, \ldots, x_n) \\ f(x_1, x_2, \ldots, x_n, y+1) = \phi\left(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y)\right) \end{cases}.$$

Приведенная выше схема справедлива для общего случая *n*-местной функции, т.е. когда $n \geq 1$, в том случае, когда $n = 0$ схема примитивной рекурсии имеет следующий вид:

$$\begin{cases} f(0) = a, \\ f(y+1) = \phi(y, f(y)), \end{cases}$$

где a – постоянная одноместная функция, равная числу a (константе).

Частичная функция $f(x_1, ..., x_n)$ называется ***примитивно рекурсивной***, если ее можно получить конечным числом операций суперпозиции и примитивной рекурсии, исходя лишь из простейших функций.

Если функции h и φ интуитивно вычислимы, то будет интуитивно вычислима и f. Например, если $a_1, a_2, \ldots, a_n$ – это набор аргументов соответствующих $x_1, x_2, \ldots, x_n$, то значения для f могут быть найдены следующим образом:

$f(x_1, x_2, \ldots x_n, 0) = h(x_1, x_2, \ldots x_n) = b_0$;

$f(x_1, x_2, \ldots x_n, 1) = h(x_1, x_2, \ldots x_n, b_0) = b_1$;

$f(x_1, x_2, \ldots x_n, 2) = h(x_1, x_2, \ldots x_n, 1, b_1) = b_2$ и т.д.

Очевидно, что функция f будет всюду определена, если всюду определены функции h и φ.

3. ***Операция минимизации (μ-оператор).*** Пусть задана некоторая функция $f(x, y)$. Фиксируя значение x выясняем, при каком y $f(x, y) = 0$.

Так как результат решения задачи зависит от x, то наименьшее значение y, при котором имеет место, $f(x, y) = 0$ является функцией от x, поэтому используется следующие обозначение:

$$\varphi(x) = \mu y\left[f(x, y) = 0 \right].$$

Приведенная запись трактуется как следующие утверждения: наименьшее y такое, что $f(x, y) = 0$.

Исходя из выше изложенного, можно определить функцию и для нескольких переменных:

$$\varphi(x_1, x_2, ..., x_n) = \mu y\left[f(x_1, x_2, ..., x_n) = 0 \right].$$

Переход от функции $f(x_1, x_2, ..., x_n, y)$ к функции $\varphi(x_1, x_2, ..., x_n)$ принято называть ***применением μ-оператора***.

Алгоритм вычисления функции φ можно представить в следующем виде.

Шаг 1. Вычисляется $f(x_1, x_2, ..., x_n, 0)$. Если $f(x_1, x_2, ..., x_n, 0) = 0$, то полагается $\varphi(x_1, x_2, ..., x_n) = 0$. Если $f(x_1, x_2, ..., x_n, 0) \neq 0$, то выполняется переход на шаг 2.

Шаг 2. Вычисляется $f(x_1, x_2, ..., x_n, 1)$.

Если $f(x_1, x_2, ..., x_n, 1) = 0$, то полагается $\varphi(x_1, x_2, ..., x_n) = 1$, иначе выполняется переход на шаг 3.

Шаг 3. Берется следующее значение *y* и повторяется действие, аналогичное предшествующему шагу.

Шаг N. Если перебраны все значения *y* и для любого из них $f(x_1, x_2, ..., x_n, y) \neq 0$, то функция $\varphi(x_1, x_2, ..., x_n)$ в этом случае считается неопределенной.

Пример 2.1. Рассмотрим функцию $f(x, y) = x - y$, которая может быть получена с помощью оператора минимизации:

$$f(x, y) = \mu z(y + z = x) = \mu z\left[I_3^2(x, y, z) + I_3^3(x, y, z) = I_3^1(x, y, z) \right].$$

Вычислим, например, $f(7, 2)$, т.е. значение функции при $y = 2$ и $y = 7$. Положим *y* = 2, а *x* будем придавать последовательные значения

z = 0, 2+0=2≠7,
z = 1, 2+1=3≠7,
z = 2, 2+2=4≠7,
z = 3, 2+3=5≠7,
z = 4, 2+4=6≠7,
z = 5, 2+5=7=7.

Таким образом, найдено значение функции $f(7, 2) = 5$.

Функция $f(x_1, ..., x_n)$ называется ***частично-рекурсивной***, если она может быть получена в конечное число шагов из простейших функций при помощи операций суперпозиции, схем примитивной рекурсии и μ-оператора исходя лишь из простейших функций.

Функция $f(x_1, ..., x_n)$ называется ***общерекурсивной***, если она частично рекурсивна и всюду определена.

Общерекурсивными можно, например, считать следующие функций:

1. $\lambda(x)$.
2. $O(x)$.
3. $J_n^m(x)$.
4. $f(y,x) = y + x$.
5. $f(y,x) = y \cdot x$.
6. $f(y,x) = y + x$.

Как видно из приведенных выше определений класс частично рекурсивных функций шире класса примитивно рекурсивных функций, так как все примитивно рекурсивные функции являются всюду определенными, а среди частично рекурсивных функций встречаются функции не всюду определенные, а также нигде не определенные.

2.2.2. Машина Тьюринга

С точки зрения английского математика А. Тьюринга алгоритмический процесс – работа некоторой воображаемой вычислительной машины. Он предложил отождествить интуитивное понятие «алгоритм» с точным понятием «машина Тьюринга» (МТ).

МТ состоит из следующих структурных элементов: ленты, автомата и устройства обращения к ленте (рис. 2.1).

Лента используется для хранения информации. Она бесконечна в обе стороны и разбита на клетки, которые никак не нумеруются и не именуются. В каждой ячейке должен быть записан один из символов конечного алфавита из множества $A = \{a_1, a_2, \ldots, a_j, \ldots, a_m\}$. В пустых ячейках записывается знак пробела λ. Содержимое клетки может меняться – в нее можно записать другой символ или стереть находящийся там символ. Множество A называется внешним алфавитом МТ.

Автомат (управляющее устройство) может находиться в одном из состояний, образующих множество $Q = \{q_1, q_2, \ldots, q_i, \ldots, q_n\}$. Множество Q называется внутренним алфавитом МТ. Для любой

МТ число состояний конечно и фиксировано. Множество Q выступает в качестве внутренней памяти.

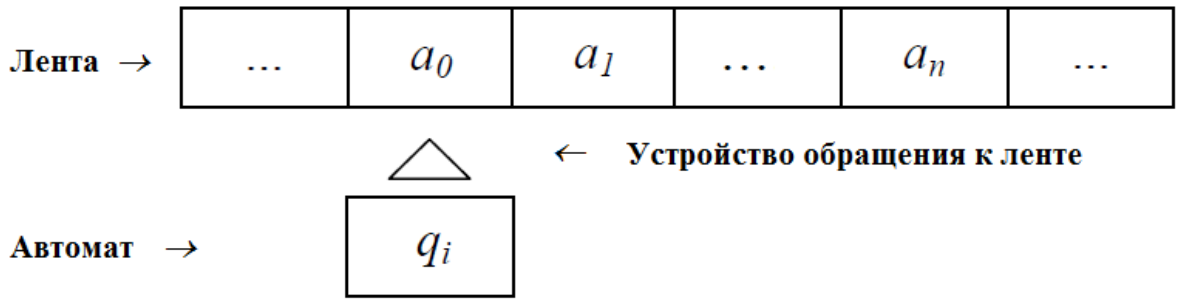


Рис. 2.1. Схема устройства машины Тьюринга

Устройство обращения к ленте – головка, осуществляющая считывание и запись данных. В общем случае МТ работает следующим образом:

1. Считывается значение символа a_j, над которым устанавливается головка.
2. В зависимости от пары значений $q_i a_j$ производится следующая последовательность действий:

а) запись нового значения $a_j^{'}$ в ту ячейку, над которой стоит головка;

б) смещение головки на одну ячейку вправо или влево;

в) переход в новое состояние $q_i^{'}$.

Реализация указанной последовательности действий называется ***тактом***, т.е. работа МТ состоит из отдельных тактов.

В зависимости от того, какая была записана исходная информация на ленте, возможны два варианта работы МТ. Пусть на ленте записана информация B_0 (построенная из символов A), тогда:

1. После конечного числа тактов МТ приходит в заключительное состояние – q_z и на ленте изображена некоторая информация В, также построенная из символов A. В этом случае говорят, что МТ применима к B_0 и перерабатывает ее в B.
2. Если МТ не приходит в состояние q_z, то говорят, что машина не применима к B_0.

Конкретная машина Тьюринга задается перечислением элементов множеств A и Q, а также набором правил преобразования. Очевидно, что различных множеств A, Q и правил преобразования может быть бесконечно много, т.е. и машин Тьюринга также может быть бесконечно много.

Правила преобразования, выполняемые МТ, определяются системой команд, которые описываются следующим образом:

$$q_i a_j \to q_i^{'} a_j^{'} d_k ,$$

где d_k – смещение (R – вправо, L – влево, E – без изменений).

Пример 2.2. Задан алфавит $A=\{a, b, c\}$. Перенести первый символ непустого слова P в его конец.

Решение:

1. Запомнить первый символ слова P, а затем стереть этот символ.
2. Передвинуть автомат вправо под первую пустую клетку за P и записать в нее запомненный символ.

Если символ в ячейке или не меняется, то они повторно в ячейке не указываются. При переходе в заключительное состояние (символ «!») направление движения не указывается.

Q/A	a	b	c	λ	Комментарий
q_1	q_2, λ, R	q_3, λ, R	q_4, λ, R	,, R	анализ 1-го символа, его удаление, разветвление
q_2	,, R	,, R	,, R	!, a,	запись a справа
q_3	,, R	,, R	,, R	!, b,	запись b справа
q_4	,, R	,, R	,, R	!, c,	запись c справа

Пример 2.3. Задан алфавит $A=\{a, b\}$. Удалить из слова P его второй символ, если такой есть.

Решение:

1. Запомнить первый символ, стереть его, сдвинуться на ячейку вправо.
2. Записать «запомненный» символ в ячейку.

Q/A	a	b	λ	Комментарий
q_1	q_2, λ, R	q_3, λ, R	, , !	Анализ и удаление 1-го символа, разветвление, сдвиг
q_2	!, ,	!, a,	!, a,	Замена второго символа на a
q_3	!, b,	!, ,	!, b,	Замена второго символа на b

Полное состояние МТ, однозначно определяющее ее дальнейшее поведение, называется ***конфигурацией*** или ***машинным словом***. Конфигурация определяется следующим образом: $K_i \to a_L q_i a_R$, где q_i – текущее состояние; a_L – последовательность символов, записанных слева; a_R – последовательность символов, записанных справа, включая ячейку, над которой находится головка. Вся совокупность входных конфигураций, при которых машина обеспечивает получение результата, образуют *класс решаемых задач*.

2.2.3. Нормальные алгоритмы А.А. Маркова

Третий вариант определения алгоритма предложил в 1947 г. российский математик А.А. Марков. С его точки зрения, алгоритмический процесс – это переработка слов некоторого алфавита с помощью точно описанных правил подстановки. Вместо интуитивного понятия «алгоритм» А.А. Марков вводит строгое понятие «нормальный алгори***ф***м».

Формулой подстановки называется запись вида α→β (читается «α заменить на β»), где α и β – любые слова (возможно, и пустые). При этом α называется левой частью формулы, а β – правой частью.

Сама ***подстановка*** (как действие) задается формулой подстановки и применяется к некоторому слову P. Суть операции сводится к тому, что в слове P отыскивается часть, совпадающая с левой частью этой формулы (т.е. с α), и она заменяется на правую часть формулы (т.е. на β). При этом остальные части слова P (слева и справа от α) не меняются. Получившееся слово R называют ***результатом подстановки***.

Если левая часть формулы подстановки входит в слово P, то говорят, что эта формула применима к P. Но если α не входит в P, то формула считается неприменимой к P, и подстановка не выполняется.

Если левая часть α входит в β несколько раз, то на правую часть β, по определению, заменяется только первое (самое левое) вхождение α в P.

Если правая часть формулы подстановки – пустое слово (в таком слове нет ни одного символа), то подстановка $\alpha\rightarrow$ сводится к вычеркиванию части α из P (в формулах подстановки не принято как-либо обозначать пустое слово).

Если в левой части формулы подстановки указано пустое слово, то подстановка $\rightarrow\beta$ сводится, по определению, к приписыванию β слева к слову P.

Нормальным алгоритмом Маркова (НАМ) называется непустой конечный упорядоченный набор формул подстановки:

$$\begin{cases} \alpha_1 \rightarrow \beta_1 \\ \alpha_2 \rightarrow \beta_2 \\ \quad \ldots \\ \alpha_k \rightarrow \beta_k \end{cases}, \ k \geq 1.$$

В этих формулах могут использоваться два вида стрелок: обычная стрелка ($\rightarrow$) и стрелка «с хвостиком» ($\mapsto$). Формула с обычной стрелкой называется ***обычной формулой***, а формула со стрелкой «с хвостиком» – ***заключительной формулой***.

Пример 2.4. Задан алфавит $A=\{a, b, c, d\}$. В слове P требуется удалить все вхождения символа c, а затем заменить первое вхождение подслова bb на ddd.

Следует отметить, что в НАМ вставка новых символов в слово – замена некоторого подслова на подслово с большим числом символов.

Решение:

Поскольку НАМ продолжает свою работу, пока применима хотя бы одна формула, нужно принудительно остановить его после того, как он заменил первое вхождение bb.

$$\begin{cases} c \to (1) \\ bb \mapsto ddd\,(2) \end{cases}.$$

Пример 2.5. A={0, 1, 2, 3}. Пусть P – непустое слово. Трактуя его как запись неотрицательного целого числа в четверичной системе счисления, требуется получить запись этого же числа, но в двоичной системе.

Решение:

Для перевода числа из четверичной системы в двоичную надо каждую четверичную цифру заменить на пару соответствующих ей двоичных цифр: 0→00, 1→01, 2→10, 3→11.

Необходимо отделить часть числа, в которой уже была произведена замена, от той, где замены ещё не было. Для этого следует пометить слева спецзнаком «*» (или любым другим символом) ту четверичную цифру, которая сейчас должна быть заменена, а после того как такая замена будет выполнена, спецзнак нужно поместить перед следующей четверичной цифрой.

Система подстановок в итоге примет вид:

$$\begin{cases} *0 \to 00*(1) \\ *1 \to 01*(2) \\ *2 \to 10*(3) \\ *3 \to 11*(4) \\ * \mapsto (5) \\ \to *(6) \end{cases}.$$

2.3 Основные способы представления алгоритмов

Для записи алгоритма решения задачи применяются следующие изобразительные способы их представления:

- словесный (формульно-словесный);
- структурно-стилизованный (псевдокод);
- операторная схема;
- графический.

Словесный способ записи алгоритма характеризуется тем, что описание осуществляется с помощью слов и формул. Содер-

жание последовательности этапов выполнения алгоритмов записывается на естественном профессиональном языке предметной области в произвольной форме.

Пример 2.6. Найти из трех заданных чисел *a*, *b*, *c* наибольшее.

Шаг 1. Сравнить *a* и *b*. В случае если *a*>*b*, то в качестве максимума *t* принять *a*, иначе (*a*<=*b*) в качестве максимума принять *b* (*t*=*b*).

Шаг 2. Сравнить *t* и *c*. В случае если *t*>*c*, то перейти к шагу 3. Иначе (*t*<*c*) принять в качестве максимума *c* (*t*=*c*).

Шаг 3. Принять *t* в качестве результата.

Словесный способ не имеет широкого распространения, так как такие описания обладают рядом недостатков:

- отсутствие строгой формализации;
- чрезмерная подробность записей;
- возможность неоднозначности при толковании отдельных предписаний.

Псевдокод – компактный, зачастую также неформальный язык описания алгоритмов, использующий ключевые слова императивных языков программирования, но опускающий несущественные для понимания алгоритма подробности и специфический синтаксис.

Главная цель использования псевдокода – обеспечить понимание алгоритма человеком, сделать описание более воспринимаемым, чем исходный код на языке программирования. Псевдокод широко используется в учебниках и научно-технических публикациях, а также на начальных стадиях разработки компьютерных программ.

В отличие от языков программирования, на синтаксис псевдокода не устанавливается стандартов, и автор каждой публикации может применять свой оригинальный псевдокод. На практике авторы обычно заимствуют нужные им конструкции из одного или нескольких широко известных и распространенных языков программирования. Сейчас обычно заимствуют элементы синтаксиса языков *C++*, *C#*, *Java*, в более старых публикациях часто использовался *Algol*.

Из псевдокода исключаются технические элементы, такие как описание переменных, системно-зависимый код, операции выделения и освобождения памяти, если только они не являются существенными элементами рассматриваемого алгоритма. Математические выражения часто включаются в псевдокод в том виде, как их принято записывать в математике, а не в языках программирования, а некоторые фрагменты псевдокода могут быть фразами естественного языка (русского, английского и т. д.).

Основные служебные слова русского алгоритмического языка псевдокода представлены в прил. 1.

Общий вид алгоритма:

```
алг название алгоритма (аргументы и результаты)
  дано условия применимости алгоритма
  надо цель выполнения алгоритма
нач описание промежуточных величин
  | последовательность команд (тело алгоритма)
кон
```

Часть алгоритма от слова «**алг**» до слова «**нач**» называется *заголовком*, а часть, заключенная между словами «**нач**» и «**кон**» – *телом алгоритма*. Предложения «**дано**» и «**надо**» не обязательны. В них рекомендуется записывать утверждения, описывающие состояние среды исполнителя алгоритма. Символ «|» используется для обозначения комментариев или парных слов.

Пример 2.7. Запись программы «Здравствуй, Мир!» с помощью псевдокода.

```
алг ЗДРАВСТВУЙМИР
нач
  вывод «Здравствуй, Мир!»
кон алг ЗДРАВСТВУЙМИР
```

Операторные схемы алгоритмов. Суть этого способа описания алгоритма заключается в том, что каждый оператор обозначается буквой (например, *A* – арифметический оператор, *P* – логический оператор).

Операторы записываются слева направо в последовательности их выполнения, причем, каждый оператор имеет индекс, указывающий порядковый номер оператора. Алгоритм записывается в одну строку в виде последовательности операторов.

Пример записи алгоритма в линейной форме приведен на рисунке 2.2.

Запись $A_i \overset{m}{\uparrow}$ означает, что после выполнения A_i будет выполняться оператор с меткой m. Запись $P_i \overset{m}{\uparrow}$ означает, что после выполнения P_i, в случае его истинности, выполняется следующий оператор, в противном случае осуществляется переход на оператор с меткой m.

$$A_1 A_2 P_3 \overset{10}{\uparrow} A_4 P_5 \overset{8}{\uparrow} A_6 A_7 \overset{13}{\uparrow} A_8 A_9 \overset{13}{\uparrow} A_{10} A_{11} P_{12} \overset{10}{\uparrow} A_{13} A_{14}$$

Рис. 2.2. Линейная форма записи алгоритма

Графический способ описания алгоритма получил самое широкое распространение. Для графического описания алгоритмов используются *структурные схемы алгоритмов* или *блочные символы* (блоки), которые соединяются между собой линиями связи.

Структурная схема представляет собой граф: вершинам соответствуют шаги (действия), а ребрам – переходы, отображающие последовательность выполнения шагов. Алгоритм с рис. 2.2 представлен в этой форме на рис. 2.3.

Блок-схемой называется графическое изображение структуры алгоритма, в которой каждый этап процесса обработки данных представляется в виде графических символов (блоков) – геометрических фигур, в контуры которых вписано содержание операции. Фигура, представленная в виде блочного символа (блока) называется символом действия.

Основные элементы блок-схем алгоритмов представлены в прил. 2.

Правила выполнения блок-схем в программной документации регламентируются в России ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». Указанный ГОСТ практически полностью соответствует международному стандарту *ISO* 5807:1985.

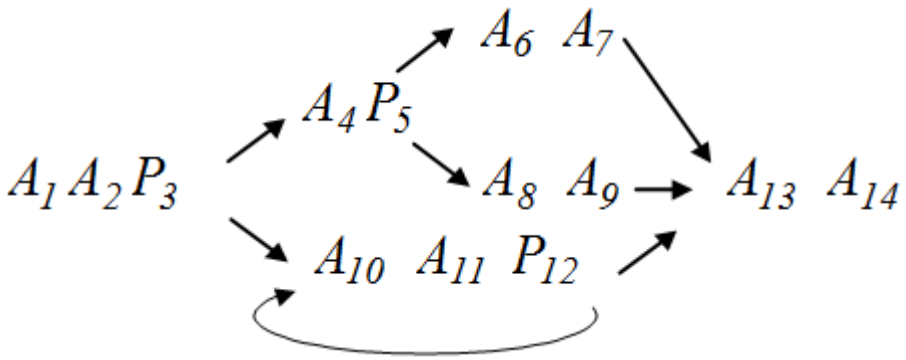


Рис. 2.3. Представление алгоритма в виде структурной схемы

Блок-схемы позволяют описывать программы, разработанные в соответствии с парадигмами структурного и процедурного программирования, и плохо подходят для описания программ построенных по принципам объектно-ориентированного программирования (тема будет рассмотрена в следующей главе). Однако с их помощью можно проиллюстрировать работу основного алгоритма программы, отдельных особо сложных методов какого-либо класса. Часто они оказываются более выразительными и наглядными, чем диаграмма деятельности в *UML*.

Если человек прежде не сталкивался с программированием, то предварительное построение блок-схемы поможет ему с программной реализацией.

В ряде случаев программирование невозможно без рисования блок-схем, т.к. это один процесс (визуальный язык программирования ДРАКОН). Кроме того, блок-схемы используются для верификации алгоритмов (формального доказательства их корректности) методом индуктивных утверждений Флойда.

2.4. Базовые алгоритмические конструкции

Выделяют три основные структуры алгоритмов:

1. Линейная (последовательная).

2. Разветвляющаяся (альтернатива «если–то–иначе» или «если–то»).

3. Циклическая (повторение).

2.4.1. Алгоритмы линейной структуры

Линейный алгоритм (линейная структура) – это такой алгоритм, в котором все действия выполняются последовательно друг за другом и только один раз. Схема представляет собой последовательность блоков, которые располагаются сверху вниз в порядке их выполнения (рис. 2.4). Первичные и промежуточные данные не оказывают влияния на направление процесса вычисления.

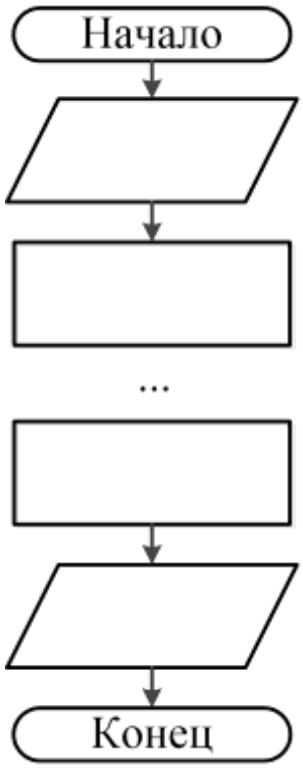


Рис. 2.4. Алгоритм линейной структуры

Присваивание переменной какого-либо значения или присваивание одной переменной значения другой переменной является наиболее часто выполняемым действием в программе, независимо от языка программирования.

В языках программирования указанное действие является операцией и обозначается, например, в *C++* как «=», в *Delphi* – «:=». При выполнении происходит не только присваивание значения определенной переменной, но и возвращается некоторое значение.

Пример 2.8. Известны плотность и геометрические размеры цилиндрического слитка, полученного в металлургической лаборатории. Найти объем, массу и площадь основания слитка.

Входные данные:

R – радиус основания цилиндра,

h – высота цилиндра,

ρ – плотность материала слитка.

Выходные данные:

m – масса слитка,

V – объем,

S – площадь основания.

Блок-схема представлена на рис. 2.5.

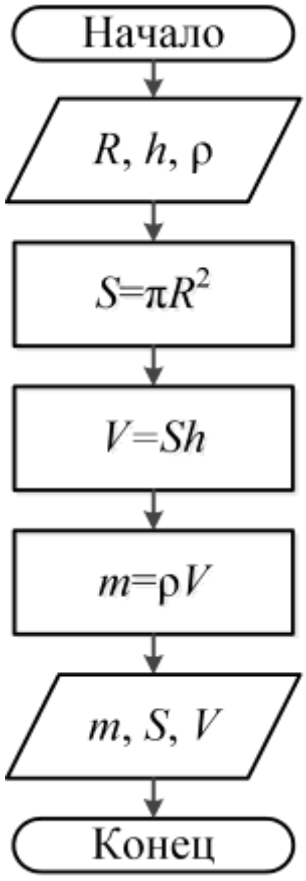


Рис. 2.5. Пример линейного алгоритма

Запись линейного алгоритма в виде псевдокода:

```
алг ОБЪЕММАССАПЛОЩАДЬ
вещ R, h, ρ, m, V, S
нач ввод R, h, ρ
  S=πR^2
  V=Sh
  m=ρV
  вывод «m=», m, «S=», S, «V=», V
кон алг ОБЪЕММАССАПЛОЩАДЬ
```

2.4.2. Разветвляющиеся алгоритмы

Разветвляющаяся структура (*ветвление*) – это структура, обеспечивающая альтернативный выбор в зависимости от заданного условия. Сначала выполняется проверка условия, затем выбирается один из путей в зависимости от истинности (обозначения: Да , 1, +) или ложности (обозначения: Нет , 0, -) условия (рис. 2.6). Каждая из ветвей ведет к общей точке слияния, так что выполнение программы продолжается независимо от того, какой путь был выбран.

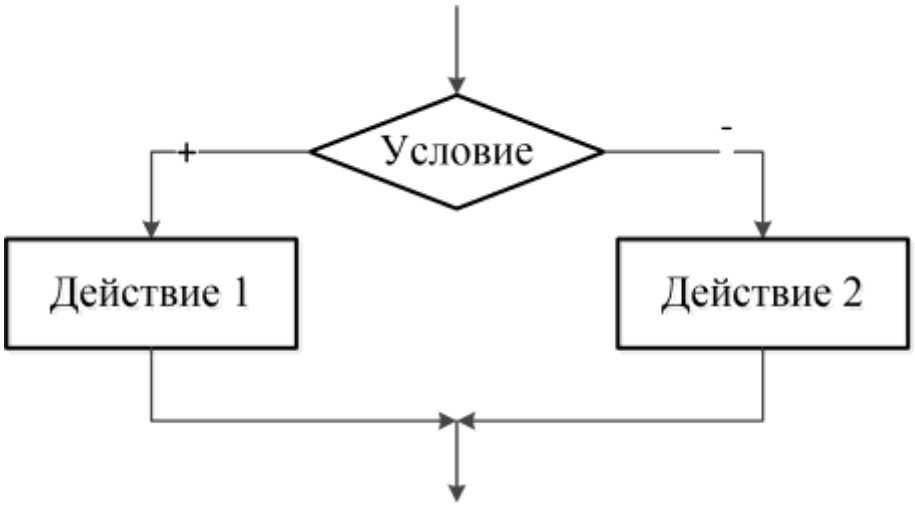


Рис. 2.6. Полная форма условного оператора

Фрагмент псевдокода полной формы условного оператора:

```
если <условие>
  то <действие 1>
  иначе <действие 2>
все если
```

Может оказаться, что для одного из результатов проверки условия ничего предпринимать не надо. В этом случае можно применять только один обрабатывающий блок (рис. 2.7).

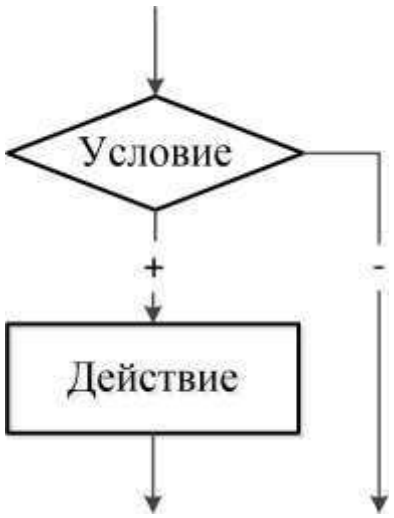


Рис. 2.7. Сокращенная форма условного оператора

Фрагмент псевдокода сокращенной формы условного оператора:

```
если <условие>
  то <действие 1>
все если
```

Пример 2.9. Известны коэффициенты *a*, *b* и *c* квадратного уравнения. Вычислить корни квадратного уравнения.

Решение уравнения зависит от значений коэффициентов *a*, *b*, *c*. Анализ вриантов приводит к следующим результатам (при поиске вещественных корней):

1. Если $a=0$, $b=0$, $c=0$, то любое x – решение уравнения.
2. Если $a=0$, $b=0$, $c\neq 0$, то уравнение действительных решений не имеет.
3. Если $a=0$, $b\neq 0$, то это линейное уравнение, которое имеет одно решение $x=-c/b$.
4. Если $a\neq 0$ и $d=b^2-4ac\geq 0$, то уравнение имеет два вещественных корня.
5. Если $a\neq 0$ и $d<0$, то уравнение не имеет вещественных корней.

Блок-схема представлена на рис. 2.8 – 2.10.

Стоить отметить, что в представленной реализации алгоритма решения квадратного уравнения используются вложенные (каскадные) условия.

При программной реализации большое количество вложений может ухудшить «читаемость» кода. Для оптимизации можно использовать сложные условия, включающие в себя логические операторы «И», «ИЛИ», «НЕ», либо применять операторы множественного выбора, если они предусмотрены в используемом языке.

Кроме того, в языках программирования предусмотрены специальные символы (операторные скобки) или служебные слова для составных операторов, которые используются, в том числе, при большом количестве вложенных условий.

Запись алгоритма решения квадратного уравнения в форме псевдокода:

```
алг КорниКвадратногоУравнения
вещ a, b, c, d, x1, x2
нач ввод a, b, c
  если a=0
    то если b=0
      то если c=0
        то вывод «Любое x - решение»
        иначе вывод «Нет решений»
        все если
      иначе x=–c/b
        вывод x
      все если
    иначе d=b^2–4ac≥0
    если d<0
      то вывод «Нет вещественный корней»
      иначе x1=(–b+√d)/(2a); x2=(–b–√d)/(2a)
        вывод «x1=», x1, «x2=», x2
    все если
  все если
кон алг КорниКвадратногоУравнения
```

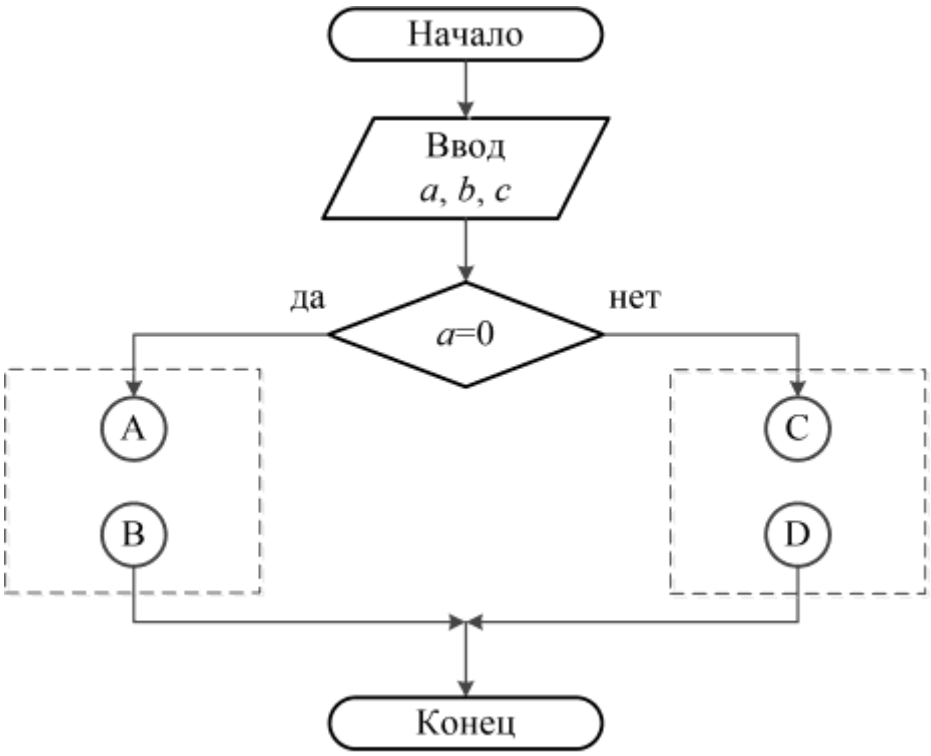


Рис. 2.8. Блок-схема решения квадратного уравнения. Часть 1

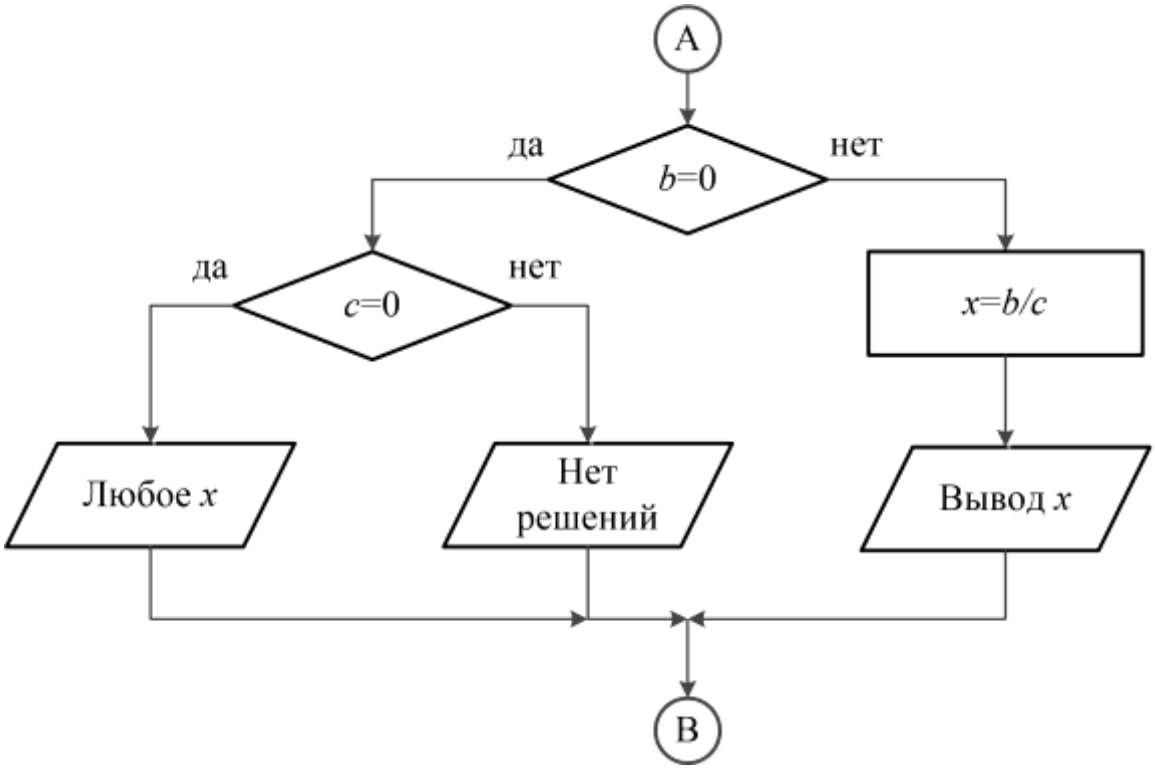


Рис. 2.9. Блок-схема решения квадратного уравнения. Часть 2

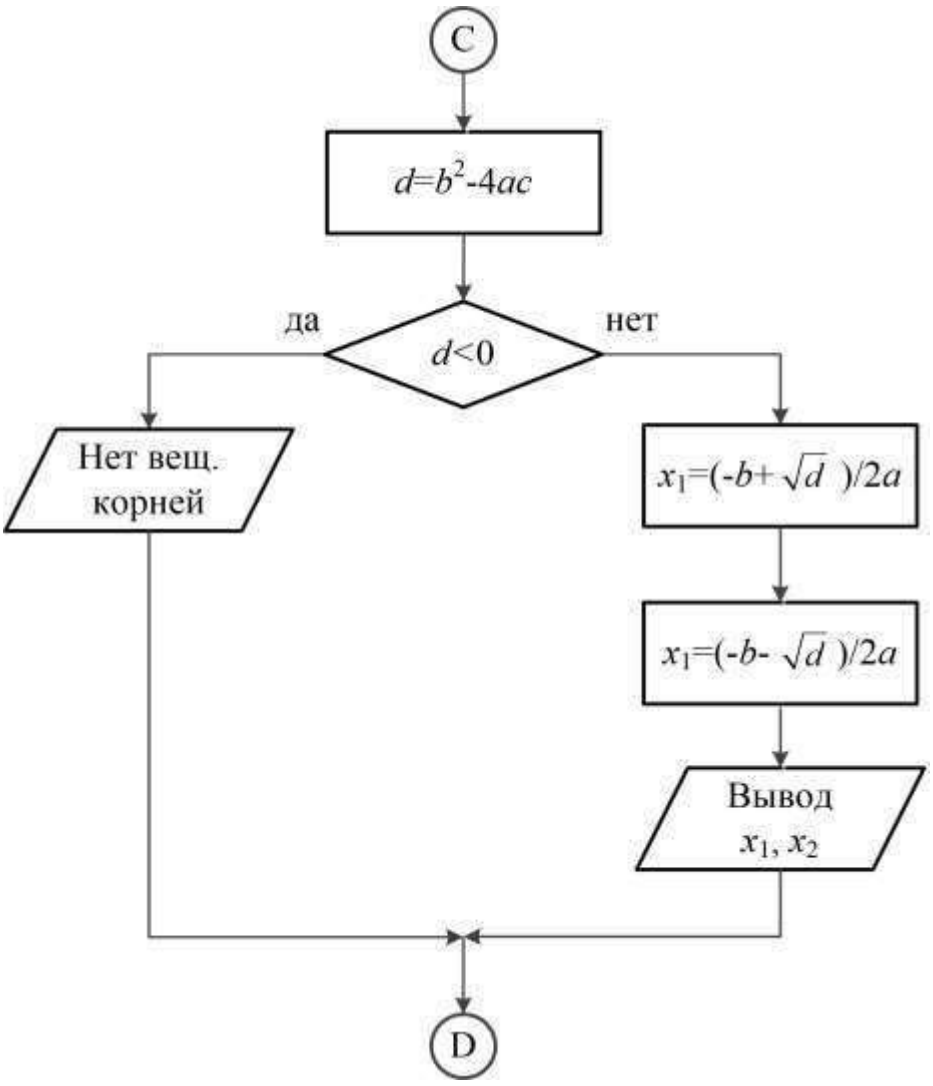


Рис. 2.10. Блок-схема решения квадратного уравнения. Часть 3

2.4.3. Циклические алгоритмы

Циклическая структура (или *повторение*) предусматривает повторное выполнение некоторого набора действий.

Циклы позволяют записать длинные последовательности операций обработки данных с помощью небольшого числа повторяющихся команд.

Итерационным называется цикл, число повторений которого не задается, а определяется в ходе выполнения цикла. В этом случае одно повторение называется итерацией (рис. 2.11). В блок-

схемах для обозначения данной алгоритмической конструкции используется элемент подготовка.

Фрагмент псевдокода итерационного цикла (УП – управляемый параметр, НЗ – начальное значение, КЗ – конечное значение):

```
нц для  УП от НЗ до КЗ, Шаг
    <действие>
кц
```

Рис. 2.11. Итерационный цикл

Различают циклы с предусловием и постусловием (рис. 2.12).

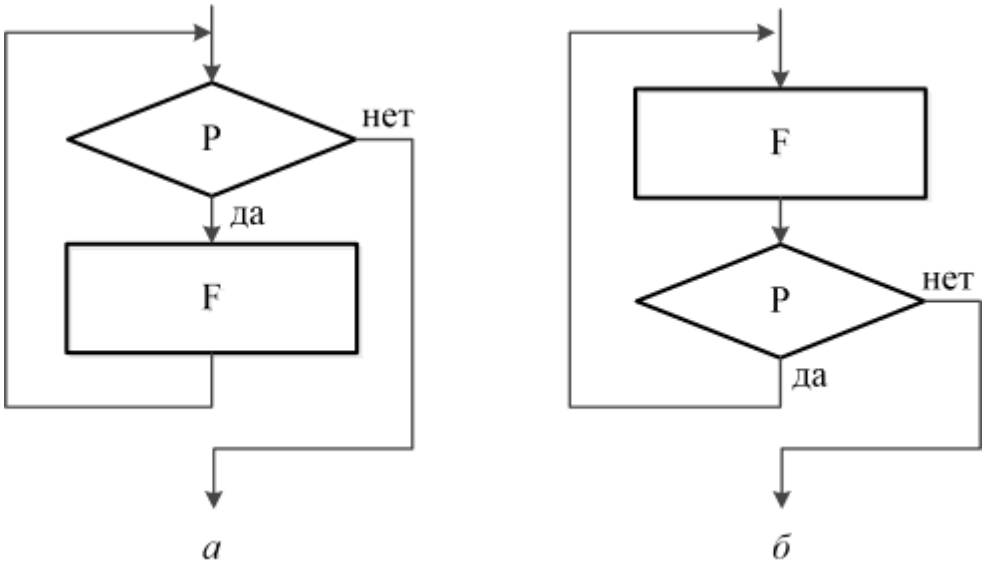


Рис. 2.12 *а* – цикл с предусловием *б* – цикл с постусловием

Фрагмент псевдокода цикла с предусловием:

```
УП=НЗ
нц пока УП <условие>
  <действие>
  <изменение УП>
Кц
```

Данный цикл начинается с проверки логического выражения «*P*». Если оно истинно, то выполняется «*F*», затем все повторяется снова, до тех пор, пока логическое выражение сохраняет значение «истина». Как только оно становится ложным, выполнение операций «*F*» прекращается, и управление передается по программе дальше.

Цикл с постусловием предусматривает проверку после выполнения команд, встроенных внутрь его тела, поэтому он будет выполнен хоты бы один раз.

Фрагмент псевдокода цикла с постусловием:

```
УП=НЗ
нц
  <действие>
  <изменение УП>
кц до УП <условие>
```

2.5. Понятие сложности алгоритма

Вычислительным процессом, порожденным алгоритмом, называется последовательность шагов алгоритма, пройденных при исполнении этого алгоритма.

Сложность алгоритма – количество элементарных действий в вычислительном процессе этого алгоритма, как функцию от исходных данных.

Сложность зависит не только от размерности входных данных, но и от самих данных. Очевидно, что чем сложнее алгоритм в вычислительном плане, тем больше времени и вычислительных ресурсов потребует его выполнение.

Существует несколько способов измерения сложности алгоритма: временная, пространственная и интеллектуальная.

Временная сложность алгоритма представляет собой время *T*, необходимое для его выполнения в зависимости от исходных

данных. Оно равно произведению числа элементарных действий k на среднее время выполнения одного действия t: $T = kt$.

Поскольку t зависит от исполнителя, реализующего алгоритм, то сложность вычислительного процесса, в первую очередь, определяется значением k.

Наряду со скоростью алгоритма важны и другие показатели: требования к объему памяти и свободному месте на диске. Использование быстрого алгоритма не приведет к ожидаемым результатам, если для его работы понадобится больше памяти, чем есть у компьютера.

Пространственная эффективность измеряется количеством памяти, требуемой для выполнения программы.

Все вычислительные машины обладают ограниченным объемом памяти. Если две программы реализуют идентичные функции, то та, которая использует меньший объем памяти, характеризуется большей пространственной эффективностью.

При анализе ***интеллектуальной сложности*** алгоритма исследуется понятность алгоритмов и сложность их разработки.

Например, алгоритм быстрой сортировки характеризуется большей интеллектуальной сложностью по сравнению с алгоритмом сортировки вставками.

Все три формы сложности обычно взаимосвязаны. Как правило, при разработке алгоритма с хорошей временной оценкой сложности приходится жертвовать его пространственной и/или интеллектуальной сложностью. Задачу можно решить быстро, используя большой объем памяти, или медленнее, занимая меньший объем.

Эффективность алгоритма – это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. Алгоритм должен быть проанализирован с целью определения необходимых алгоритму ресурсов. Эффективность алгоритма можно рассматривать как аналог производственной производительности повторяющихся или непрерывных процессов.

2.5.1. Расчет временной сложности

Временную сложность алгоритма обычно связывают с размером входных данных n и определяют как функцию $T(n)$. Например, для алгоритмов обработки массивов – структур данных, хранящих набор значений – в качестве размера n используют длину массива.

Также рассматривают понятие *среднее время работы алгоритма*, то есть математическое ожидание времени работы алгоритма.

Рассмотрим алгоритмы выполнения нескольких операций с массивом A длины n, который может быть объявлен в программе на алгоритмическом языке как: **цел *A*[1:*n*]**.

Пример 2.10. Вычислить сумму первых трех элементов массива.

Решение этой задачи содержит всего один оператор

```
S = A[1] + A[2] + A[3].
```

Этот алгоритм включает две операции сложения и одну операцию записи значения в память, поэтому его временная сложность $T(n) = 3$ не зависит от размера массива вообще.

Пример 2.11. Вычислить сумму всех элементов массива.

Для решения задачи необходима циклическая конструкция в алгоритме:

```
S = A[1]
нц для i от 1 до n, шаг=1
  S = S + A[i]
кц
```

Здесь выполняется $n - 1$ операций сложения и n операций записи в память, поэтому его суммарная сложность $T(n) = 2n - 1$ возрастает линейно с увеличением длины массива.

Пример 2.12. Рассмотрим фрагмент псевдокода, который для матрицы $A[n \times n]$ находит максимальный элемент в каждой строке.

```
max=A[1,1]
нц для i от 1 до n, шаг 1
  нц для j от 1 до n, шаг 1
    если A[i,j]>max
```

```
        то max = A[i,j]
      все если
    кц
  кц
```

В этом алгоритме переменная i меняется от 1 до n. При каждом изменении i, переменная j тоже меняется от 1 до n. Во время каждой из n итераций внешнего цикла, внутренний цикл тоже выполняется n раз. Общее количество итераций внутреннего цикла равно $n*n$. Это определяет временную сложность алгоритма $T(n) = n^2$.

2.5.2. Асимптотическая сложность

В некоторых задачах функция временной сложности алгоритма может быть определена точно, но в большинстве случаев искать точное ее значение бессмысленно.

Точное значение временной сложности зависит, во-первых, от способа определения элементарных операций, во-вторых, при увеличении размера входных данных вклад постоянных множителей и слагаемых низших порядков, фигурирующих в выражении для точного времени работы, становится крайне незначительным.

Рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма приводят к понятию асимптотической сложности алгоритма.

Алгоритм имеет ***асимптотическую сложность*** $O(f(n))$, если найдется такая постоянная c, для которой, начиная с некоторого n, выполняется условие $T(n) \le c \cdot f(n)$.

Это значит, что график функции $c \cdot f(n)$ идет выше, чем график функции $T(n)$, по крайней мере, при $n \ge n_0$ (рис. 2.13).

Алгоритм имеет ***сложность O(f(n)) (порядок сложности f(n))***, если при увеличении размерности входных данных n, время выполнения алгоритма возрастает с той же скоростью, что и функция $f(n)$.

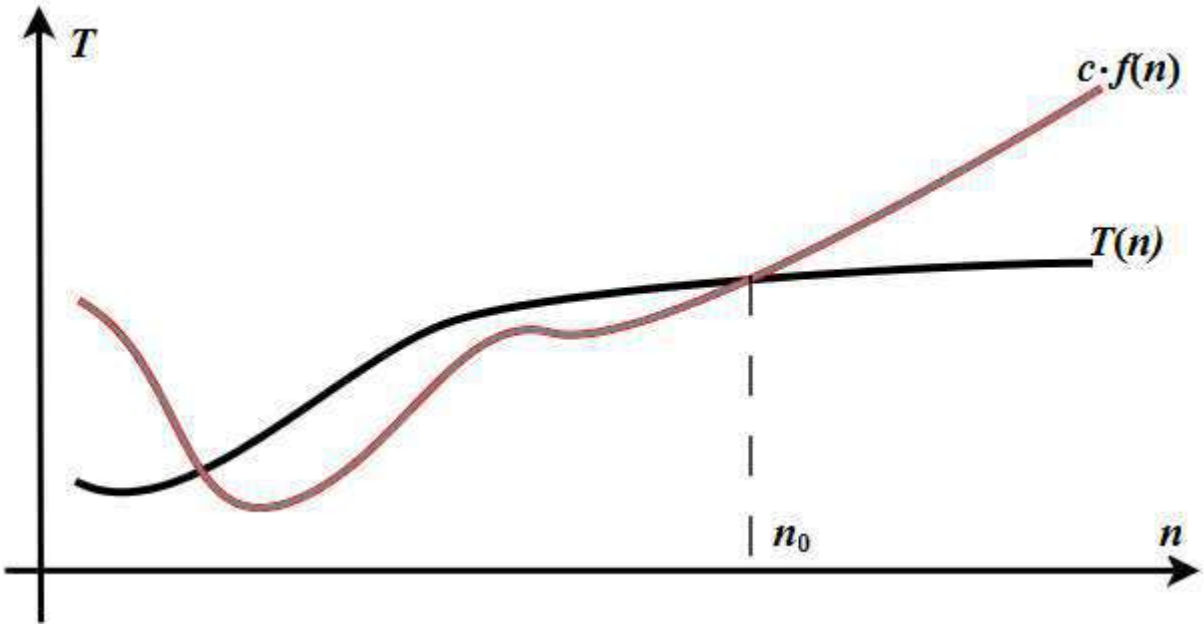


Рис. 2.13. Асимптотическая сложность

Чаще всего для вычисления сложности используются следующие функции $f(n)$:

- c – константа;
- $\log(\log(n))$;
- $\log(n)$;
- n^c, $0<c<1$;
- n;
- $n*\log(n)$
- n^c, $c>1$;
- c^n, $c>1$;
- $n!$.

Функции перечислены в порядке возрастания сложности. Чем выше в этом списке находится функция, тем быстрее будет выполняться алгоритм с такой оценкой.

Один из способов рассмотрения относительных размеров этих функций заключается в определении времени, которое требуется для решения задач различных размеров.

В табл. 2.1 приведено время выполнения алгоритма с определенной сложностью в зависимости от размера входных данных при скорости 10^6 операций в секунду.

Таблица 2.1 Время выполнения алгоритмов

Сложность	Размер входных данных					
	10	20	30	40	50	60
n	0,00001 с	0,00002 с	0,00003 с	0,00004 с	0,00005 с	0,00006 с
n^2	0,00010 с	0,00040 с	0,00090 с	0,00160 с	0,00250 с	0,00360 с
n^3	0,00100 с	0,00800 с	0,02700 с	0,06400 с	0,12500 с	0,21600 с
n^5	0,10000 с	3,20000 с	24,3000 с	1,70000 мин	5,20000 мин	13,0 мин
2^n	0,00010 с	1,00000 с	17,9000 мин	12,7 дней	35,7 веков	366 веков
3^n	0,05900 с	58,0 мин	6,5 лет	3855 веков	2×10^8 веков	$1{,}3\times10^{13}$ веков

2.5.3. Классы сложности алгоритмов

В теории алгоритмов классами сложности называются множества вычислительных задач, примерно одинаковых по сложности вычисления. Один класс может содержать все задачи другого, плюс задачи, требующие таких дополнительных вычислительных ресурсов, как время или память.

Существует множество различных классов сложности (рис. 2.14), хотя в большинстве случаев исследователи не смогли доказать, что один класс четко отличается от других. Доказательства различия между классами – одни из самых трудных и важных задач в этой области.

Разница между классами может быть едва различимой или явной, и хорошо разбираться во всех классах трудно.

Класс P вмещает все те проблемы, решение которых считается «быстрым» и полиноминально зависит от размера входных данных.

Класс NP содержит задачи, которые могут быть решены за недетерминированное полиномиальное время.

Класс BPP совпадает с *P*, с той разницей, что алгоритм может включать шаги, в которых принятие решений происходит

случайно. Алгоритмам в *BPP* необходимо давать правильные ответы с вероятностью, близкой к 1.

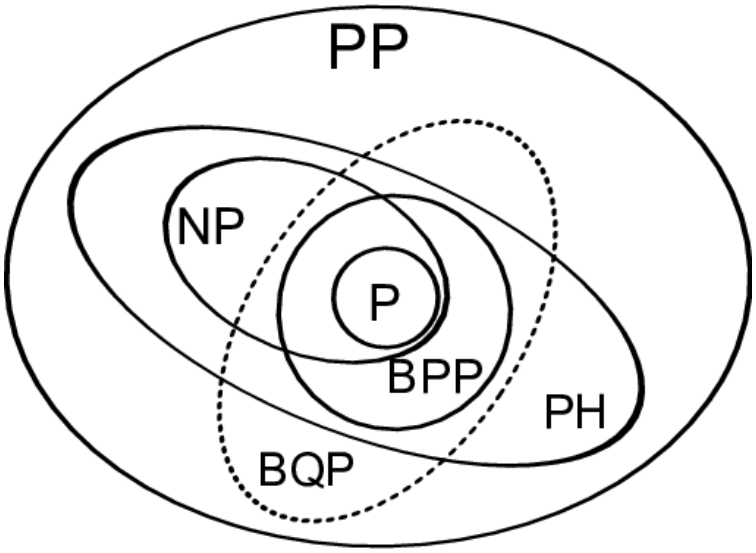


Рис. 2.14. Иерархия классов сложности алгоритмов

Класс PH является объединением всех классов сложности в полиномиальной иерархии.

Класс BQP включает в себя задачи, которые можно решить на квантовом компьютере за полиномиальное время с ограничением вероятности ошибок.

Класс PP (PSPACE) содержит все задачи, которые можно решить, используя разумное количество памяти.

Наряду с полиномиальной сложностью выделяют и экспоненциальную сложность.

Так, ***класс EXPTIME*** включает все задачи, которые можно решить за экспоненциальное время на классическом компьютере. Алгоритмы с экспоненциальной сложностью принято считать «медленными». Однако в некоторых случаях для малых значений *n* полиномиальное время может превосходить экспоненциальное.

2.6. Задачи для самоконтроля

1. Описать Машину Тьюринга для выполнения следующих действий:
   1.1. Алфавит $A=\{a, b, c\}$. Приписать слева к слову P символ b ($P \rightarrow bP$).
   1.2. Алфавит $A=\{a, b, c\}$. Приписать справа к слову P символы bc ($P \rightarrow Pbc$).
   1.3. Алфавит $A=\{a, b, c\}$. Заменить на a каждый второй символ в слове P.
   1.4. Алфавит $A=\{a, b, c\}$. Оставить в слове P только первый символ (пустое слово не менять).
   1.5. Алфавит $A=\{a, b, c\}$. Оставить в слове P только последний символ (пустое слово не менять).
   1.6. Алфавит $A=\{a, b, c\}$. Пусть P имеет нечетную длину. Оставить в P только средний символ.
   1.7. Алфавит $A=\{a, b, c\}$. Если слово P имеет четную длину, то оставить в нем только левую половину.
   1.8. Алфавит $A=\{a, b, c\}$. Приписать слева к непустому слову P его первый символ.
2. Описать систему подстановок нормального алгоритма Маркова для реализации следующих действий:
   2.1. Алфавит $A=\{f, h, p\}$. В слове P заменить все пары ph на f.
   2.2. Алфавит $A=\{f, h, p\}$. В слове P заменить на f только первую пару ph, если такая есть.
   2.3. Алфавит $A=\{a, b, c\}$. Приписать слово bac слева к слову P.
   2.4. Алфавит $A=\{a, b, c\}$. Заменить слово P на пустое слово, т.е. удалить из P все символы.
   2.5. Алфавит $A=\{a, b, c\}$. Заменить любое входное слово на слово a.
   2.6. Построить НАМ, не меняющий входное слово (при любом алфавите A).
3. Разработайте алгоритм для вычисления значения выражения в формах блок-схемы и псевдокода. Реализуйте проверку корректности входных данных.

3.1. $s = \dfrac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y| \cdot \left(\sin^2(z) + \mathrm{tg}(z)\right)}$.

3.2. $s = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x + \dfrac{y}{2}}{2 \cdot |x+y|} \cdot (x+1)^{-1/\sin(z)}$.

3.3. $s = \dfrac{x^{1+y} + e^{y-1}}{1 + x \cdot |y - \mathrm{tg}(z)|} \cdot (1 + |y-x|) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$.

3.4. $s = \dfrac{\sqrt[3]{8 + (x+y)^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|} \cdot \cos^2\left(\dfrac{\pi}{4}\right)$.

3.5. $s = \dfrac{\sqrt[4]{y + x^2}}{|x-y| \cdot \left(1 + \sin^2(x+y)\right)}$.

4. Написать псевдокод алгоритма вычисления значения выражения и определить его временную сложность при $5 \le n \le 10$, если не указано иное.

4.1. $Q = \cos x + \cos 2x + ... + \cos nx$.

4.2. $L = \dfrac{1}{\sin 1} + \dfrac{1}{\sin 1 + \sin 2} + ... + \dfrac{1}{\sin 1 + ... + \sin n}$.

4.3. $V = \prod_{m=1}^{m=5} (m+k)!$.

2.7. Вопросы для самостоятельного изучения

1. Вычислимость функции по Тьюрингу.
2. Многоленточные машины Тьюринга.
3. Устройство машины Поста. Принцип и примеры работы.
4. Операции над нормальными алгоритмами: композиция, разветвление, повторение. Нормально вычислимые функции и принцип нормализации Маркова.
5. Понятие рекурсии. Восходящая и нисходящая рекурсии. Алгоритмическая конструкция.
6. Массовые проблемы. Экстраалгоритм и неразрешимые проблемы.
7. Класс вычислительной сложности алгоритмов *EXPSPASE*.

3. ЯЗЫКИ И СРЕДЫ ПРОГРАММИРОВАНИЯ

3.1. Задачи, языки и программы

Программирование – процесс превращения алгоритма в код, написанный на языке программирования, который может быть выполнен компьютером.

Программирование является собирательным понятием и может рассматриваться и как наука и как искусство, что обуславливает научно-практический подход к разработке программ.

Процесс решения задачи с помощью компьютера представляет собой совместную деятельность человека и ЭВМ.

Действия, выполняемые человеком:

1. Постановка задачи.
2. Анализ и исследование задачи, модели.
3. Разработка алгоритма.
4. Программирование.
5. Тестирование и отладка.
6. Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5.
7. Сопровождение программы.

Деятельность компьютера связана с этапами обработки информации в соответствии с разработанным алгоритмом.

Взаимодействие сторон в виде укрупненной схемы представлено на рис. 3.1.

Программа – результат интеллектуального труда. В любом коде присутствует индивидуальность ее разработчика. Вместе с тем программирование предполагает и рутинные работы, которые могут и должны иметь строгий регламент выполнения и соответствовать стандартам.

Язык программирования – это формальная знаковая система, содержащая набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит компьютер под ее управлением, является «посредником» между человеком и компьютером.

Рис. 3.1. Основные этапы решения задачи с помощью компьютера

Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Языки программирования всегда представляют собой некоторые модели виртуальных вычислительных машин, позволяющие наиболее эффективно использовать возможности вычислительных средств, существенные для конкретных областей применения.

Фактически, при написании программ ориентируются не на вычислительную машину как техническое устройство («железо»), а на некоторую абстрактную модель вычислительного устройства. Качество абстрактной модели, ее соответствие сути решаемых задач во многом определяет качество и эффективность проектирования с использованием этой модели.

Сложность задач, которые возможно решить с применением тех или иных концепций проектирования и выражающих их языков, непосредственно связана с ***уровнем абстракции*** как при постановке задачи, так и в ходе ее решения.

На нижнем уровне абстракции находятся языки ассемблера (*assembly language*), позволяющие избежать необходимости программирования в машинном коде («родном» языке компьютера).

Ассемблеры являются абстракцией компьютера (точнее говоря, компьютерной архитектуры), для которого были спроектированы. Это – машинно-ориентированные языки низкого уровня с командами, обычно соответствующими командам компьютера.

Пример 3.1. Листинг программы «Здравствуй, Мир!» на языке Ассемблер.

```
    use16               ; генерация 16-битного кода
    org 100h            ; начало адресации команд и данных
    mov dx,hello        ; hello помещается в регистр dx
    mov ah,9            ; функция вывода строки
    int 21h             ; получение следующей клавиши
    ; стандартное завершение процесса
    mov ax,4C00h
    int 21h
    ; в hello хранится приветствие, выводимое на экран
    hello db 'Hello, World!$'
```

Человеку свойственно формулировать и решать задачи в выражениях более общего характера, чем команды ЭВМ. Поэтому с развитием программирования появились языки, ориентированные на более высокий уровень абстракции при описании решаемой задачи. Эти языки получили название языков высокого уровня.

Пример 3.2. Листинг программы «Здравствуй, Мир!» на языке С.

```
    // подключение библиотеки ввода-вывода
    #include <stdio.h>

    // главная функция программы
    int main()
    {
      // вывод сообщения на экран
      printf("Hello, World!\n");
      // завершение функции
      return 0;
    }
```

Пример 3.3. Листинг программы «Здравствуй, Мир!» на языке *C#*.

```
// подключение пространства имен System
using System;

// класс точки входа для приложения
class Program
{
  // основной метод
  public static void Main(string[] args)
  {
   // вывод сообщения на экран
   Console.WriteLine("Hello, world!");
  }
}
```

На настоящий момент выделяют пять поколений языков программирования.

В ***первое поколение*** входят языки, созданные в начале 50-х годов, когда появились первые компьютеры. Первая реализация языка Assembler была создана по принципу – «одна инструкция – одна строка».

Второе поколение (60-е): разработан символический *Assembler*, в котором появилось понятие переменной. Возрастает скорость разработки и надежность программ, появляются Fortran и COBOL.

Третье поколение (70-е): рождаются универсальные языки высокого уровня, которые повышают производительность труда программистов. На этих языках пишутся инженерные и экономические задачи (*Basic*, *Algol*).

Языки ***четвертого поколения*** носят ярко выраженный непроцедурный характер, определяемый тем, что программы описывают только что, а не как надо, сделать. Типичными примерами непроцедурных языков являются языки, используемые для задач искусственного интеллекта (например, *Prolog*, *Langin*). Так как непроцедурные языки имеют минимальное число синтаксических правил, они значительно более пригодны для применения непрофессионалами в области программирования.

Второй тенденцией развития четвертого поколения являются объектно-ориентированные языки, базирующиеся на понятии программного объекта, который состоит из структур данных и алгоритмов, при этом каждый объект знает, как выполнять операции со своими собственными данными. Такими свойствами обладают объектно-ориентированные *C++, SmallTalk, Simula, Actor* и ряд других языков программирования.

Третьим направлением развития языков четвертого поколения можно считать языки запросов, позволяющих пользователю получать информацию из баз данных. Среди языков запросов фактическим стандартом стал язык *SQL* (*Structured Query Language*).

И, наконец, четвертым направлением развития являются языки параллельного программирования (модификация ЯВУ *Fortran,* языки *Occam, SISAL, FP* и др.), которые ориентированы на создание программного обеспечения для вычислительных средств параллельной архитектуры (многомашинные, мультипроцессорные среды и др.).

К языкам ***пятого поколения*** (середина 90-х) относятся системы автоматического создания прикладных программ с помощью визуальных средств разработки, без знания программирования (*InterLisp, ExpertLisp, IQLisp, SAIL, Clout, QA, HAL*).

Каждый язык программирования может быть представлен в виде набора формальных спецификаций, определяющих его синтаксис и семантику.

Эти спецификации обычно включают в себя описание:

1. Типов и структур данных.
2. Операционную семантику (алгоритм вычисления конструкций языка).
3. Семантические конструкции языка.
4. Библиотеки примитивов (например, инструкции ввода-вывода).
5. Философии, назначения и возможностей языка.

Выбор типа языка зависит от многих факторов: назначения, удобства написания исходных программ, эффективности получаемых объектных программ.

3.2. Трансляция

Программа, написанная на языке высокого уровня, перед исполнением должна быть преобразована в программу на языке команд компьютера («машинном языке»). Такой процесс называется трансляцией. По типу выходных данных различают два основных вида трансляторов:

- компилирующие окончательный выполнимый код;
- компилирующие интерпретируемый код, для выполнения которого требуется дополнительное программное обеспечение.

Окончательным выполнимым кодом являются приложения, реализованные как *exe*-файлы, *dll*-библиотеки, *com*-компоненты.

К интерпретируемому коду можно отнести байт-код *Java*-программ, выполняемый посредством виртуальной машины *JVM* (*Java Virtual Machine*).

Языки, предполагающие формирование компилятором окончательного выполнимого кода, называются компилируемыми языками. К ним относятся языки *C*, *C++*, *FORTRAN*, *Pascal*.

Языки, предполагающие формирование компилятором интерпретируемого кода, называются интерпретируемыми языками. К таким языкам относятся языки *Java*, *C#*, *LISP*, *Perl*, *Prolog*.

В большинстве случаев код, получаемый в результате процесса трансляции, формируется из нескольких программных модулей (программный модуль – это определенным образом оформленный код на ЯВУ). Процесс трансляции в этом случае может выполняться как единое целое – компиляция и редактирование связей, или как два отдельных этапа – сначала компиляция в объектные модули, а затем вызов редактора связей, создающего окончательный код. Последний подход более удобен для разработки программ. Он реализован в трансляторах языков *C* и *C++*.

Объектный код, создаваемый компилятором, представляет собой область данных и область машинных команд, имеющих адреса, которые в дальнейшем «согласуются» редактором связей (его еще называют компоновщиком или линковщиком). Редактор связи размещает в едином адресном пространстве все по отдель-

ности откомпилированные объектные модули и статически подключаемые библиотеки.

3.3. Основные парадигмы программирования

Парадигма – совокупность фундаментальных научных установок, представлений и терминов, принимаемая и разделяемая научным сообществом и объединяющая большинство его членов. Обеспечивает преемственность развития науки и научного творчества.

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

В соответствии с концепцией фон-Неймана – основателя теоретической концепции компьютерной техники, процессор обрабатывает данные, выполняя инструкции (команды), которые находятся в той же оперативной памяти, что и данные.

Таким образом, можно выделить две основные сущности процесса обработки информации: код, как совокупность инструкций, и данные. Все программы в соответствии с выбранной технологией программирования концептуально организованы вокруг своего кода или вокруг своих данных.

Основные (однако, есть и более узкоспециализированная классификация) на сегодняшний день парадигмы программирования представлены в виде схемы на рис. 3.2. Для иллюстрации некоторых парадигм, в прил. 3 будут рассмотрены примеры программ с комментариями к коду.

Императивная парадигма программирования описывает процесс вычисления в виде инструкций, изменяющих состояние программы.

Особенности:

- в исходном коде программы записываются инструкции (команды);
- инструкции должны выполняться последовательно;

– при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
– данные, полученные при выполнении инструкции, могут записываться в память.

Рис. 3.2. Основные парадигмы программирования

Процедурное программирование – программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения про-

граммиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Структурное программирование – парадигма, в основе которой лежит представление программы в виде иерархической структуры блоков. Структурное программирование в основном придерживается тех же принципов, что и процедурное, лишь немного дополняя их полезными приемами.

Данная методология появилась как следствие возрастания сложности решаемых на компьютерах задач, и соответственно, усложнения программного обеспечения.

В соответствии с рассматриваемой парадигмой любая ее программа строится *без использования оператора «goto»*, состоит из трех базовых управляющих конструкций: последовательность, ветвление, цикл и содержит подпрограммы (процедуры или функции). Разработка ведется пошагово, методом «сверху вниз».

Процедурное программирование применяется не только в классических языках *C* и *Pascal*, но и в современных, например, в *Go*.

Объектно-ориентированная парадигма, при которой программа рассматривается как совокупность фрагментов кода, обрабатывающих отдельные совокупности данных – объекты. Эти объекты взаимодействуют друг с другом посредством так называемых интерфейсов. Данные управляют доступом к коду. Объектно-ориентированные языки программирования: *C++*, *C#*, *Java*, *Object Pascal* и др.

Изучение ООП целесообразно начать на примере объектно-ориентированного языка программирования *C++*, как наиболее теоретически выдержанного в этой части. Другие языки, поддерживающие идеи ООП, такие, как *Object Pascal*, *Java*, *C#* разрабатывались, в первую очередь, с учетом удобства программирования задач в соответствующих предметных областях.

Декларативная парадигма программирования задает спецификацию решения задачи, то есть описывает, что представляет собой проблема и ожидаемый результат. В общем и целом, декларативное программирование идет от человека к машине, тогда как императивное – от машины к человеку. Как следствие, декла-

ративные программы не используют понятия состояния, то есть не содержат переменных и операторов присваивания.

При декларативном программировании в программе описывается результат, который нужно получить, а детали того, как выполнить его, оставлены интерпретатору языка.

В **функциональной парадигме** (языки *Lisp*, *Clojure*, *Haskell*, *Scala*, *R*) основная идея заключается в формализованном определении функции, которую выполняет программа. Таким образом, вместо определения последовательности состояний, через которые должен пройти компьютер, чтобы получить требуемый результат, необходимо определить функцию, при применении которой к исходным данным получается требуемое решение.

Многие императивные языки поддерживают возможности функционального программирования.

Парадигма логического программирования (языки *Prolog*, *Erlang*, *Mercury*, *Fril*, *Oz*) основана на использовании системы правил. Операторы программы выполняются не в той последовательности, в которой они написаны, а на основе анализа разрешающих условий.

В рамках изучения дисциплины «Информатика» предусмотрено выполнение лабораторных работ по изучению основ программирования на языках *C* и *C++*.

3.4. Обзор сред разработки для языков *C/C++*

Способность понимать и писать код – ключевые моменты, которые превращают обучающегося в программиста или разработчика. Важным критерием в процессе разработки приложения является выбор интегрированной среды разработки (*IDE*), зависящий как от платформы, так и от предпочтений и уровня подготовки студента.

IDE оказывает большую помощь программисту, предоставляя все необходимые для работы удобства. Также она повышает производительность труда разработчика благодаря полезным инструментам, функциям автозаполнения и сотням сочетаний клавиш.

3.4.1. Microsoft Visual Studio

Visual Studio – среда разработки от компании *Microsoft*, представляет собой полный набор инструментов, позволяющий произвести точную отладку и настройку приложения. В основном используется для написания приложений, включающих в себя *.NET*. Есть как *Community*-версия, так и *PRO*.

Visual Studio предлагает множество функций, некоторые из которых:

- интеллектуальное автодополнение кода;
- дизайнер графических форм (*GUI*);
- простая в использовании навигационная система.

IDE может быть использована для разработки компьютерных программ для *Microsoft Windows*, а также веб-сайтов, веб-приложений и веб-сервисов.

Недостатки: новичку будет просто невозможно самостоятельно разобраться с *Visual Studio* без прохождения специальных курсов и чтения литературы. Этот продукт предназначен для более опытных разработчиков, обращающих внимание на качество редактора и функции тестирования.

Поддерживаемые языки: Basic.NET, C++, C#, F#.

3.4.2. Xcode

Xcode – интегрированная среда разработки (*IDE*) программного обеспечения для платформ *macOS*, *iOS*, *watchOS* и *tvOS*, разработанная корпорацией *Apple*.

Достоинства:

- создание приложений под все *Apple*-платформы;
- автодополнение кода;
- наличие встроенных инструментов машинного обучения;
- удобная работа с *GUI*.

Поддерживаемые языки: C, C++, Objective-C, Objective-C++, Swift, Java, AppleScript, Python и *Ruby*. Сторонними разработчиками реализована поддержка *GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell* и *D*.

3.4.3. Eclipse

Eclipse – простая в использовании и при этом мощная *IDE* для *C* и *C++*. Изначально она главным образом использовалась для разработки на *Java*, но сейчас поддерживает большее разнообразие языков. Эта *IDE* поставляется с отличным графическим пользовательским интерфейсом и функционалом *drag-and-drop*.

Достоинства Eclipse:

- простота установки и использования;
- бесплатный и открытый источник;
- поддержка нескольких платформ;
- простота создания *GUI*-интерфейсов;
- наличие встроенного юнит-тестирования, оптимизации тестов;
- удаленный системный проводник.

Eclipse доступна для *Windows*, *Linux* и *MacOS*.

Поддерживаемые языки: C, *C++*, *C#*, *Java*, *JavaScript*, *Perl*, *PHP*, *Python*, *COBOL* и т. д.

3.4.4. Qt Creator

Qt Creator – самая известная среда разработки для создания графических приложений, является платной, однако имеется и свободно-распространяемая версия

Достоинства Qt Creator:

- простой и удобный конструктор *GUI*-форм;
- кроссплатформенность;
- поддержка отладки, компиляции, профилирования, автозаполнения кода и рефакторинга;
- поддержка анализа статического кода;
- быстрый компилятор *Qt*;
- визуализация данных *Qt*.

Поддерживаемые языки: Основной упор сделан на языки *C*/*C++*. Есть также «привязки» ко многим другим языкам программирования: *Python – PyQt*, *PySide*; *Ruby – QtRuby*; *Java – Qt Jambi*; *PHP – PHP-Qt* и другие.

3.4.5. Code::Blocks

Code::blocks – свободная и открытая среда *IDE* для *C* и *C++*, написанная с использованием *GNU C++*. Имея открытую архитектуру, может масштабироваться за счет подключаемых модулей.

Главный плюс – кроссплатформенность программы (*Windows*, *Linux* и *Mac OS X*).

Некоторые из функций *Code::Blocks*:

- простая и быстрая установка;
- наличие портативной версии;
- удобный конструктор *GUI*-форм;
- встроенная возможность создания блок-схем.

Недостатки: нестабильная работа функции автодополнение.

Поддерживаемые языки: C, *C++*, *D* (с ограничениями) и *Fortran*.

3.4.6. Dev-C++

Dev-C++ – бесплатная интегрированная среда разработки с открытым исходным кодом, написанная в *Delphi* для *Windows*. Это легкая *IDE*, которой требуется всего на пару минут для установки. В ней можно установить плагин для создания *GUI*-интерфейсов методом перетаскивания элементов.

Достоинства оболочки Dev-C++:

- малый вес;
- простая в использовании панель инструментов;
- простая установка.
- графический интерфейс;
- русскоязычное меню;
- встроенный отладчик *GDB*;
- возможность создавать консольные и графические программы.

Недостатки: проект не поддерживается.

Поддерживаемые языки: C и *C++*.

3.5. Краткое практическое руководство

Как уже отмечалось, выбор *IDE* зависит от уровня знаний и практической подготовки. Для новичков лучшими средами разработки являются *Dev-C++* и *Code::Blocks*. При наличии базовых навыков стоит обратить внимание на среды Qt Creator и *Visual Studio*.

Кроме того, не все *IDE* являются кроссплатформенными.

3.5.1. Создание консольного приложения в среде Dev-C++

Для создания приложения необходимо выполнить следующую последовательность шагов:

1. Создать папку для хранения будущего приложения.
2. Запустить среду *Dev-C++* из списка установленных приложений.
3. Создать новый проект (рис. 3.3): Файл→Создать→Проект.

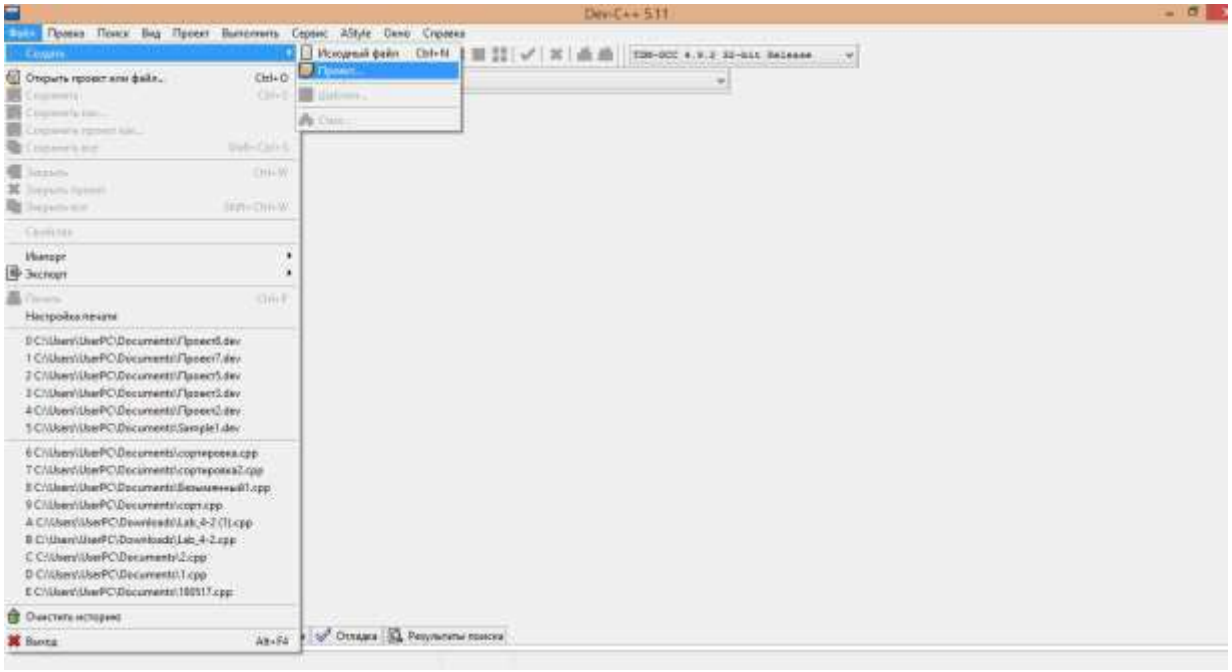


Рис. 3.3. Начало создания проекта в среде *Dev-C++*

4. Задать начальные установки и имя проекта (рис. 3.4).

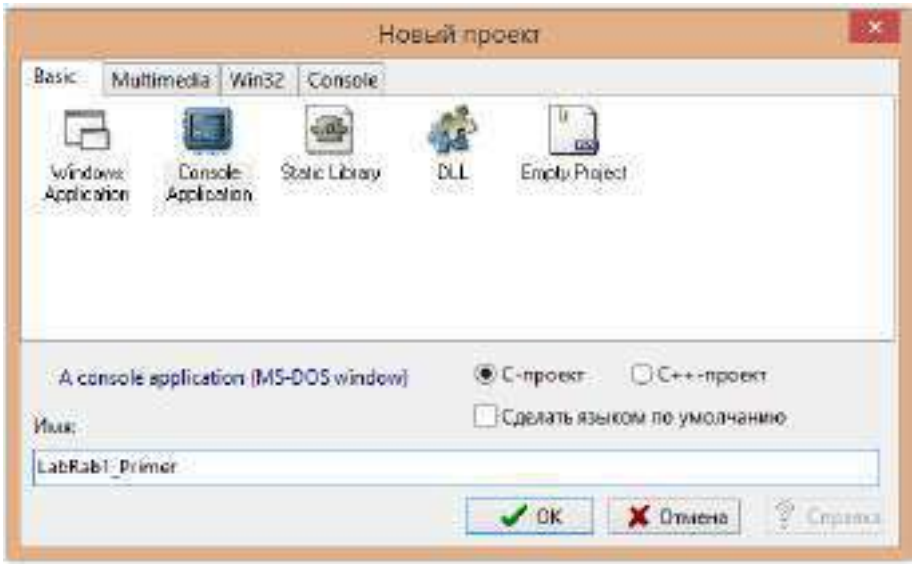


Рис. 3.4. Начальные установки проекта

5. Сохранить проект в заранее созданную папку (рис. 3.5).

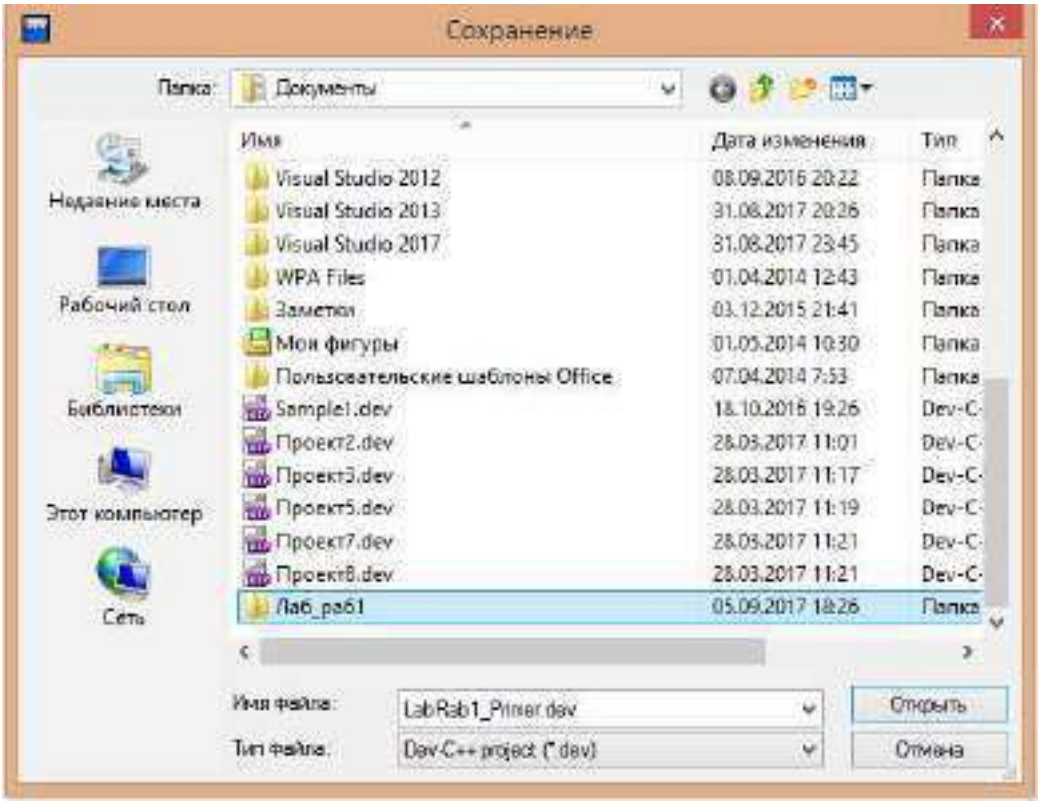


Рис. 3.5. Сохранение проекта

6. Набрать код в открывшемся редакторе.
7. Выполнить компиляцию проекта (рис. 3.6).

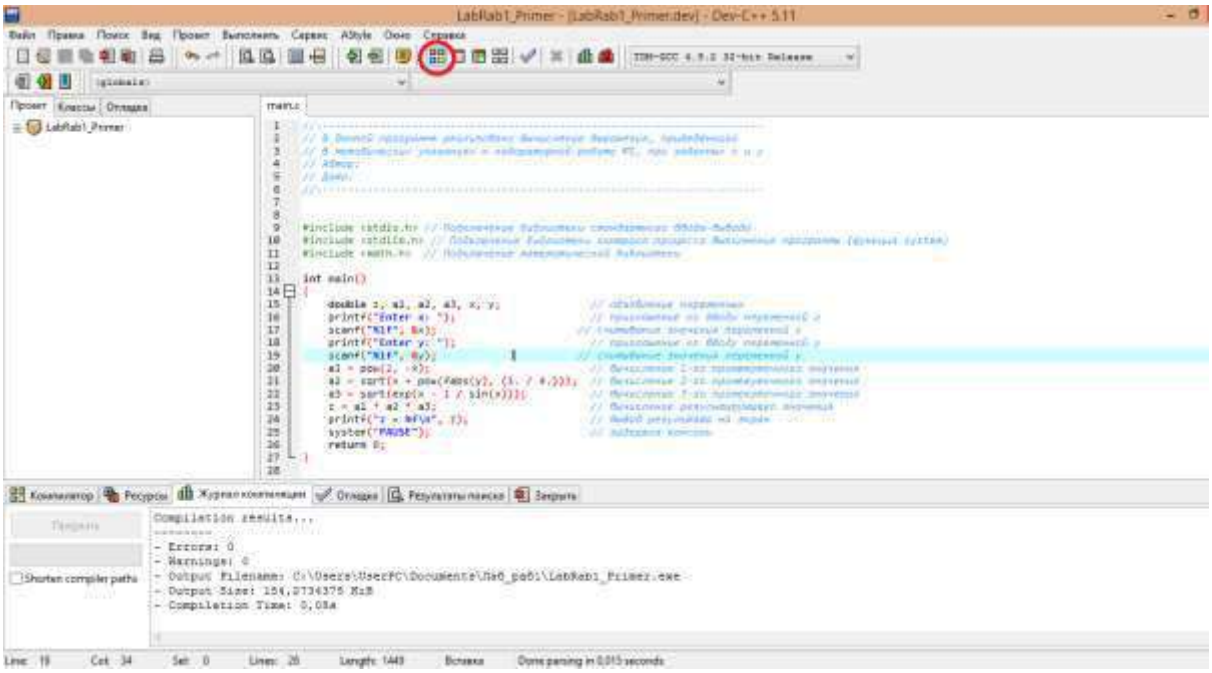


Рис. 3.6. Компиляция проекта

8. Тестирование работы приложения (рис. 3.7).

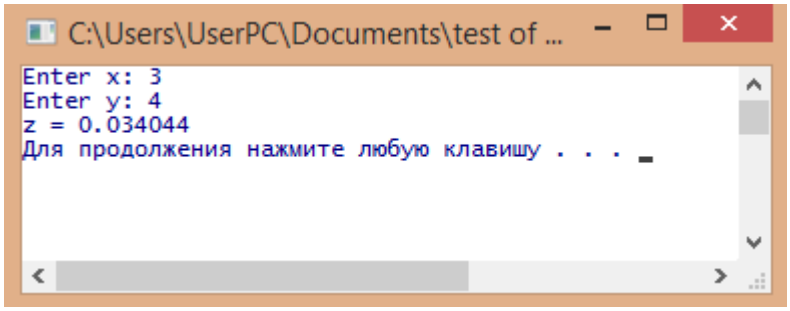


Рис. 3.7. Тестирование приложения

3.5.2. Создание консольного приложения в среде Code::Blocks

Для создания приложения необходимо выполнить следующую последовательность шагов:

1. Создать папку для хранения будущего приложения.
2. Запустить среду *Code::Blocks* из списка установленных приложений.
3. Создать новый проект (рис. 3.8): *File→New→Project*.

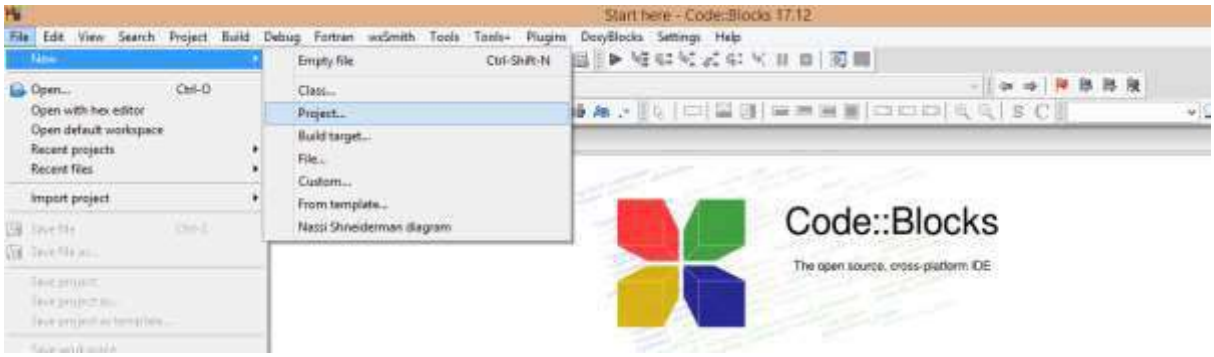


Рис. 3.8. Начало создания проекта в среде *Code::Blocks*

4. Задать тип проекта (рис. 3.9), язык разработки (рис. 3.10), указать название проекта и директорию размещения его файлов (рис. 3.11) и определить настройки компилятора (используются значения по умолчанию) (рис. 3.12). Указанные действия выполняются последовательно, в открывающихся окнах.

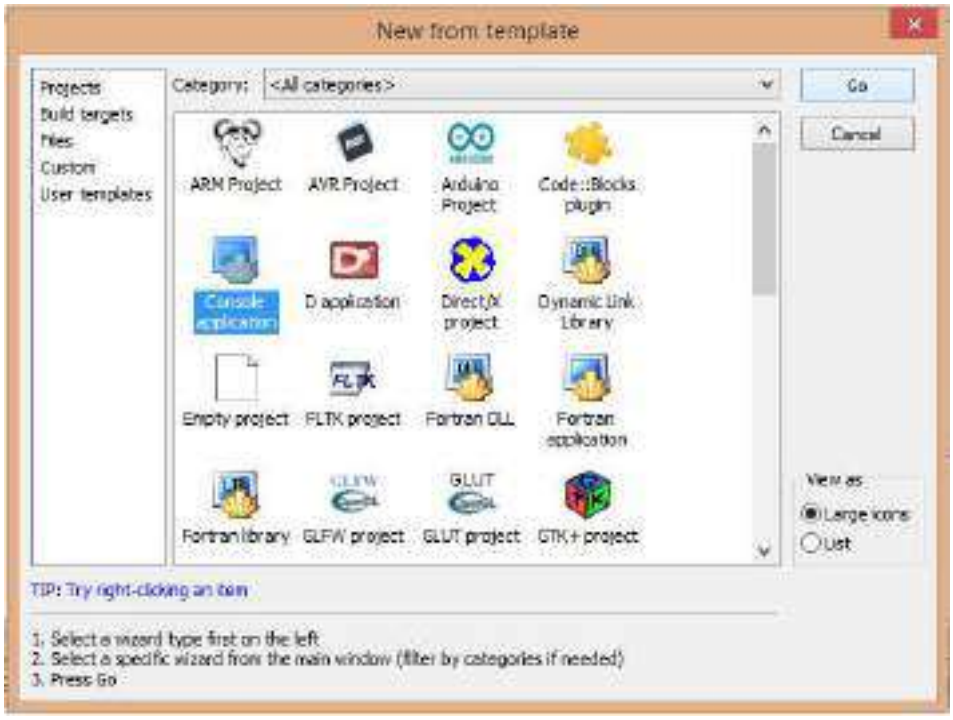


Рис. 3.9. Выбор типа проекта

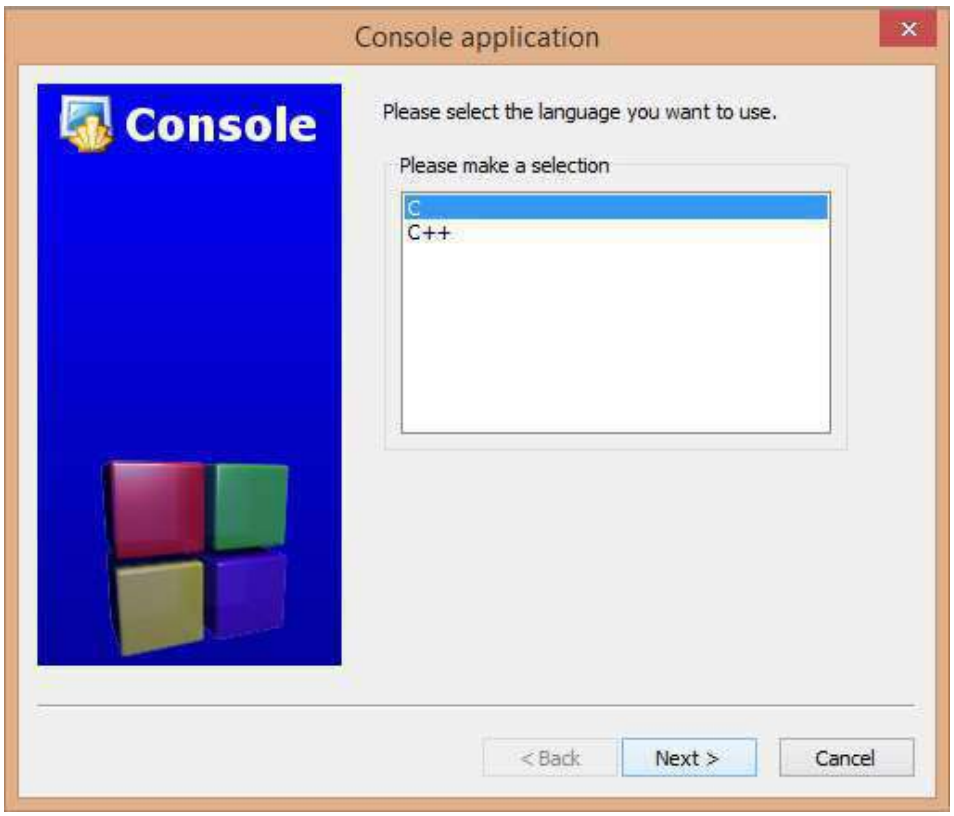


Рис. 3.10. Задание языка разработки

Рис. 3.11. Начальные установки проекта

Рис. 3.12. Подтверждение настроек компилятора

5. В менеджере выбрать папку с проектом и нажать на файл main.c. По умолчанию генерируются код, демонстрирующий вывод текста на экран.
6. Компиляция и запуск приложения осуществляется нажатием клавиши F9 или зеленого треугольника.

3.5.3. Создание консольного приложения в среде Visual Studio

Для создания приложения необходимо выполнить следующую последовательность шагов:

1. Создать папку для хранения будущего приложения.
2. Запустить среду *Visual Studio* из списка установленных приложений.
3. Создать новый проект (рис. 3.13): Файл→Создать→Проект.

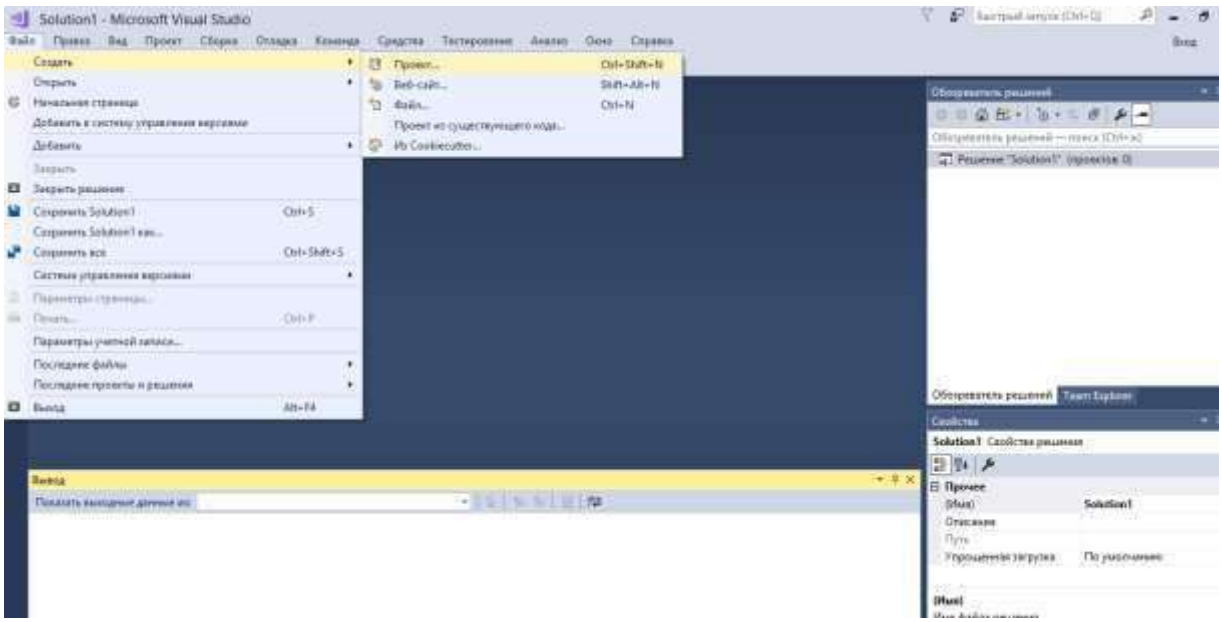


Рис. 3.13. Начало создания проекта в среде *Visual Studio*

4. Задать начальные установки и имя проекта (рис. 3.14,3.15): *Visual C++*→Классическое приложение *Windows*→ Консольное приложение *Windows*. Задать расположение создаваемого проекта (указать созданную ранее папку).

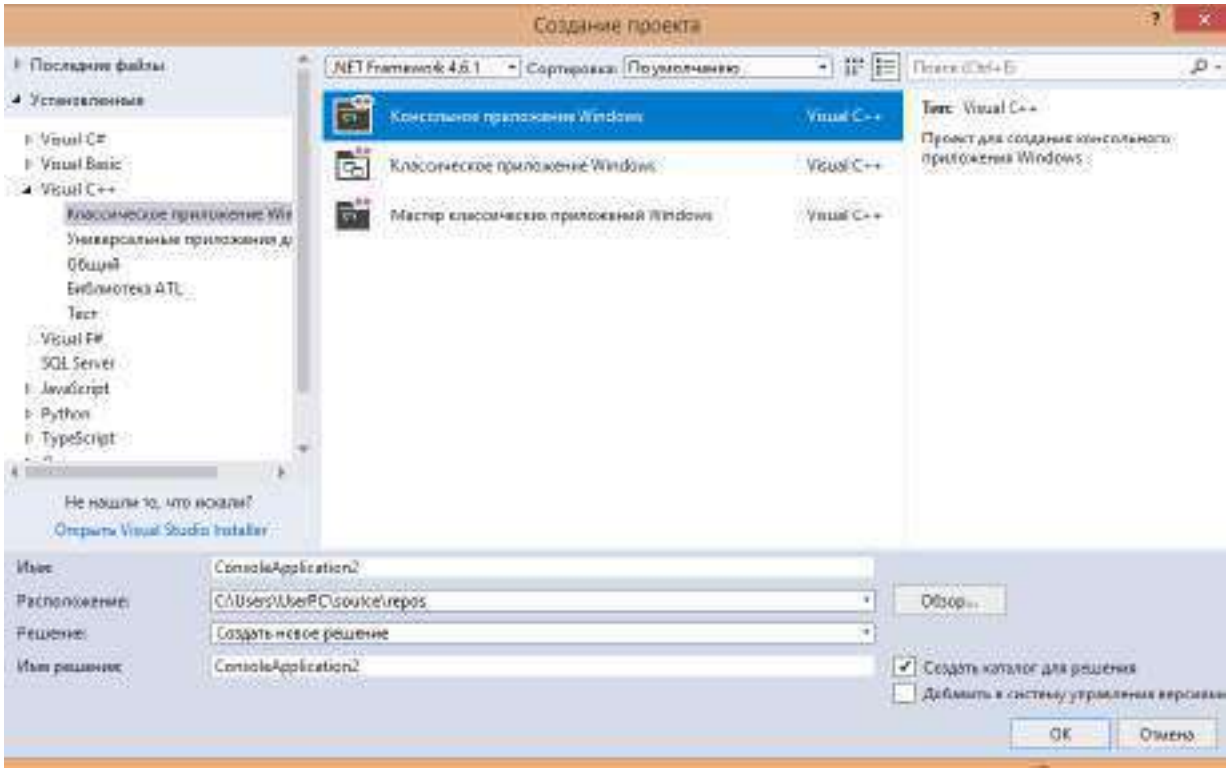


Рис. 3.14. Начальные установки проекта

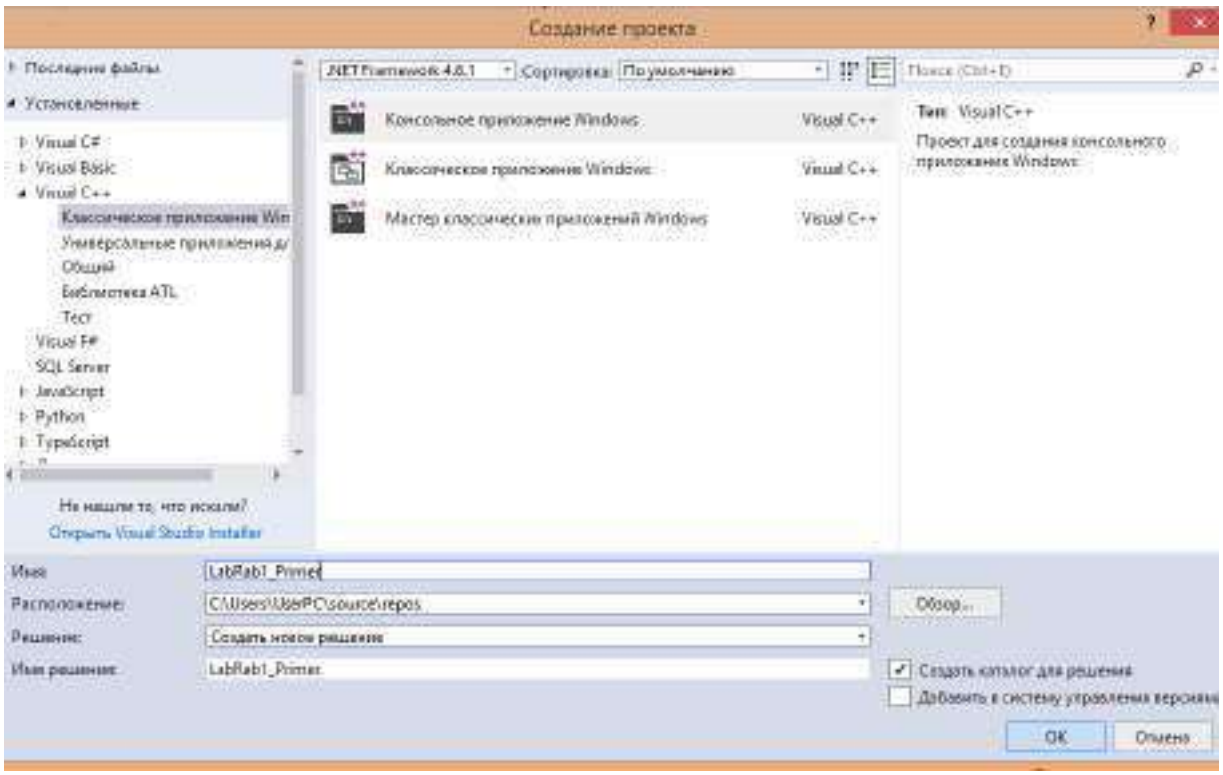


Рис. 3.15. Задание имени проекта

5. Набрать код в открывшемся редакторе (рис. 3.16).

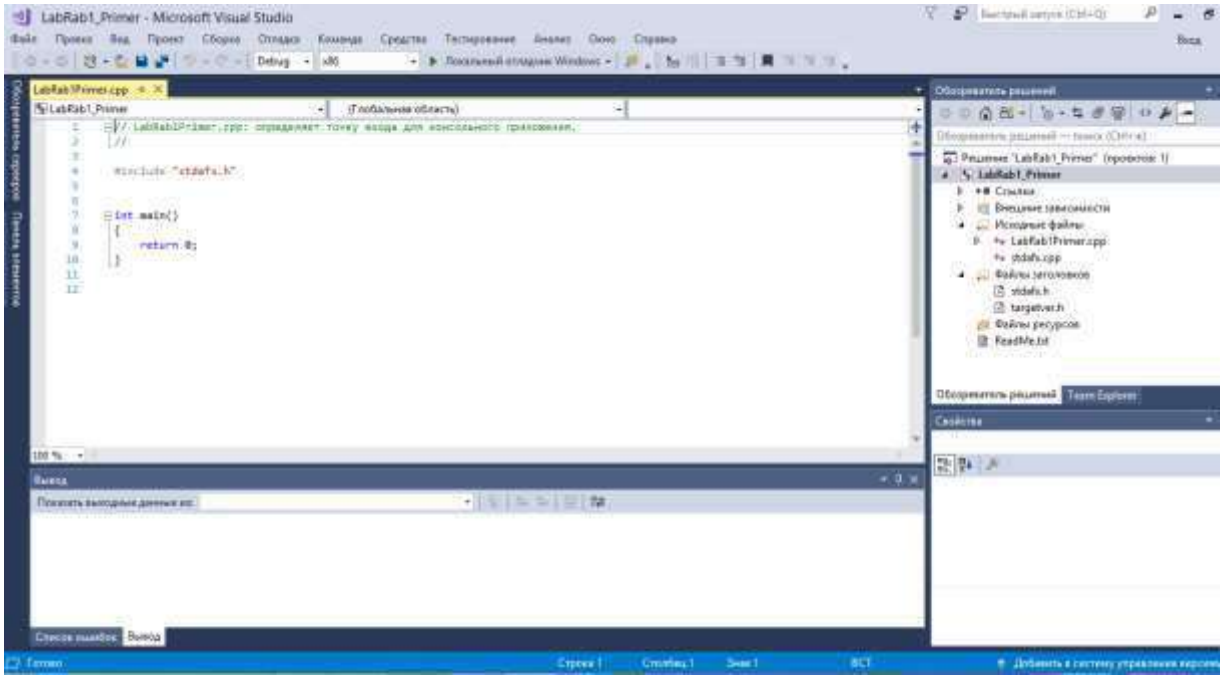


Рис. 3.16. Редактор кода среды *Visual Studio*

6. Выполнить компиляцию проекта (рис. 3.17).

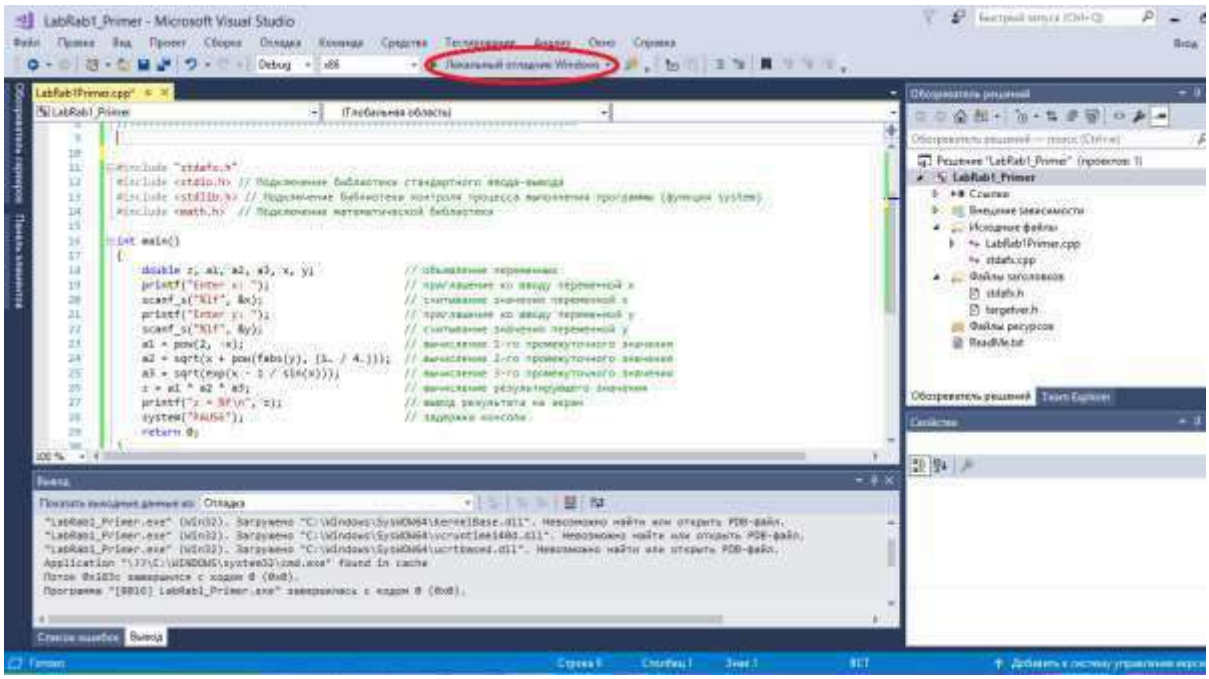


Рис. 3.17. Компиляция проекта

Следует отметить, что при выборе конкретной *IDE* необходимо работать с ее справочной документацией, т.к. среды могут отличаться используемыми в них компиляторами языка.

Так, *Dev-C++* и *Code::Blocks* используют один и тот же компилятор *GCC* для языков *C* и *C++*. Проект, созданный в одной среде, запустится и в другой без дополнительных модификаций.

В *Visual Studio* нужно будет создать новый проект, а код программы может потребовать изменений в силу особенностей компилятора.

3.6. Вопросы для самостоятельного изучения

1. Аспектно-ориентированная парадигма программирования.
2. Событийно-ориентированная парадигма программирования.
3. Агентно-ориентированная парадигма программирования.
4. Компонентно-ориентированная парадигма программирования.
5. Метапрограммирование.
6. Языки параллельного программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Информатика, информационные технологии //Дисциплина информатика: предмет и направления практического применения информатики URL: http://csaa.ru/disciplina-informatika-predmet-i-napravlenija/ (дата обращения 20.04.2020)
2. Информатика, информационные технологии // Тема 5. алгоритмизация и программирование URL: http://csaa.ru/tema-5-algoritmizacija-i-programmirovanie/ (дата обращения 20.04.2020)
3. Бекман И.Н. Информатика: курс лекций [текст] / Московский государственный университет им. М.В. Ломоносова. – М.: Изд-во МГУ, 2009.
4. Планета Информатики // Информатика как наука // URL: https://inf1.info/informaticsscience (дата обращения 15.06.2020)
5. Колокольникова А.И. Спецразделы информатики: основы алгоритмизации и программирования [Электронный ресурс]: практикум. – М.|Берлин: Директ-Медиа, 2019. – 424 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=560695
6. Голов Р.С., Теплышев В.Ю., Шинелев А.А. Комплексная автоматизация в энергосбережении: учебное пособие. – М.: Инфра-М, 2017. – 312 с.
7. Ильиных А.П. Теория алгоритмов: учебное пособие / Урал. гос. пед. ун-т.– Екатеринбург: 2006. – 149 с.
8. Макоха А.Н. Математическая логика и теория алгоритмов [Электронный ресурс]: учебное пособие. – Ставрополь: СКФУ, 2017. – 418 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=467015
9. Пильщиков В.Н., Абрамов В.Г., Вылиток А.А., Горячая И.В. Машина Тьюринга и алгоритмы Маркова. Решение задач: учебно-методическое пособие / 2-е исп. и доп. изд. – М.: МГУ, 2016. – 72 с.
10. Гринченков Д.В., Гринченков В.Д.. Основы теории алгоритмов [Электронный ресурс]: учеб. пособие для студентов, обучающихся по программе среднего профессионального образования, специальность «Программирование в компьютер-

ных системах». - Новочеркасск: ЮРГПУ (НПИ), 2014. - 120 с. – Режим доступа: http://lib.npi-tu.ru/_scripts/show_book2.php?s=12ffb2a497e8453205994c5e1c2d2922bd&i=12&t=pdf&d=1
11. Портал Tproger // Оценка сложности алгоритмов, или Что такое O(log n) URL: https://tproger.ru/articles/computational-complexity-explained/ (дата обращения 20.04.2020)
12. Иванченко А.Н., Масленников А.А., Иванченко П.А. Основы программирования (язык С): учеб. пособие / Юж.-Рос. гос. политехн. ун-т (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2016. – 88 с.
13. Компьютер – это просто! // Первая программа на Assembler URL: https://it-black.ru/pervaya-programma-na-assembler/ (дата обращения 15.06.2020)
14. Лубашева Т.В., Железко Б.А. Основы алгоритмизации и программирования [Электронный ресурс]: учеб. пособие. - Минск: РИПО, 2016. - 378 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=463632
15. Волкова Т.И. Введение в программирование [Электронный ресурс]: учеб. пособие. – М.|Берлин: Директ-Медиа, 2018. - 139 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=493677
16. Царев Р.Ю., Прокопенко А.В. Алгоритмы и структуры данных (CDIO) [Электронный ресурс]: учебник. - Красноярск: СФУ, 2016. - 204 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=497016
17. Городняя Л.В. О классификациях парадигм программирования и параллельном программировании // Образовательные ресурсы и технологии. – 2016. – № 2(14). – С. 138–144.

ПРИЛОЖЕНИЕ 1 Служебные слова псевдокода

Алголо-подобный алгоритмический язык с русским синтаксисом был введен в употребление академиком А.П. Ершовым. Данная методика лежит в основе языка и системы программирования КуМир, предназначенных для поддержки начальных курсов информатики и программирования в средней и высшей школе. В 2018 году НИИСИ РАН была выпущена версия 2.0 пакет КуМир для *Windows* и *Linux*, свободно распространяемая на условиях лицензии *GNU GPL* 2.0.

Рассмотрим основные служебные слова русского алгоритмического языка.

Описание алгоритма:

- **алг** (алгоритм);
- **арг** (аргумент);
- **рез** (результат);
- **нач** (начало) – начало алгоритма;
- **кон** (конец) – конец алгоритма;
- **дано** – исходные данные в произвольной форме;
- **надо** – цель алгоритма;
- **знач** (значения функций) – описываются указанием типа перед именем алгоритма или функции

Типы данных:

- **цел** (целый);
- **вещ** (вещественный);
- **сим** (символьный);
- **лит** (литера) – строка;
- **лог** (логический);
- **таб** (таблица) – для обозначения массива;
- **длин** (длина) – количество элементов массива;

Обозначение условий:

- **если**;
- **то**;
- **иначе**;
- **все**;

– **выбор**;
– **при**.

Обозначение циклов:

– **нц** (начало цикла);
– **кц** (конец цикла);
– **пока**;
– **для**;
– **от**;
– **до**;
– **шаг**.

Логические функции и значения для составления выражений:

– **и**;
– **или**;
– **не**;
– **да**;
– **нет**.

Ввод-вывод:

– **ввод**;
– **вывод**.

В предложении **алг** после названия алгоритма в круглых скобках указываются характеристики и тип значения всех входных (аргументы) и выходных (результаты) переменных.

Пример записи предложения **алг**:

```
алг ОбъемПлощадьЦилиндра (арг вещ R,H, рез вещ V,S)
```

В записи алгоритма ключевые слова могут выделяться подчеркиванием или полужирным шрифтом. Для выделения логических блоков применяются отступы, а парные слова начала и конца блока могут соединяться вертикальной чертой.

ПРИЛОЖЕНИЕ 2 Основные элементы блок-схем

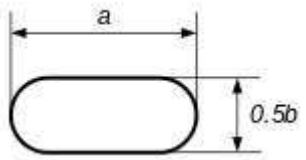
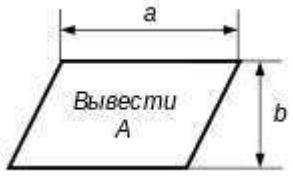
Схема – графическое представление определения, анализа или метода решения задачи, в котором используются символы для отображения данных, потока, оборудования и т.д.

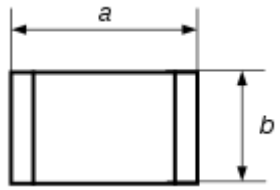
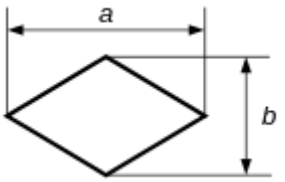
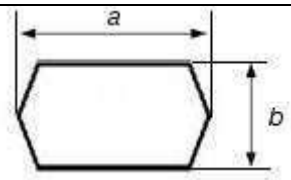
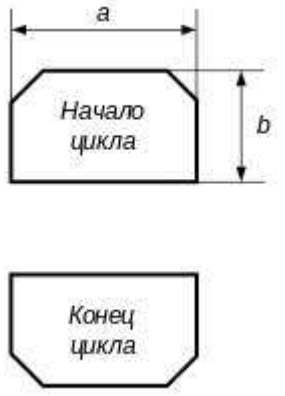
Блок-схема – распространенный тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности.

При изображении элементов рекомендуется придерживаться строгих размеров, определяемых двумя значениями *a* и *b*. Значение *a* выбирается (в миллиметрах) из ряда 15, 20, 25, и т.д. Значение *b* рассчитывается из соотношения $2a = 3b$.

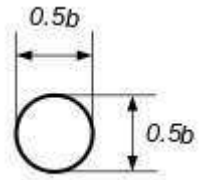
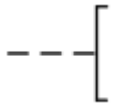
Блок-схема представляет собой совокупность символов (табл. П2.1), соответствующих этапам работы алгоритма и соединяющих их линий.

Таблица П2.1 Элементы блок-схем алгоритмов

Изображение	Назначение элемента
	*Ограничитель (терминатор)* Элемент используется для обозначения начала и конца любой функции (процедуры). Тип возвращаемого значения и аргументов функции обычно указывается в комментариях к блоку терминатора
	*Данные (ввод/вывод)* Перечисляются данные, которые необходимы для работы алгоритма
	*Процесс* В блоке операций обычно размещают одну или несколько (ГОСТ не запрещает) операций присваивания, не требующих вызова внешних функций

	*Предопределенный процесс* Вызов внешних процедур и функций помещается в прямоугольник с дополнительными вертикальными линиями
	*Решение (вопрос или условие)* Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Для случая, когда блок имеет 2 выхода (соответствует оператору ветвления), на них подписывается результат сравнения – «да/нет». Если из блока выходит большее число линий (оператор выбора), внутри него записывается имя переменной, а на выходящих дугах – значения этой переменной
	*Подготовка* Символ «подготовка данных» в произвольной форме (в ГОСТ нет ни пояснений, ни примеров), задает входные значения. Используется обычно для задания циклов со счетчиком
	*Цикл* Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие. Расположение условия определяет тип оператора, соответствующего символам на языке высокого уровня – оператор с предусловием (*while*) или постусловием (*do … while*)

Окончание табл. П2.1

	*Соединитель* Если блок-схема не умещается на лист, используется символ соединителя, отражающий переход потока управления между листами
	*Комментарий* Связь между элементами блок-схемы и пояснениями. Используется для более подробного описания шага, процесса или группы процессов. Описание помещается со стороны квадратной скобки и охватывается ей по всей высоте. Пунктирная линия идет к описываемому элементу, либо группе элементов (при этом группа выделяется замкнутой пунктирной линией). Также символ комментария следует использовать в тех случаях, когда объем текста, помещаемого внутри некоего символа (например, символ процесса, символ данных и др.), превышает размер самого этого символа

Основным направлением потока в схемах алгоритмов принято направление сверху-вниз, слева-направо. Если линии потока идут в основном направлении и не имеют изломов, стрелками их можно не обозначать. В остальных случаях направление линии потока обозначать стрелкой обязательно.

Записи внутри символа должны быть представлены так, чтобы их можно было читать слева направо и сверху вниз, независимо от направления потока.

В схеме символу может быть присвоен идентификатор, который должен помещаться слева над символом.

Допускается краткая информация о символе (описание, уточнение или другие перекрестные ссылки для более полного понимания функции данной части схемы). Описание символа должно помещаться справа над символом.

В случае необходимости слияния линий потока место слияния желательно обозначать точкой или символом 0 (рис. П2.1).

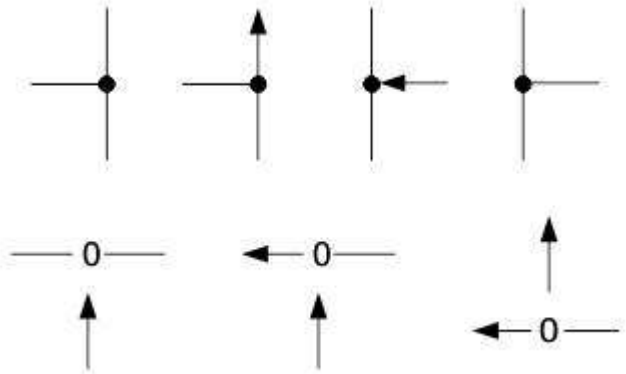


Рис. П2.1. Обозначение слияния потоков на блок-схеме

Все фигуры соединяются линиями (вертикальными и горизонтальными) к середине блока. Каждый блок имеет только одну точку входа и только одну точку выхода (кроме блока «условие», где может быть два и более выходов, на каждом помечается причина\условие). Несколько линий могут соединяться над блоком, нисходящая линия не может разбиваться.

Примеры правильных и неправильных фрагментов блок-схем приведены на рис. П2.2, П2.3.

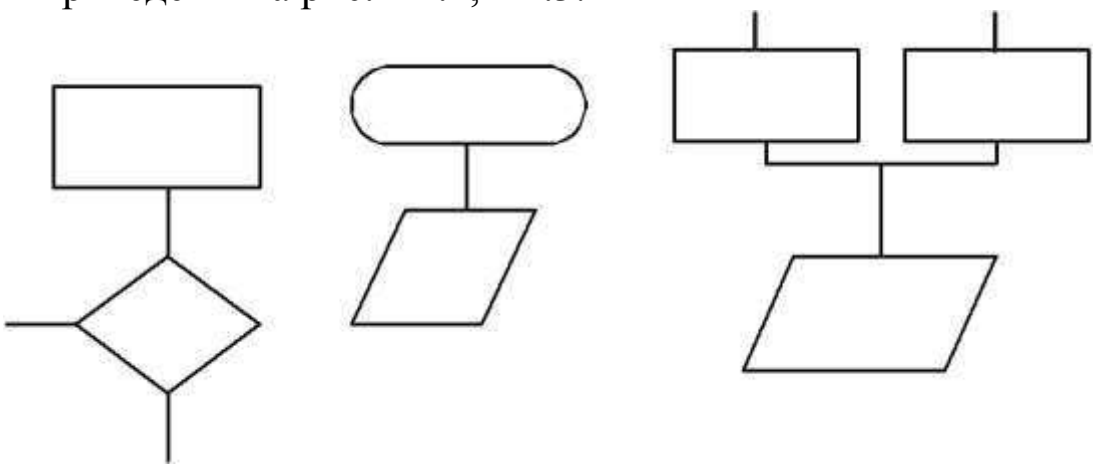


Рис. П2.2. Правильные фрагменты блок-схем

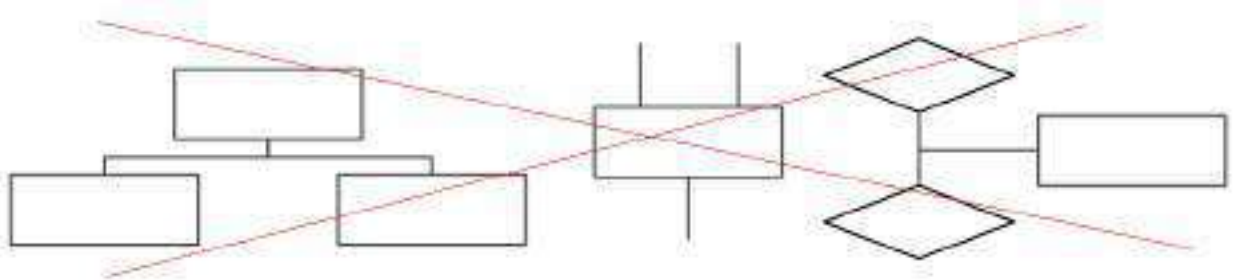


Рис. П2.3. Неправильные фрагменты блок-схем

ПРИЛОЖЕНИЕ 3 Примеры стилей программирования

Рассмотрим листинг программы решения квадратного уравнения на нескольких языках программирования. Будем считать, что коэффициент *a* не равен нулю.

Процедурный стиль. Листинг на языке С

```
// Подключения библиотеки стандартного ввода-вывода
#include <stdio.h>
// Подключение математической библиотеки
#include <math.h>
//Подключение библиотеки локализации (для кириллицы)
#include <locale.h>

//Заголовок функции вычисления корней уравнения
float roots(float a, float b, float c){
    float x1, x2, d;// Объявление переменных
    d=b*b-4*a*c;//Вычисление дискриминанта
    if (d>=0)//
    {   // Дискриминант положительный
        x1=(-b+sqrt(d))/(2*a);//Первый корень
        x2=(-b-sqrt(d))/(2*a); //Второй корень
        //Вывод результата
        printf ("x1=%f x2=%f", x1, x2);
    }
    //Дискриминант отрицательный
    else printf("Вещественных корней нет");
}
//Заголовок функции программы
int main(){
    setlocale(LC_ALL, "Russian");//Отображение кирил-
лицы
    float a, b, c, d, x1, x2;//Объявление переменных
    printf("Введите a: ");//Приглашение ко вводу
    scanf("%f",&a);//Считывание значения переменной
    printf("Введите  b: ");
    scanf("%f",&b);
    printf("Введите c: ");
    scanf("%f",&c);
    //Вызов функции с фактическими параметрами
```

```
    roots(a, b, c);
    return 0;
   }
```

Объектно-ориентированный стиль. Язык C#

```
//Подключение пространства имен System
using System;

namespace ConsoleApp1{//Пространство имен приложения

// Описание квадратного уравнения в виде класса
class Quadratic{
    private double a, b, c;
    private double D;
    private double x1, x2;

    // Метод создания объекта
    public Quadratic(double a, double b, double c){
        this.a = a;
        this.b = b;
        this.c = c;
    }

    // Метод нахождения корней
    public void Roots(){
        D = Math.Pow(b, 2) - 4 * a * c;
        if (D >= 0){
            x1 = (-b + Math.Sqrt(D)) / (2 * a);
            x2 = (-b - Math.Sqrt(D)) / (2 * a);
            Console.WriteLine("x1=  {0:  0.000}\nx2=
        {1: 1.000}", x1, x2);
            Console.ReadKey();
        }
        else{
            Console.WriteLine("Вещественных    корней
        нет");
            Console.ReadKey();
        }
    }
}
// Класс точки входа для приложения
```

```
class Program
{
    // Основной метод
    static void Main(string[] args){
        //Создание экземпляра класса
        Quadratic q = new Quadratic(3, -14, -5);
        //Вызов метода нахождения корнеф
        q.Roots();
        Console.ReadKey();
    }
}
}
```

Функциональный стиль. Язык Haskell

```
--Заголовок функции вычисления корней уравнения
roots :: (Float, Float, Float) -> (Float, Float)

--Реализация функции
--Форма результата зависит от значения дискриминанта
roots (a,b,c) = if d<0 then error "No roots" else
(x1,x2)

--Описывается процедура нахождения корней
--(в обратном порядке)
  where x1 = (-b+sd)/(2*a)
        x2 = (-b-sd)/(2*a)
        d  = b*b - 4*a*c
        sd=sqrt d
--Вывод результатов
main= do print $ roots (1.0,4.0,1.0)
```

Логический стиль. Язык Visual Prolog

```
% начало программы и подключение библиотек
implement main
    open core, stdio, math

% раздел объявлений
class predicates
    % дискриминант
```

```
    discriminant:(real, real, real, uReal[out]).
    % корни уравнения
    roots: (real, real,  uReal) nondeterm.
% раздел определений
clauses
    % нахождение дискриминанта
    discriminant(A, B, C, D):-D=B*B-4*A*C.

    % дискриминант равен нулю – один корень
    roots(A, B, 0):-
        X=(-B)/(2*A),  write("X1=X2=", X).

    % дискриминант больше нуля – два корня
    roots(A, B, D):-
        D>0,
        X1=(-B+sqrt(D))/(2*A),
        X2=(-B-sqrt(D))/(2*A),
        writef("X1=% X2=%", X1, X2).

    % дискриминант меньше – корней нет
    roots(_, _, D):-
         D<0, write("Вещественных корней нет").

    % запуск вычислительного процесса, ввод данных
         run() :-
        write("Введите a, b, c через пробел"), nl,
        A = read(),
        B = read(),
        C = read(), nl,
        discriminant(A, B, C, D),
        roots(A, B, D),
        fail
        or
        _ = readLine().

end implement main

goal
    % запуск вычислительного процесса
    console::runUtf8(main::run).
```

Учебно-методическое издание

Гринченков Дмитрий Валерьевич
Кущий Дарья Николаевна

ИНФОРМАТИКА. ОСНОВЫ АЛГОРИТМИЗАЦИИ

Учебно-методическое пособие
по изучению дисциплины «Информатика»
для направлений подготовки «Бизнес-информатика»,
«Информатика и вычислительная техника»,
«Математическое обеспечение и администрирование
информационных систем», «Прикладная математика»
и «Программная инженерия»

Редактор *Я.В. Максименко*.

Подписано в печать 03.08.2020.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 4,65. Уч.-изд. л. 5,0. Тираж 50 экз. Заказ 46-0345.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

АДМИНИСТРИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Учебное пособие

Новочеркасск
ЮРГПУ (НПИ)
2020

УДК 004.71:004.72 (075.8)

Рецензент – кандидат технических наук, доцент, заведующий кафедры
«Программное обеспечение вычислительной техники»
Гринченков Дмитрий Валерьевич

Синецкий Р. М.

Администрирование компьютерных сетей : учеб. пособие / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова.– Новочеркасск: ЮРГПУ (НПИ), 2020. – 89 с.

В данном учебном пособии представлен теоретический и практический материал по ряду вопросов администрирования компьютерных сетей, который позволит студентам овладеть навыками использования специализированного программного обеспечения и командной строки для моделирования, настройки, поиска и устранения неисправностей сетевого оборудования и компьютерной сети.

Учебное пособие предназначено для студентов очной формы обучения по программе бакалавриата 02.03.03 Математическое обеспечение и администрирование информационных систем, 09.03.01 Информатика и вычислительная техника, 09.03.04 Программная инженерия, а также студентов заочной формы обучения по программе бакалавриата 09.03.01 Информатика и вычислительная техника

УДК 004.71:004.72 (075.8)

Содержание

Кабельная система Ethernet **4**
Стандарты и схемы подключения кабелей *Ethernet* 4
Изготовление прямого кабеля *Ethernet* 6
Проверка работоспособности прямого кабеля *Ethernet* 9
Подключение сетевого кабеля *Ethernet* к розетке 10
Проверка правильности подключения к розетке 13
Подключение узлов сети изготовленным кабелем 14
Диагностика IP-протокола **16**
Проверка настройки конфигурации компьютера 16
Проверка связи с удаленным компьютером 18
Выяснение источника неполадок в маршруте 21
Проверка настроек маршрутизации узла 22
Моделирование компьютерной сети **27**
Сетевые эмуляторы 27
Режим симуляции сети 30
Базовая конфигурация сетевого маршрутизатора **35**
Подключение и настройка маршрутизатора 35
Настройка и проверка *DHCP* 38
Сохранение конфигурации на внешнем сервере 39
Внутриполосное подключение к маршрутизатору 41
Удаление конфигурации оборудованиям 41
Анализ сетевого трафика **43**
Сбор и анализ данных протокола *ICMP* 43
Анализ обмена сообщениями *ARP* 48
Перехват установки сессии TCP 50
Настройка протокола маршрутизации *RIP* **53**
Тестовая сеть 53
Базовая настройка маршрутизатора 54
Проверка общей связности в сети 55
Настройка динамической маршрутизации 56
Планирование адресация *IPv*4 в компьютерных сетях **57**
Двоичные побитовые операции над *IP*-адресами 57
Вычисление адресов для заданной сети 60
Классификация *IPv*4-адресов 62
Определение количества подсетей 65
Разбиение сети на подсети 71
Планирование сетевой адресации 77
Библиографический список **88**

КАБЕЛЬНАЯ СИСТЕМА ETHERNET

Стандарты и схемы подключения кабелей *Ethernet*

Правила использования неэкранированных витых пар в локальных средах определяют стандарты *TIA/EIA*. Существует два коммерческих кабельных стандарта для локальных сетей: *TIA/EIA* 568-*A* и 568-*B*, которые широко применяются в разводке локальных сетей для организаций и, кроме прочего, определяют цвет каждого кабеля для разных контактов. Оба стандарта могут применяться для сетей 10/100/1000*Base-TX Ethernet*.

Для подключения кабеля к сетевым устройствам применяется 8-контактный разъем *RJ*-45, назначение выводов которого определяет стандарт.

Таблица 1 – Схема 568-*A TIA/EIA*

Номер контакта	Номер пары	Цвет провода	10Base-T 100Base-TX	1000Base-T
1	2	Белый/зелёный	Передача	BI_DA+
2	2	Зелёный	Передача	BI_DA-
3	3	Белый/оранжевый	Приём	BI_DB+
4	1	Синий	Не используется	BI_DC+
5	1	Белый/синий	Не используется	BI_DC-
6	3	Оранжевый	Приём	BI_DB-
7	4	Белый/коричневый	Не используется	BI_DD+
8	4	Коричневый	Не используется	BI_DD-

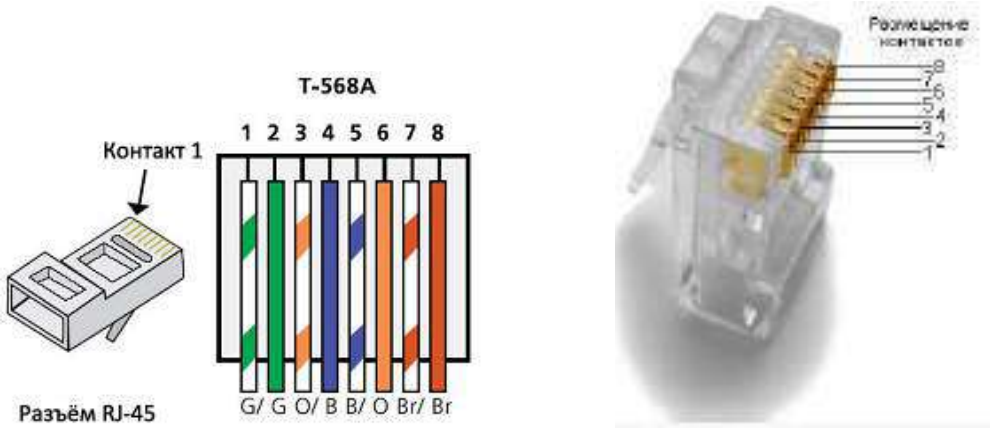


Рисунок 1 – Расположение контактов *RJ*-45

Таблица 1 демонстрирует цветовую схему и расположение выводов, а также работу четырёх пар проводов, предусмотренных стандартом *TIA/EIA* 568-*A*.

На рисунке 1 показано, как цвета и расположение выводов разъёма *RJ*-45 соотносятся со стандартом 568-*A*.

Цветовая схема и расположение выводов для стандарта **TIA/EIA 568-B** показана в таблице 2.

Таблица 2 – Схема 568-*B TIA/EIA*

Номер контакта	**Номер пары**	**Цвет провода**	**10Base-T 100Base-TX**	**1000Base-T**
1	2	Белый/оранжевый	Передача	BI_DA+
2	2	Оранжевый	Передача	BI_DA-
3	3	Белый/зелёный	Приём	BI_DB+
4	1	Синий	Не используется	BI_DC+
5	1	Белый/синий	Не используется	BI_DC-
6	3	Зелёный	Приём	BI_DB-
7	4	Белый/коричневый	Не используется	BI_DD+
8	4	Коричневый	Не используется	BI_DD-

Проанализируйте цветовое расположение контактов стандартов 568-A и 568-B. В чем разница между стандартами?

В сетевых кабельных системах применяет два типа кабеля: прямой и кроссовый.

Прямой сетевой кабель на обоих концах имеет разъем *RJ*-45 к которому подключены провода по одинаковой цветовой схеме. То есть на обоих концах прямого кабеля применен один и тот же стандарт 568-*A* и 568-*B*. Прямой кабель используется для подключения узлов сети *Ethernet* к коммутирующему оборудованию, например, компьютера к концентратору, компьютера к коммутатору, маршрутизатора к коммутатору и т.п. Подключать узлы сети друг к другу прямым кабелем не рекомендуется (например, компьютер к компьютеру или компьютер к маршрутизатору). В этом случае некоторые сетевые узлы могут не работать.

В ***кроссовом кабеле*** вторая и третья пары разъёма *RJ*-45 на одном конце кабеля переворачиваются на другом конце, что меняет местами пары отправки и приёма. На одном конце кабеля исполь-

зуется схема подключения кабеля со стандартом 568-*A*, а на другом – со стандартом 568-*B*. Кроссовые кабели обычно используются для подключения концентраторов к концентраторам, коммутаторов к коммутаторам, маршрутизаторов к маршрутизаторам, а также компьютеров к компьютерам напрямую.

Современные сетевые устройства могут иметь функцию автоматического определения типа кабеля и инвертирования приемопередающих пар, что позволяет использовать для их подключения как прямой, так и кроссовый кабель.

Объясните, почему кроссовый кабель позволяет связать узлы сети друг с другом, минуя коммутирующее оборудование.

Изготовление прямого кабеля *Ethernet*

Если необходимо изготовить кабель *Ethernet*, то вам понадобятся: обжимное устройство (рисунок 2, *а*), кабель *UTP* категории 5 и два разъема *RJ*-45 (рисунок 2, *б*).

а *б*

Рисунок 2 – Оборудование и материалы для изготовления кабеля

Последовательность действий выглядит следующим образом:

1) Отрежьте примерно 30–40 см кабеля.

2) Очистите от оболочки оба конца кабеля на 2 см. Раскрутите и распрямите провода, как показано на рисунке 3.

3) Выберите стандарт 568-*A* или 568-*B*, по которому будете соединять кабель. Выстройте провода в порядке, соответствующем

выбранному стандарту. При необходимости обращайтесь к рисунку.

Рисунок 3 – Очистка кабеля от изоляции

4) Сплющите, выпрямите и выровняйте провода.

5) Убедитесь в том, что провода кабеля расположены в правильном порядке, соответствующем стандарту. С помощью кусачек обрежьте четыре пары в прямую линию до длины 1,3–1,7 см. Должно получиться так, как показано на рисунке 4.

Рисунок 4 – Распрямление проводов

6) На конце кабеля установите разъём *RJ*-45, выступ-зажим которого должен быть направлен вниз. Нумерация контактов начинается слева. Плотно вставьте провода в разъём *RJ*-45. Все провода должны быть видны в конце разъёма на соответствующих местах. Если провода не достигают конца разъёма, извлеките кабель, ровно обрежьте провода и вставьте провода обратно в разъём *RJ*-45. Пример правильно установленного разъема показан на рисунке 5.

Рисунок 5 – Установка разъема

7) Если всё сделано правильно, вставьте разъём *RJ*-45 с кабелем в обжимной инструмент, ка показано на рисунке 6.

Рисунок 6 – Вставка разъема в обжимной инструмент

8) Сожмите кабель в инструменте до упора, так чтобы контакты на разъёме *RJ*-45 прошли через изоляцию проводов, обеспечивая таким образом электрическую связь. Зажим в нижней части разъема фиксирует оболочку кабеля, не давая ему выпасть из разъема. Рисунок 7 иллюстрирует этот этап.

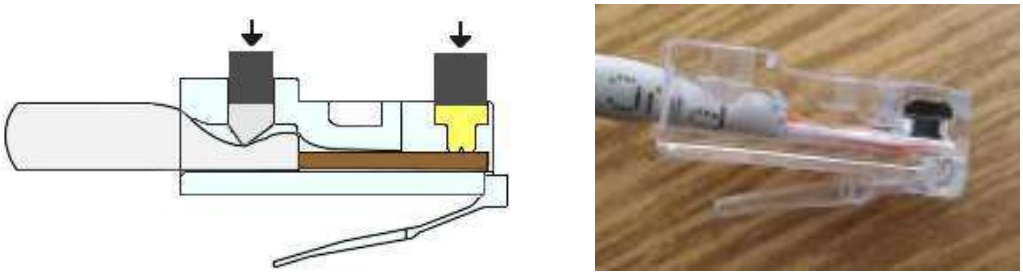


Рисунок 7 – Фиксация разъема

9) Повторите все операции с другом концом кабеля. Используйте одинаковый стандарт на обоих концах.

В результате вы получили готовый прямой кабель *Ethernet*. Прежде, чем использовать его в компьютерной сети, необходимо убедиться в его работоспособности.

Проверка работоспособности прямого кабеля *Ethernet*

Для проверки работоспособности вам понадобится кабельный тестер для кабелей *Ethernet*. Пример кабельного тестера показан на рисунке 8.

Рисунок 8 – Кабельный тестер

Вставьте оба конца готового кабеля в гнезда кабельного тестера. Передвиньте тумблер на боковой панели тестера в положение On. Светодиоды на передней панели тестера отображают соединение между проводами концов кабеля. При правильно обжатом прямом кабеле светодиоды должны загораться напротив друг друга, как показано на рисунке 9.

Рисунок 9 – Проверка кабеля тестером

Если какая-то пара светодиодов не загорается, это означает, что электрический контакт по этому проводу отсутствует. Если светодиоды загораются не напротив друг друга, это означает, что цвета проводов были перепутаны, или что концы кабеля обжаты

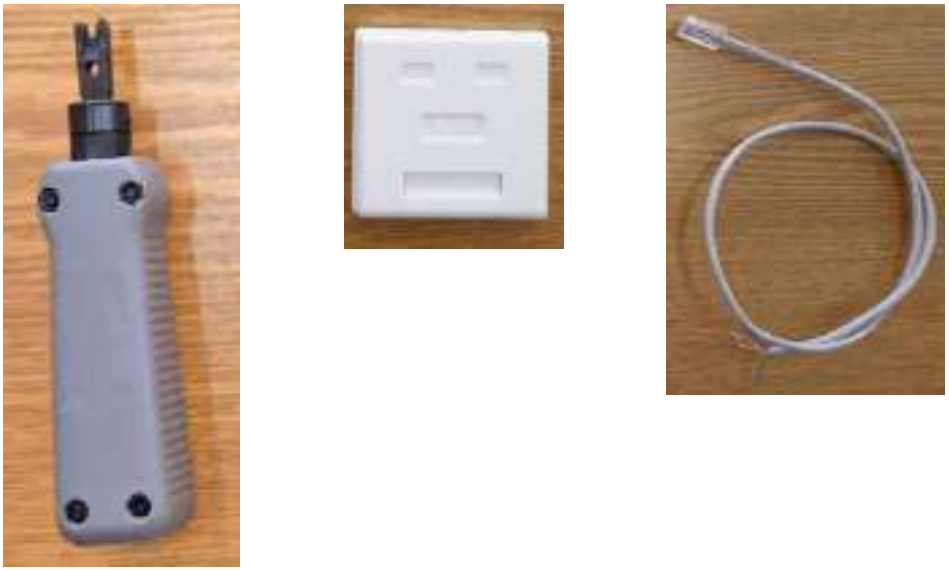
по разным стандартам (кроссовый кабель).

Ваш кабель изготовлен правильно? Если нет, опишите его неисправности и причины их возникновения.

Подключение сетевого кабеля *Ethernet* к розетке

Нередко при прокладке структурированной кабельной сети возникает необходимость в подключении сетевого кабеля к розетке. Розетки удобны тем, что располагаются на стене, к ним можно при помощи прямого кабеля подключить любой сетевой узел.

Для подключения кабеля к сетевой розетке вам понадобится: крепежный инструмент – импактор (рисунок 10, *а*), сетевая розетка (рисунок 10, *б*), обжатый с одной стороны кабель (рисунок 10, *в*).



а *б* *в*

Рисунок 10 – Оборудование и материалы
для подключения к сетевой розетке

Откройте сетевую розетку. Изнутри в ней расположены контакты для подключения сетевого кабеля, которые подведены к гнезду *Ethernet*. Чаще всего встречаются розетки на 1 или 2 гнезда, как показано на рисунке 11.

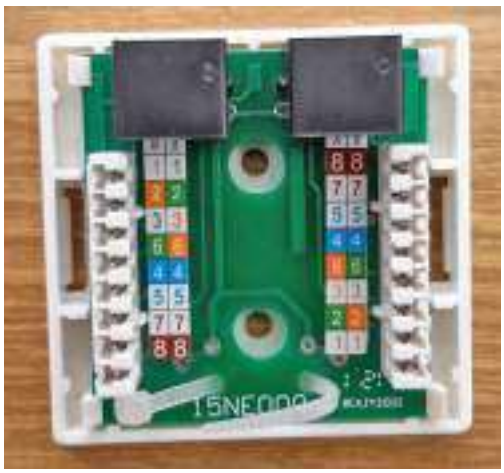


Рисунок 11 – Сетевая розетка изнутри

Возьмите кабель, обжатый с одной стороны. На одном конце кабеля уже установлен разъем *RJ*-45, обжатый по стандарту 568-*B*. С другой стороны кабеля снимите изоляцию на 3–4 см, раскрутите и распрямите провода.

Подключать кабель к розетке необходимо по стандарту 568-*B*. Порядок следования цветов провода может не совпадать с порядком для разъема *RJ*-45, потому что определяется разводкой печатной платы розетки. На самой плате нарисована правильная для этой розетки цветовая схема, необходимо ориентироваться на колонку, подписанную как *B*.

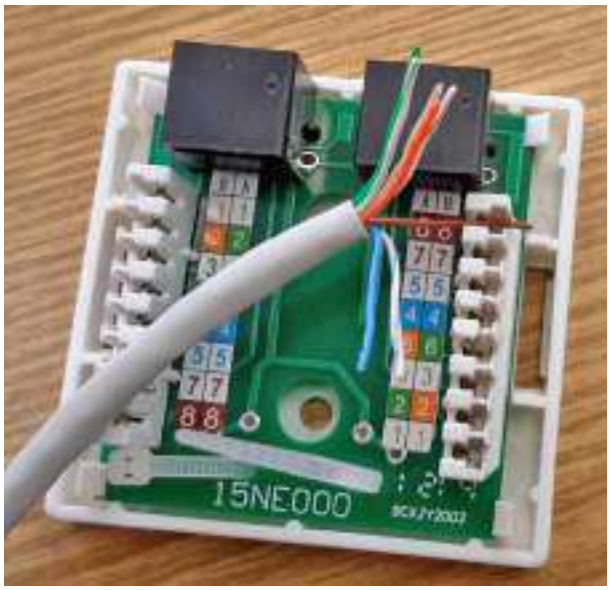


Рисунок 12 – Вставка провода в розетку

Для закрепления провода в колодке розетки заведите его изнутри розетки наружу и поместите в прорезь напротив нужного цвета, как показано на рисунке 12.

Для закрепления провода в колодке необходимо использовать крепежный инструмент, который обычно называют импактором (в иностранной литературе встречаются названия «*impact tool*», «*punch tool*» или «*krone tool*»). На конце инструмента находится вилочка, одна из сторон который имеет острую кромку для отрезания провода (рисунок 13).

Рисунок 13 – Рабочая часть импактора

Установите инструмент вилочкой в прорезь колодки розетки. *Острый край вилочки инструмента должен быть снаружи колодки*, как показано на рисунке 14. Он отрезает излишки провода. Если импактором пользовались достаточно часто, его острая кромка может затупиться, тогда провод отрезаться не будет. У некоторых моделей импакторов в комплекте имеются запасные части.

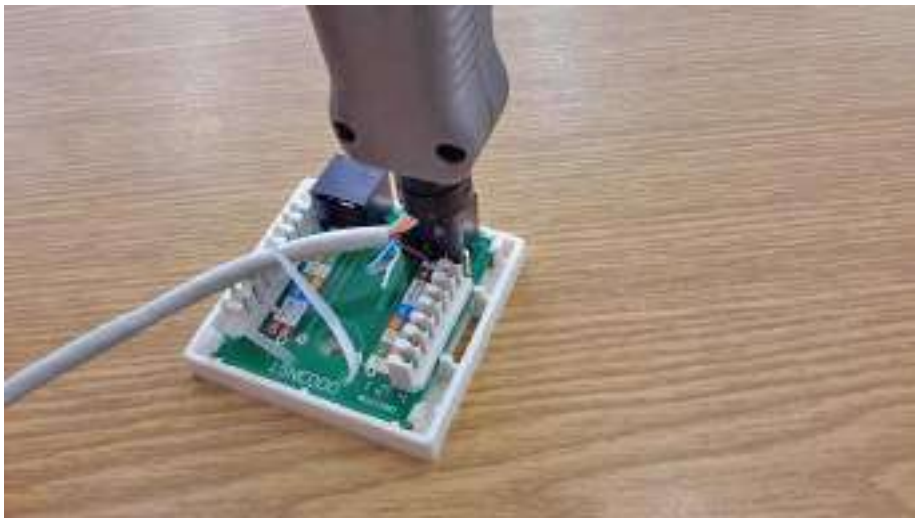


Рисунок 14 – Фиксация провода импактором

Надавите на инструмент сверху вниз до щелчка. Провод закреплен.

Закрепите при помощи инструмента все восемь проводов в соответствии с цветом по стандарту 568-*B*.

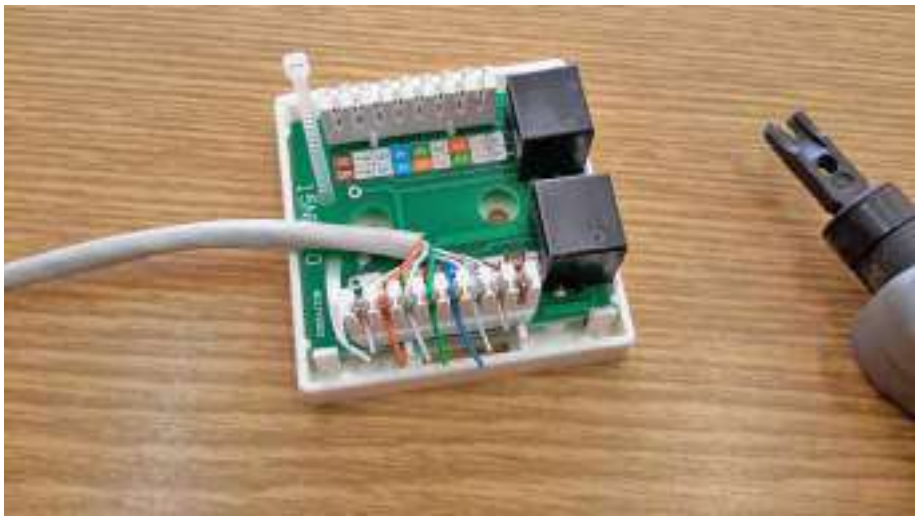


Рисунок 15 – Готовая розетка

В результате вы подключили кабель к сетевой розетке, как показано на рисунке 15. Перед использованием полученной розетки при построении сети, необходимо протестировать ее в работе.

Проверка правильности подключения к розетке

Для проверки работоспособности сетевой розетки вам понадобится кабельный тестер и готовый прямой кабель.

Вставьте один конце готового кабеля в гнездо розетки, второй конец готового кабеля вставьте в кабельный тестер. Обжатый конец провода, подключенного к розетке, вставьте в свободный разъем кабельного тестера.

Переключите тумблер тестера в положение ***On***. Аналогично проверке прямого кабеля *Ethernet*, светодиоды на передней панели тестера должны загораться напротив друг друга, как показано на рисунке 16.

Если какая-то пара светодиодов не загорается, это означает, что электрический контакт по этому проводу отсутствует. Если светодиоды загораются не напротив друг друга, это означает, что цвета проводов были перепутаны.

Ваша розетка подключена правильно? Если нет, опишите неисправности и причины их возникновения.

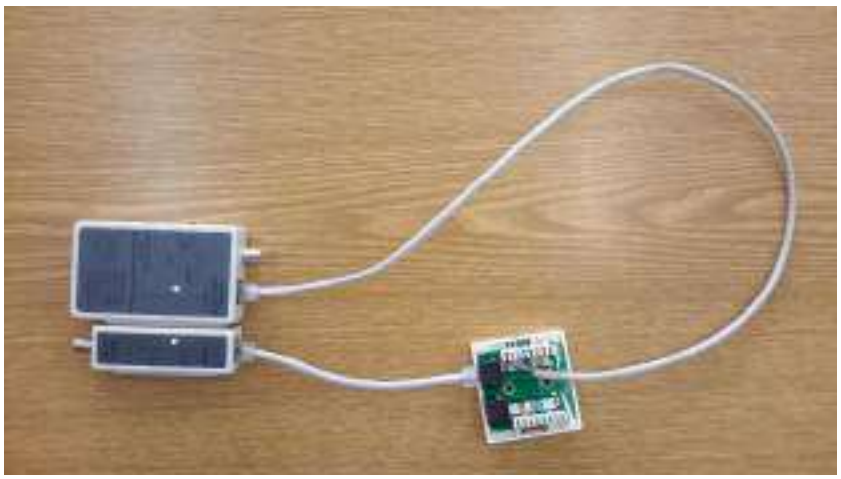


Рисунок 16 – Проверка правильности подключения к розетке

Подключение узлов сети изготовленным кабелем

Имея готовый обжатый прямой сетевой кабель можно использовать его для построения компьютерной сети. Одна из простейших топологий компьютерной сети показана на рисунке 17, а соответствующая ей схема адресации – в таблице 3.

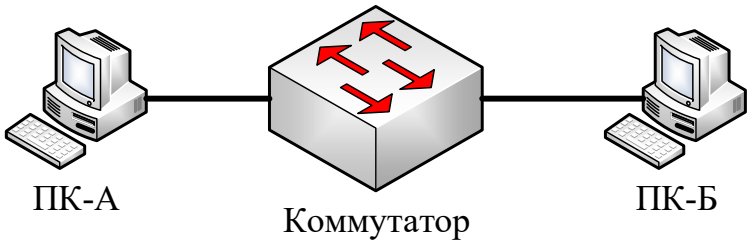


Рисунок 17 – Топология тестовой сети

Таблица 3 – Таблица адресации

Узел	Порт	IP-адрес	Маска	Шлюз	DNS
ПК 1	LAN	192.168.10.2	255.255.255.0	192.168.10.1	192.168.10.1
ПК 2	LAN	192.168.10.3	255.255.255.0	192.168.10.1	192.168.10.1

Для построения такой сети понадобится четыре исправных прямых кабеля *Ethernet* (при необходимости можно использовать готовые заводские патч-корды), два компьютера, патч-панель с подключенными розетками, коммутатор.

Соедините одним кабелем гнездо сетевой карты компьютера (интерфейс *LAN* или *CiscoLAN*) со свободным гнездом сетевой розетки на стене.

Вторым кабелем соедините на патч-панели в стойке выход сетевой розетки со свободным портом коммутатора. Аналогично подключите второй компьютер.

Установите на каждом компьютере сетевой адрес в соответствии с таблицей адресации. Для этого запустите программу *ipconf.exe* (рисунок 18) или стандартный интерфейс операционной системы для настройки IP-адреса.

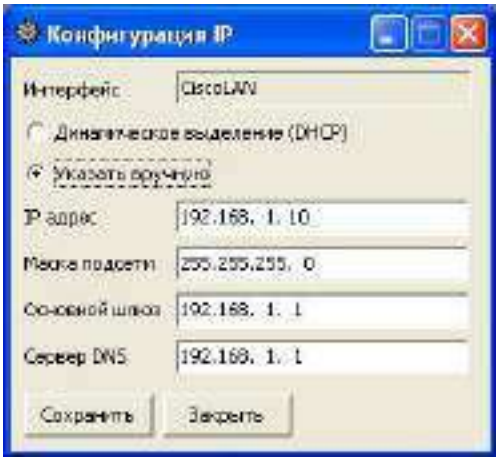


Рисунок 18 – Настройка сетевого адреса

В программе укажите адрес, маску, шлюз и *dns* в соответствии с таблицей для каждого компьютера. Нажмите кнопку сохранить и дождитесь появления окна с сообщением «Параметры установлены».

Через 15–30 секунд после подключения на коммутаторе рядом с портами, в которые подключены кабели, должны загореться зеленые светодиоды. Если они не горят, или горят оранжевым, в кабельной системе неисправность (проверьте правильность кабелей, правильность номеров портов и сетевых карт на компьютерах).

Для проверки связи между компьютерами можно использовать стандартную утилиту *ping*. Она выполняет отправку нескольких служебных пакетов другому компьютеру в сети, дожидается от него ответа и отображает время ожидания и результат. Если программа показывает, что время ожидания истекло и ответа от другого узла не получено, связи между компьютерами нет.

Запустите командную строку *Windows* (Пуск → Выполнить → *cmd.exe*).

На первом компьютере подайте команду:

```
ping 192.168.10.3
```

На втором компьютере подайте команду:

```
ping 192.168.10.2
```

Была ли проверка связи между узлами успешной? Если нет, опишите неисправности и причины их возникновения.

ДИАГНОСТИКА IP-ПРОТОКОЛА

Проверка настройки конфигурации компьютера

В операционных системах *Microsoft Windows* имеется утилита командной строки *ipconfig* для вывода деталей текущего соединения и управления клиентскими сервисами *DHCP* и *DNS*. Утилита *ipconfig* позволяет определять, какие значения конфигурации были получены с помощью *DHCP* либо заданы администратором вручную. Утилита имеет множество функций, которые задаются через ключи и параметры командной строки. Некоторые из этих ключей представлены в таблице 4.

Таблица 4 – Доступные ключи ipconfig

Ключ	Описание
/*all*	Отображение полной информации по всем адаптерам.
/*release* [адаптер]	Отправка сообщения *DHCPRELEASE* серверу *DHCP* для освобождения текущей конфигурации *DHCP* и удаления конфигурации *IP*-адресов для всех адаптеров (если адаптер не задан) или для заданного адаптера. Этот ключ отключает протокол *TCP/IP* для адаптеров, настроенных для автоматического получения *IP*-адресов.
/*renew* [адаптер]	Обновление *IP*-адреса для определённого адаптера или если адаптер не задан, то для всех. Доступно только при настроенном автоматическим получением *IP*-адресов.
/*flushdns*	Очищение *DNS* кэша.
/*registerdns*	Обновление всех зарезервированых адресов *DHCP* и перерегистрация имен *DNS*.
/*displaydns*	Отображение содержимого кэша *DNS*.
/*showclassid* адаптер	Отображение кода класса *DHCP* для указанного адаптера. Доступно только при настроенном автоматическим получением *IP*-адресов.
/*setclassid* адаптер [код_класса]	Изменение кода класса *DHCP*. Доступно только при настроенном автоматическим получением *IP*-адресов.
/?	Справка.

Для применения *ipconfig* на практике нажмите кнопку *Пуск*, выберите строку меню *Выполнить*, наберите символы *cmd* и нажмите клавишу *Enter* на клавиатуре.

В открывшемся окне наберите *ipconfig /all*. При нормальной

работе компьютера на экран должен вывестись примерно такой листинг:

```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : techstation
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Atheros L1 Gigabit
Ethernet 10/100/1000Base-T Controller
   Physical Address. . . . . . . . . : 00-1D-60-80-57-2B
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.1.20(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 192.168.1.1
   DNS Servers . . . . . . . . . . . : 87.117.0.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Tunnel adapter isatap.{6786B578-3A7B-4AEA-B733-B009EB8C7BB3}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft ISATAP
Adapter
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Teredo Tunneling
Pseudo-Interface
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . :
2001:0:5ef5:73bc:14db:d458:af01:98f1(Pref
erred)
   Link-local IPv6 Address . . . . . :
fe80::14db:d458:af01:98f1%13(Preferred)
   Default Gateway . . . . . . . . . : ::
   NetBIOS over Tcpip. . . . . . . . : Disabled
```

Попробуйте отключить сетевое подключение (можно также отключить сетевой кабель от сетевой карты компьютера), повторите команду. При отсутствующем соединении на экран выводится примерно такой листинг:

```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : techstation
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Teredo Tunneling
Pseudo-Interface
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
```

Попробуйте работу утилиты *ipconfig*. Сколько сетевых адаптеров на компьютере?

Проверка связи с удаленным компьютером

Если необходимо выяснить, работает ли удаленный компьютер, имеется ли с ним связь, то можно воспользоваться утилитой ping.

ping – утилита для проверки соединений в сетях на основе *TCP/IP*. Она отправляет запросы (*ICMP Echo-Request*) протокола *ICMP* указанному узлу сети и фиксирует поступающие ответы (*ICMP Echo-Reply*). Время между отправкой запроса и получением ответа (*RTT*, от англ. *Round Trip Time*) позволяет определять двусторонние задержки (*RTT*) по маршруту и частоту потери пакетов, то есть косвенно определять загруженность на каналах передачи данных и промежуточных устройствах.

Также ***пингом*** иногда называют время, затраченное на передачу пакета информации в компьютерных сетях от клиента к серверу и обратно от сервера к клиенту. Это время также называется

лагом (англ. *отставание*, *задержка*, *запаздывание*) или собственно задержкой и измеряется в миллисекундах. Лаг связан со скоростью соединения и загруженностью каналов на всём протяжении от клиента к серверу.

Полное отсутствие *ICMP*-ответов может также означать, что удалённый узел (или какой-либо из промежуточных маршрутизаторов) блокирует *ICMP Echo-Reply* или игнорирует *ICMP Echo-Request.*

Практическое использование утилиты *ping*:

– можно узнать, работает ли удаленный сервер.

– можно узнать завис ли только удаленный сервер или в сети глобальные проблемы.

– проблемы с настройкой *DNS* серверов на компьютере можно выявить, задав в *ping* сначала доменное имя, а потом *IP*-адрес.

– можно узнать скорость соединения, так как *ping* показывает сколько запросов удалось выполнить в секунду.

– можно узнать качество канала, посмотрев сколько ответов не пришло.

– и т.д.

Рассмотрим несколько примеров. Для проверки правильности установки протокола *TCP/IP* в системе откройте командную строку и выполните команду:

```
ping 127.0.0.1
```

Адрес 127.0.0.1 – это локальный адрес любого компьютера. Таким образом, эта команда проверяет прохождение пакетов *на самого себя*. Она может быть выполнена без наличия какого-либо сетевого подключения. Вы должны увидеть приблизительно следующие строки:

```
Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
```

```
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

По умолчанию команда посылает пакет 32 байта. Размер пакета может быть увеличен до 65 кбайт. Так можно обнаружить ошибки при пересылке пакетов больших размеров. За размером тестового пакета отображается время отклика удаленной системы (в нашем случае – меньше 1 миллисекунды). Потом показывается еще один параметр протокола – значение *TTL*. *TTL* – «время жизни» пакета. На практике это число маршрутизаторов, через которые может пройти пакет. Каждый маршрутизатор уменьшает значение *TTL* на единицу. При достижении нулевого значения пакет уничтожается. Такой механизм введен для исключения случаев зацикливания пакетов в сети.

Если будет показано сообщение о недостижимости пакетов, то это означает ошибку установки протокола IP.

Для проверки связи с другим компьютером сети выполните команду:

```
ping 192.168.100.1
```

На экран должны быть выведены примерно такие строки:

```
Pinging 192.168.100.1 with 32 bytes of data:
Reply from 192.168.100.1: bytes=32 time=491ms TTL=254
Reply from 192.168.100.1: bytes=32 time=493ms TTL=254
Reply from 192.168.100.1: bytes=32 time=490ms TTL=254
Reply from 192.168.100.1: bytes=32 time=488ms TTL=254

Ping statistics for 192.168.100.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 488ms, Maximum = 493ms, Average = 490ms
```

Наличие отклика свидетельствует о том, что канал связи установлен и работает.

Узнайте при помощи утилиты *ipconfig* адрес другого компьютера в сети и попробуйте команду *ping* с его адресом. Была ли связь успешной? Если нет, то какие могут быть причины отсутствия связи?

Выяснение источника неполадок в маршруте

При работе в Сети одни информационные серверы откликаются быстрее, другие медленнее, бывают случаи недостижимости узла. Для выяснения причин в подобных ситуациях можно использовать специальные утилиты.

Например, команда ***tracert*** показывает путь прохождения пакетов до желаемого узла. Зачастую это позволяет выяснить причины плохой связи с ним. Точка в маршруте до узла, до которой время отклика резко увеличено, свидетельствует о наличии в этом месте «узкого горлышка», не справляющегося с нагрузкой.

Утилита *tracert* выполняет отправку данных указанному узлу сети по протоколу *ICMP*, при этом отображая сведения о всех промежуточных маршрутизаторах, через которые прошли данные на пути к целевому узлу.

Программа работает только в направлении от источника пакетов. В силу особенностей работы протоколов маршрутизации в сети Интернет, обратные маршруты часто не совпадают с прямыми, поэтому *ICMP* ответ от каждого промежуточного узла может идти своим собственным маршрутом, затеряться или прийти с большой задержкой, хотя в реальности с пакетами, которые адресованы конечному узлу, этого не происходит. Кроме того, на промежуточных маршрутизаторах могут стоять ограничения протокола *ICMP*, что приводит к появлению ложных потерь. Тем не менее, в подавляющем большинстве случаев, *tracert* является очень полезным инструментом диагностики сети.

Принцип работы tracert следующий. Для определения промежуточных маршрутизаторов tracert отправляет серию (обычно три) пакетов *ICMP* целевому узлу, при этом каждый раз увеличивая на 1 значение поля *TTL*. Первая серия пакетов отправляется с *TTL*, равным 1, и поэтому первый же маршрутизатор возвращает обратно сообщение *ICMP*, указывающее на невозможность доставки данных. *Tracert* фиксирует адрес маршрутизатора, а также время между отправкой пакета и получением ответа. Затем *tracert* повторяет отправку серии пакетов, но уже с *TTL*, равным 2, что позволяет первому маршрутизатору пропустить их дальше, тогда мы получим ответ от второго маршрутизатора в пути до целевого компьютера и так далее.

Процесс повторяется до тех пор, пока при определённом значении *TTL* пакет не достигнет целевого узла. При получении ответа от этого узла процесс трассировки считается завершённым.

Рассмотрим пример использования утилиты *tracert*. В командной строке введите команду:

```
tracert ya.ru
```

Вы должны увидеть примерно такой листинг:

```
Tracing route to ya.ru [213.180.204.8]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  192.168.1.1
  2     5 ms     4 ms     5 ms  intor.novoch.ru [80.254.102.1]
  3     5 ms     5 ms     5 ms  8.108.254.80.static.donpac.ru
[80.254.108.8]
  4     6 ms     8 ms     7 ms  cs7-rmts.donpac.ru
[80.254.111.1]
  5    31 ms    28 ms    32 ms  platov-rnd-ix.yandex.net
[193.232.140.33]
  6    64 ms    66 ms    70 ms  korolev-vlan2301.yandex.net
[213.180.208.18]
  7    32 ms    32 ms    34 ms  popovich-vlan121.yandex.net
[87.250.233.110]
  8    34 ms    32 ms    32 ms  gallium-vlan2.yandex.net
[87.250.228.139]
  9    36 ms    40 ms    39 ms  ya.ru [213.180.204.8]

Trace complete.
```

Проверка настроек маршрутизации узла

Утилита ***route*** выводит на экран и изменяет записи в локальной таблице *IP*-маршрутизации. Запущенная без параметров, команда *route* выводит справку. Синтаксис команды:

```
route [-f] [-p] [команда [конечная_точка] [mask маска_сети]
[шлюз] [metric метрика]] [if интерфейс]]
```

Параметры имеют следующее значение:

-f – очищает таблицу маршрутизации от всех записей, которые не являются узловыми маршрутами (маршруты с маской подсети 255.255.255.255), сетевым маршрутом замыкания на себя (маршруты с конечной точкой 127.0.0.0 и маской подсети 255.0.0.0) или

маршрутом многоадресной рассылки (маршруты с конечной точкой 224.0.0.0 и маской подсети 240.0.0.0). При использовании данного параметра совместно с одной из команд (таких, как *add*, *change* или *delete*) таблица очищается перед выполнением команды;

-p – при использовании данного параметра с командой *add* указанный маршрут добавляется в реестр и используется для инициализации таблицы *IP*-маршрутизации каждый раз при запуске протокола *TCP/IP*. По умолчанию добавленные маршруты не сохраняются при запуске протокола *TCP/IP*. При использовании параметра с командой *print* выводит на экран список постоянных маршрутов. Все другие команды игнорируют этот параметр. Постоянные маршруты хранятся в реестре по адресу *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\PersistentRoutes*;

команда – указывает команду, которая будет запущена на удаленной системе. Допустимые команды:

add – добавление маршрута;

change – изменение существующего маршрута;

delete – удаление маршрута или маршрутов;

print – печать маршрута или маршрутов.

конечная_точка – определяет конечную точку маршрута. Конечной точкой может быть сетевой *IP*-адрес (где разряды узла в сетевом адресе имеют значение 0), *IP*-адрес маршрута к узлу, или значение 0.0.0.0 для маршрута по умолчанию;

mask **маска_сети** – указывает маску сети в соответствии с точкой назначения. Маска сети может быть маской подсети, соответствующей сетевому *IP*-адресу, например, 255.255.255.255 для маршрута к узлу или 0.0.0.0. для маршрута по умолчанию. Если данный параметр пропущен, используется маска подсети 255.255.255.255. Конечная точка не может быть более точной, чем соответствующая маска подсети. Другими словами, значение разряда 1 в адресе конечной точки невозможно, если значение соответствующего разряда в маске подсети равно 0;

шлюз – указывает *IP*-адрес пересылки или следующего перехода, по которому доступен набор адресов, определенный конечной точкой и маской подсети. Для локально подключенных марш-

рутов подсети, адрес шлюза – это *IP*-адрес, назначенный интерфейсу, который подключен к подсети. Для удаленных маршрутов, которые доступны через один или несколько маршрутизаторов, адрес шлюза – непосредственно доступный *IP*-адрес ближайшего маршрутизатора;

***metric* метрика** – задает целочисленную метрику стоимости для маршрута (в пределах от 1 до 9999), которая используется при выборе в таблице маршрутизации одного из нескольких маршрутов, наиболее близко соответствующего адресу назначения пересылаемого пакета. Выбирается маршрут с наименьшей метрикой. Метрика отражает количество переходов, скорость прохождения пути, надежность пути, пропускную способность пути и средства администрирования;

***if* интерфейс** – указывает индекс интерфейса, через который доступна точка назначения. Для вывода списка интерфейсов и их соответствующих индексов используется команда *route print*. Значения индексов интерфейсов могут быть как десятичные, так и шестнадцатеричные. Перед шестнадцатеричными номерами вводится 0x. В случае, когда параметр *if* пропущен, интерфейс определяется из адреса шлюза;

/? – отображает справку в командной строке.

Рассмотрим несколько примеров использования команды *route*. Чтобы вывести на экран все содержимое таблицы *IP*-маршрутизации, введите команду:

```
route print
```

Вывод команды на экран в этом случае приблизительно следующий:

```
===========================================================
Список интерфейсов
  9 ...00 19 d2 c4 a2 19 ...... Intel(R) PRO/Wireless 3945ABG
Network Connection

  8 ...00 16 d4 5b 0a 0c ...... Broadcom NetLink (TM) Gigabit
Ethernet
  1 ........................... Software Loopback Interface 1
 14 ...00 00 00 00 00 00 00 e0  isatap.{1DD5A5E7-25E5-4FCF-
B8E1-26FE0F2AA8EB}
 16 ...00 00 00 00 00 00 00 e0  isatap.{6E9A2AE5-1D46-40C7-
8626-9746808B4001}
 11 ...02 00 54 55 4e 01 ...... Teredo Tunneling Pseudo-Inter-
face
===========================================================
```

```
IPv4 таблица маршрута
===========================================================
Активные маршруты:
Сетевой адрес           Маска сети      Адрес шлюза       Ин-
терфейс  Метрика
          0.0.0.0          0.0.0.0    192.168.0.253
192.168.0.240     30
        127.0.0.0        255.0.0.0         On-link
127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link
127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link
127.0.0.1    306
      192.168.0.0    255.255.255.0         On-link
192.168.0.240    286
    192.168.0.240  255.255.255.255         On-link
192.168.0.240    286
    192.168.0.255  255.255.255.255         On-link
192.168.0.240    286
        224.0.0.0        240.0.0.0         On-link
127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link
192.168.0.240    286
  255.255.255.255  255.255.255.255         On-link
127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link
192.168.0.240    286
===========================================================
Постоянные маршруты:
  Отсутствует
```

Выполните команду *route print*. Сколько маршрутов на вашем компьютере?

Чтобы вывести на экран маршруты из таблицы *IP*-маршрутизации, которые начинаются с 10, введите команду:

```
route print 10.*
```

Чтобы добавить маршрут по умолчанию с адресом стандартного шлюза 192.168.12.1, введите команду:

```
route add 0.0.0.0 mask 0.0.0.0 192.168.12.1
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1
```

Чтобы добавить постоянный маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

```
route -p add 10.41.0.0 mask 255.255.0.0 10.27.0.1
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1 и метрикой стоимости 7, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 metric 7
```

Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1 и использованием индекса интерфейса 0x3, введите команду:

```
route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 if 0x3
```

Чтобы удалить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0, введите команду:

```
route delete 10.41.0.0 mask 255.255.0.0
```

Чтобы удалить все маршруты из таблицы IP-маршрутизации, которые начинаются с 10., введите команду:

```
route delete 10.*
```

Чтобы изменить следующий адрес перехода для маршрута с конечной точкой 10.41.0.0 и маской подсети 255.255.0.0 с 10.27.0.1 на 10.27.0.25, введите команду:

```
route change 10.41.0.0 mask 255.255.0.0 10.27.0.25
```

МОДЕЛИРОВАНИЕ КОМПЬЮТЕРНОЙ СЕТИ

Сетевые эмуляторы

Cisco Packet Tracer – это программа-эмулятор сетевой среды, который позволяет делать работоспособные модели сети, состоящие из рабочих персональных компьютеров (ПК), серверов и различного сетевого оборудовании (коммутаторы, маршрутизаторы и т.д.), настраивать маршрутизаторы и коммутаторы, взаимодействовать между узлами сети. Для связи между узлами сети поддерживаются различные виды кабелей (витая пара, коаксиальный кабель, оптоволокно и различные их модификации), а также беспроводные соединения *Wi-Fi*. Успешно позволяет создавать даже сложные макеты сетей, проверять на работоспособность топологии.

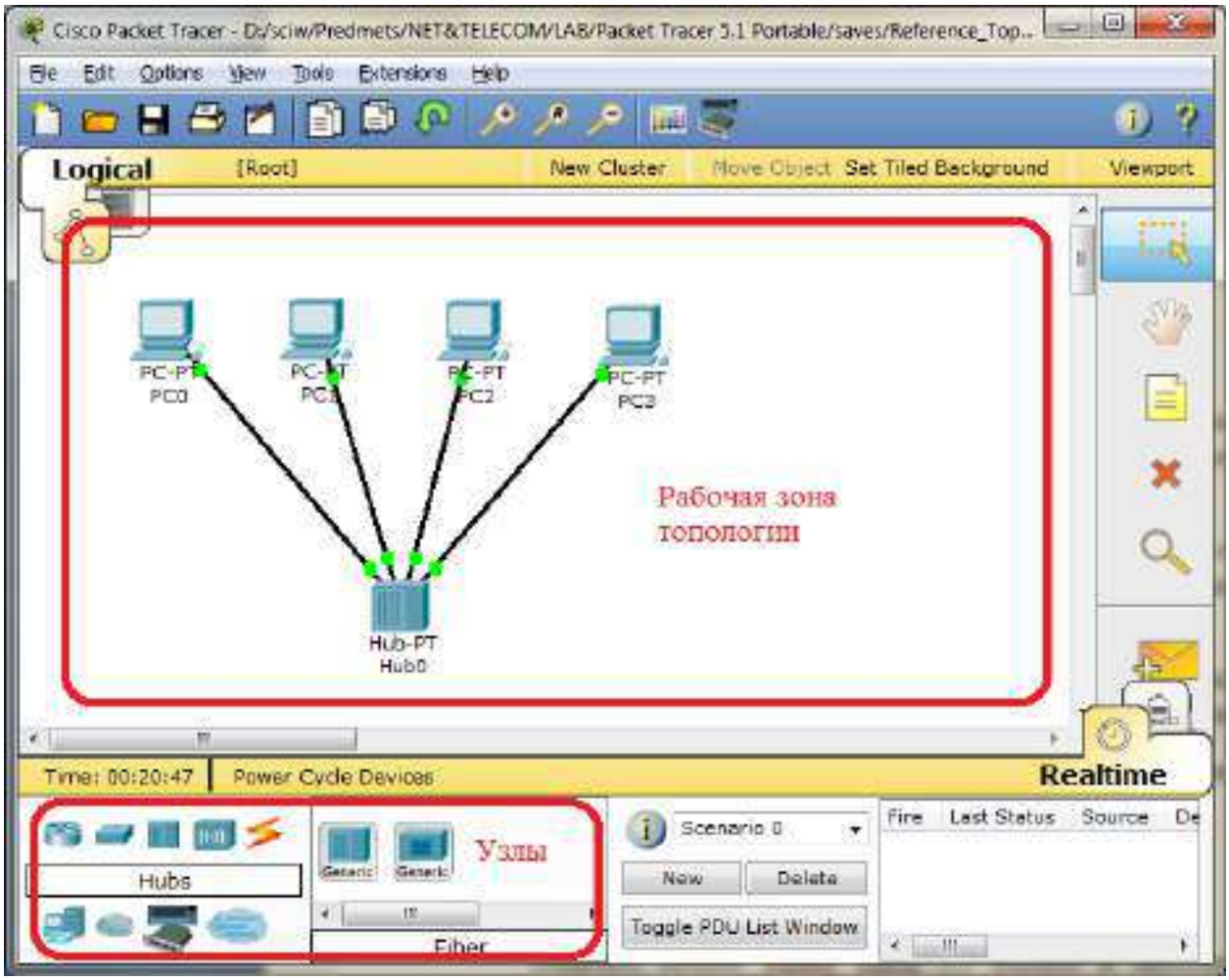


Рисунок 19 – Главное окно PacketTracer

В состав программы входит функция симуляции работы сети, которая позволяет наглядно посмотреть прохождение различных

типов пакетов по сети, изучить работу сетей в различных топологиях и с применением различного оборудования. Главное окно программы выглядит, как показано на рисунке 19.

Основными зонами в этом окне для нас являются две:

– рабочая зона топологии – содержит модель топологии конструируемой сети;

– панель узлов – содержит всевозможные сетевые устройства (ПК, хабы и т.д.), а также возможные типы связей между ними.

Панель узлов состоит из двух частей: левая часть панели содержит перечисление разделов (*Routers*, *Switches*, *Hubs*, *Wireless Devices*, *Connections*, *End Devices*, *WAN Emulation*, *Custom Made Devices*, *Multiuser Connection*), а в правой части панели для каждого раздела из левой части показаны возможные узлы сети из этого раздела. Например, если выбрать раздел *End Devices*, то будут перечислены следующие узлы: ПК, сервер, принтер, телефон, как показано на рисунке 20.

Для добавления узла выберите раздел *End Devices*, затем нажмите мышкой на иконку с изображением компьютера. Переместите курсор в зону топологии и нажмите левой кнопкой мыши на пустом месте. В этом месте будет помещен один узел сети, представляющий собой обычный ПК. Разместите недалеко еще один такой же компьютер, как показано на рисунке 21.

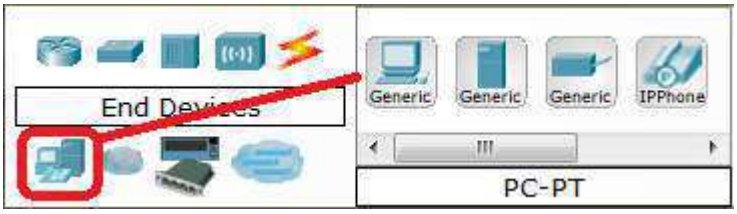


Рисунок 20 – Раздел *End Devices*

Рисунок 21 – Два узла в зоне эмуляции

Чтобы соединить узлы в сети кабелем откройте в панели узлов закладку *Connections* и выберите подключение *Copper Cross-Over* (рисунок 22).

Рисунок 22 – Закладка *Connections*

Нажмите левой кнопкой мыши на первом ПК в топологии, в появившемся ниспадающем списке выберите пункт *FastEthernet* (рисунок 23).

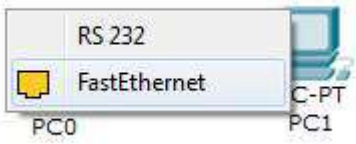


Рисунок 23 – Выбор порта подключения

Теперь подведите мышь ко второму ПК (за курсором будет тянуться линия соединения от первого ПК) и также нажмите на нем левой кнопкой мыши и выберите *FastEthernet* (рисунок 24). Теперь компьютеры подключены друг к другу.

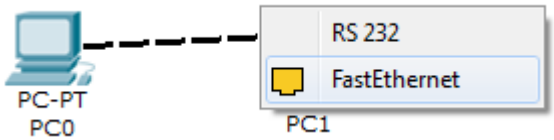


Рисунок 24 – Подключение к другому узлу

Следующим этапом для правильного функционирования сети компьютеры необходимо сконфигурировать – присвоить им различные *IP*-адреса из одной сети.

Нажмите левой кнопкой мыши на первом компьютере. Перед вами появится окно конфигурации этого компьютера. Здесь можно изменить множество параметров, среди которых *IP*-адрес и маска подсети. Они находятся на второй странице окна параметров. В левой части окна нажмите на кнопку *FastEthernet*, откроются параметры настройки сети. В поле *IP*-адреса (*IP Address*) введите адрес 192.168.1.1, а в поле маски подсети (*Subnet Mask*) введите 255.255.255.0 (рисунок 25).

После изменения окно можно закрыть стандартным крестиком в углу экрана. Изменения автоматически сохранятся. Аналогичные действия необходимо выполнить со вторым ПК, присвоить

ему *IP*-адрес 192.168.1.2 и маску 255.255.255.0.

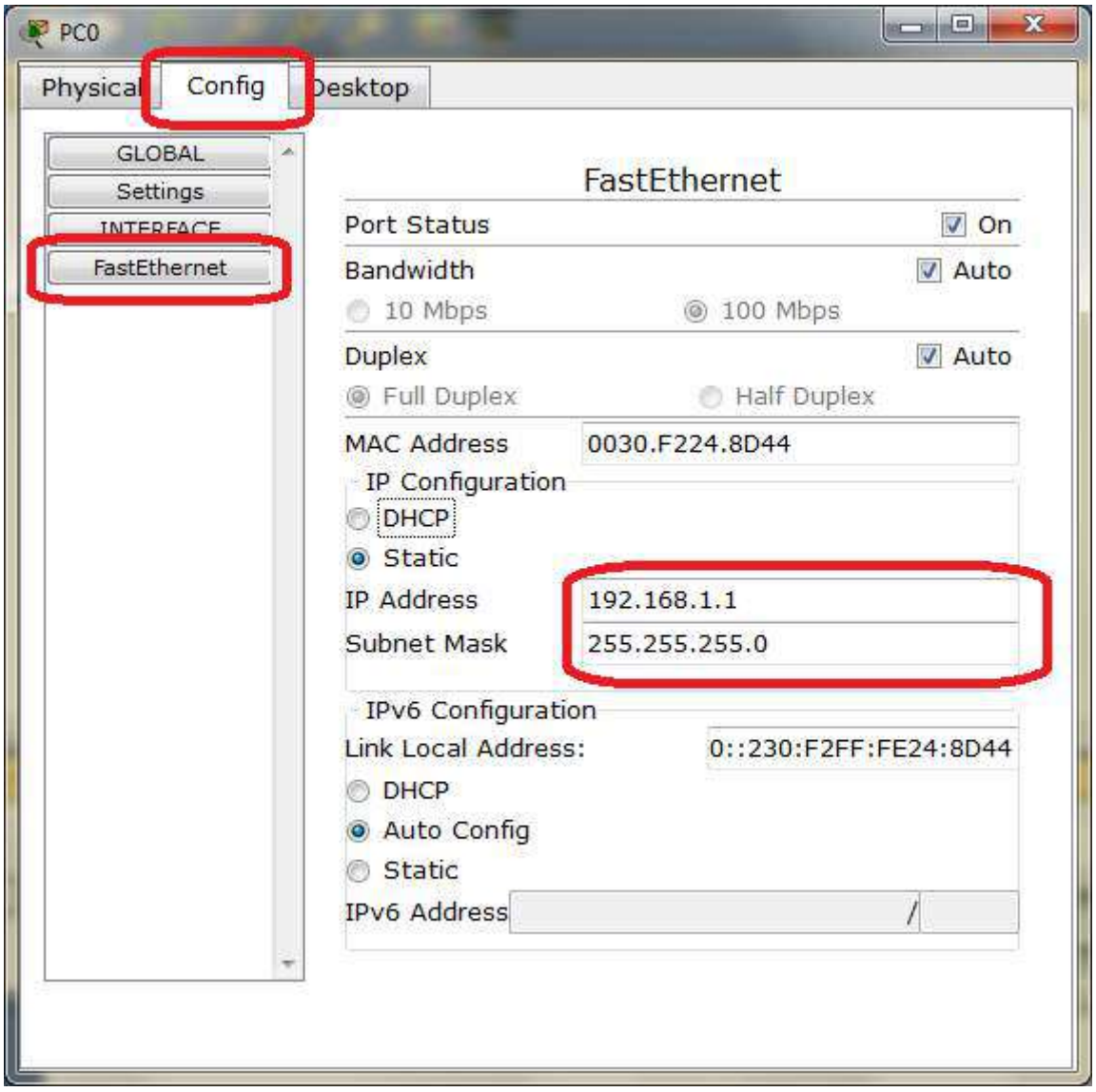


Рисунок 25 – Настройка конфигурации узла

В результате получена сеть из двух ПК, соединенных сетью Ethernet. Теперь можно симулировать прохождение пакетов по этой модели сети.

Режим симуляции сети

В правом нижнем углу программы нажмите на кнопку *Simulation Mode*, чтобы переключится в режим симуляции.

В зоне топологии появится дополнительное окно симуляции (рисунок 26). Для облегчения наблюдения над процессом симуляции в этом окне можно настроить фильтры. При стандартном функционировании сети *Ethernet* узлы обмениваются большим ко-

личеством различных пакетов. Для примера ограничимся лишь одним типом пакетов – *ICMP*. Этими пакетами обмениваются узлы при выполнении команд *ping* и *tracert* с командной строки ПК.

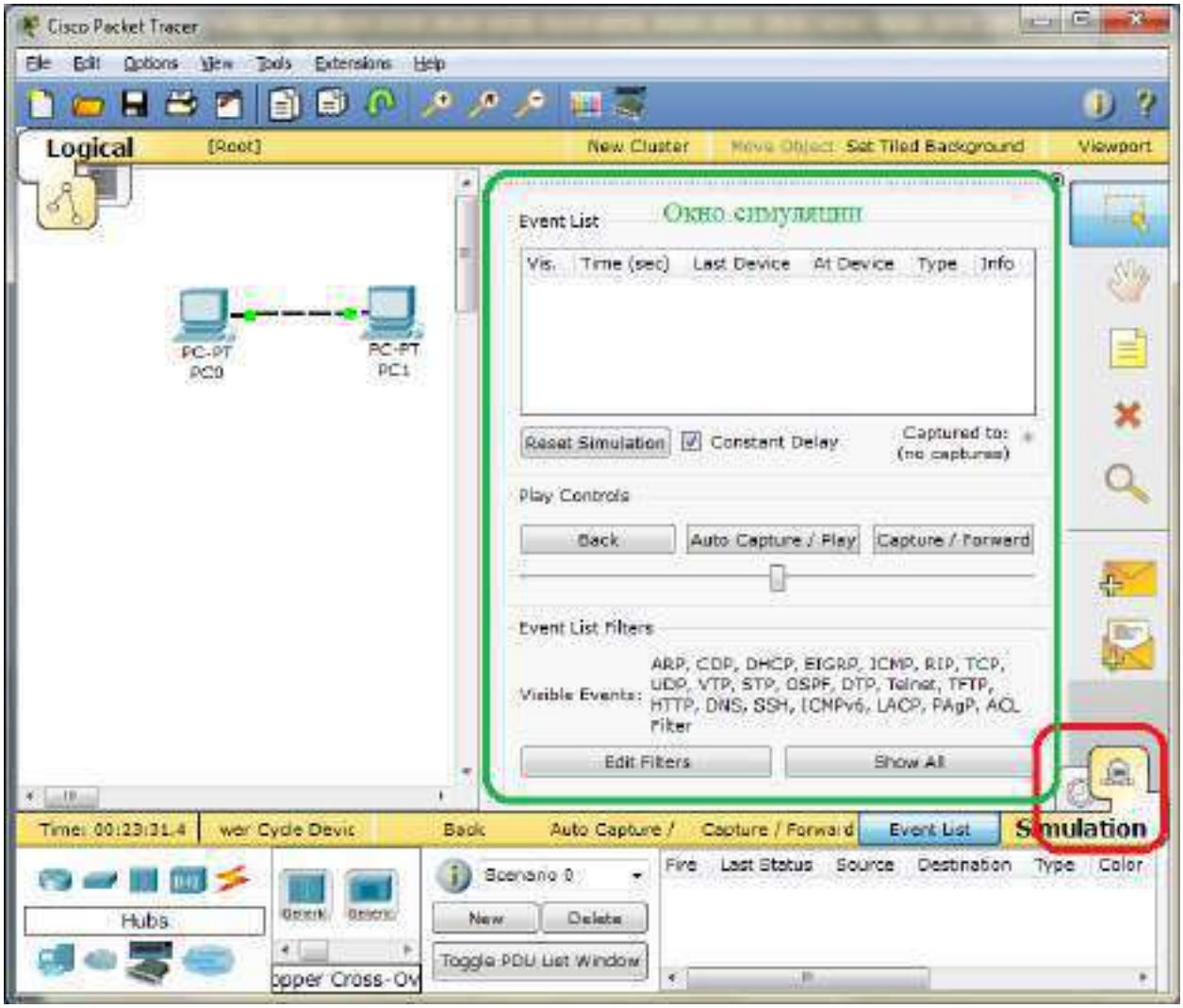


Рисунок 26 – Включение режима симуляции

Нажмите на кнопку *Edit Filters*, будет открыто окошко с множеством галочек для каждого из поддерживаемых типов пакетов, как показано на рисунке 27. Уберите все галочки, кроме *ICMP*. Это можно сделать, нажав на галочку *Show All*/*None*, чтобы убрать все, а потом выбрать *ICMP*. Нажатием левой кнопки мыши за пределами окошка настроек фильтра, его можно закрыть.

Теперь добавим в сеть пакет. Для этого нужно нажать на кнопку *Add Complex PDU* справа от окна симуляции, как показано на рисунке 28.

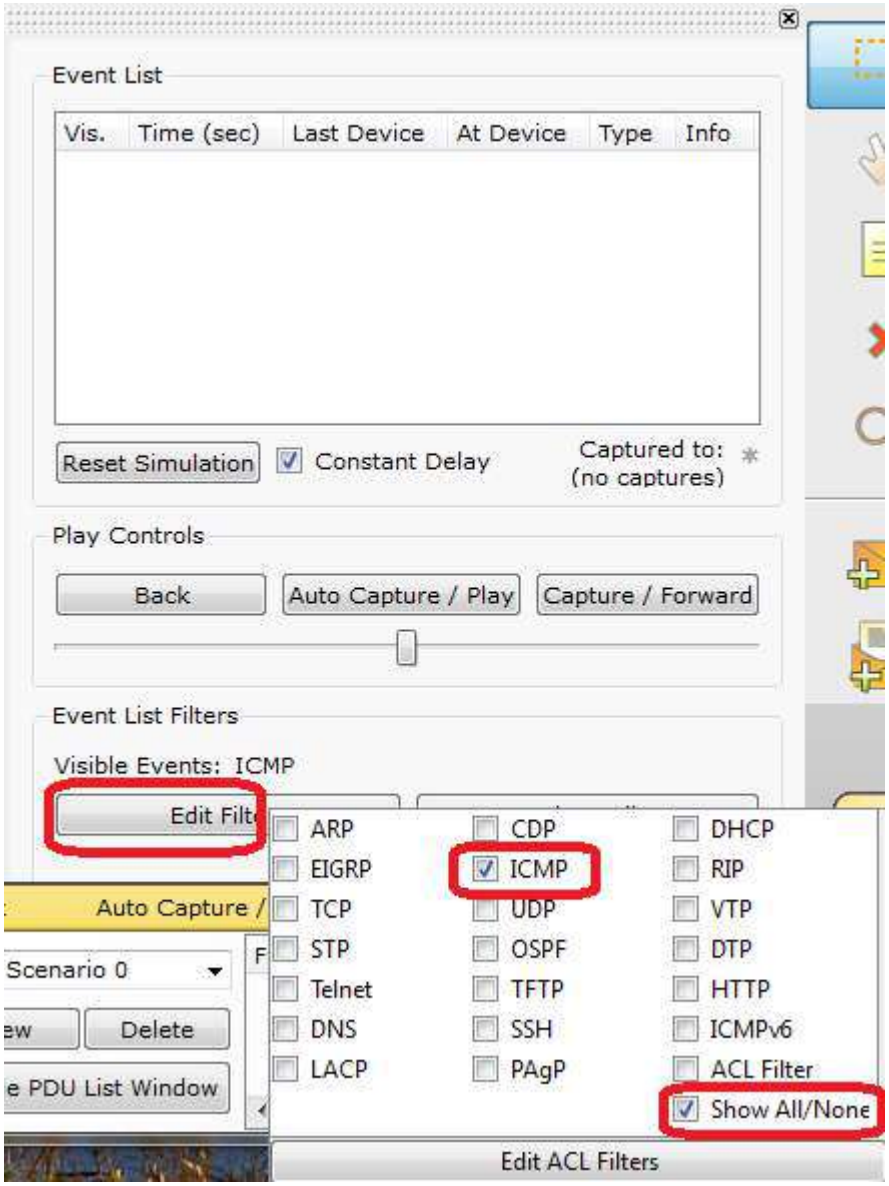


Рисунок 27 – Настройка фильтрации пакетов

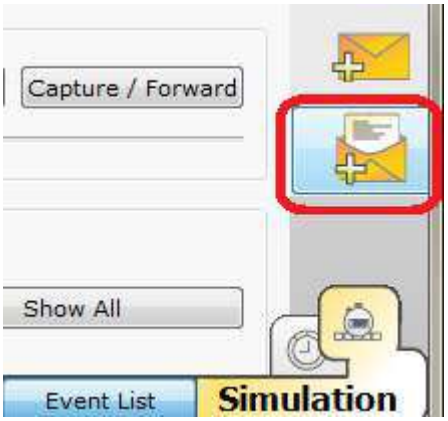


Рисунок 28 – Создание нового пакета

Курсор мыши пример вид изображения конверта. Подведите его к первому ПК (которому присваивали *IP*-адрес 192.168.1.1) и нажмите левую кнопку мыши. Появится окно с параметрами создаваемого пакета.

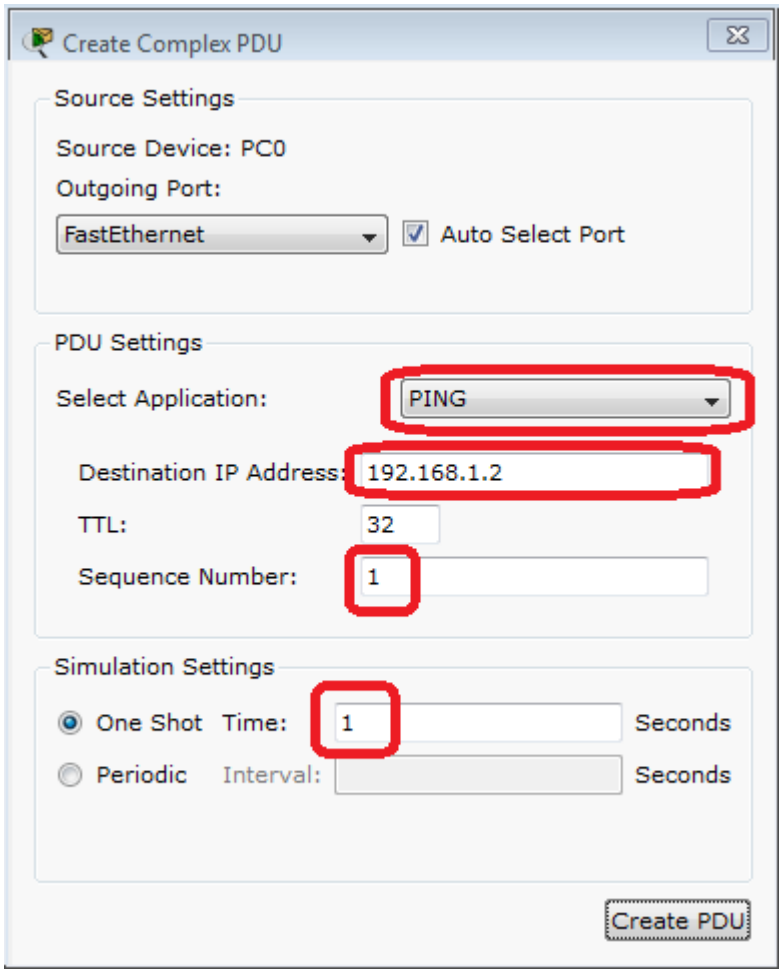


Рисунок 29 – Параметры пакета

Введите параметры аналогично рисунку 29 и нажмите кнопку *Create PDU*. Подготовка к симуляции завершена.

Для запуска симуляции нажмите кнопку *Auto Capture / Play* (рисунок 30) и проследите за движениями пакетов по сети. После того, как симуляция завершится, можно нажать кнопку *Reset Simulation* и запустить ее снова.

Кнопка *Auto Capture / Play* запускает автоматический режим симуляции, в котором можно проследить весь процесс симуляции до конца без остановок.

Для более детального изучения поведения сети удобнее воспользоваться пошаговым режимом. Для его выполнения используется кнопка *Capture / Forward*. Нажатие этой кнопки приводит к выполнению одного шага симуляции (создания пакета, передачи

пакета от одного узла к другому и т.д.).

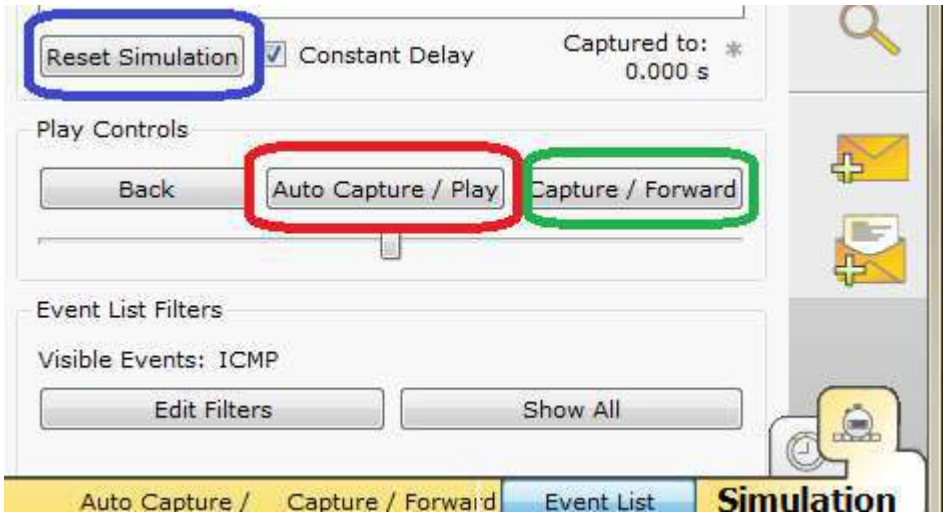


Рисунок 30 – Управление режимом симуляции

Рассмотрим подробнее процессы, которые происходят в нашей сети:

Шаг 1. Создание запроса *ICMP* на ПК1. (это запрос, который посылает ПК1 на ПК2, чтобы проверить его работу, например, командой *ping*).

Шаг 2. Передача запроса с ПК1 на ПК2.

Шаг 3. Передача ответа с ПК2 на ПК1.

Мы рассмотрели действия, которые происходят в сети при передаче *ICMP* пакетов, происходящей при выполнении команды ping с указанием *IP*-адреса соседней машины в сети.

Создайте одну модель сети, которая состоит из четырех узлов, подключенных к одному концентратору, и одну модель сети, состоящую также из четырех узлов, но с коммутатором. Проследите прохождение пакетов в обоих сетях при подаче команду *ping* от одного узла сети к другому. В чем разница в работе концентратора и коммутатора?

БАЗОВАЯ КОНФИГУРАЦИЯ СЕТЕВОГО МАРШРУТИЗАТОРА

Подключение и настройка маршрутизатора

Базовая настройка маршрутизатора является одной из основных задач при создании компьютерной сети. Рассмотрим конфигурирование устройства при примере маршрутизатора *Cisco* 1801, работающего в сети с топологией 31 и таблицей адресации 5.

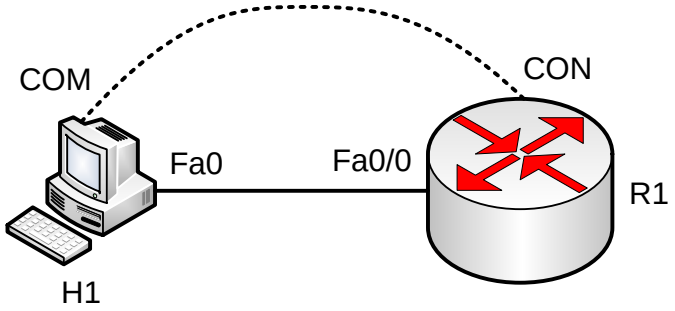


Рисунок 31 – Топология сети

Таблица 5 – Таблица адресации

Узел	Порт	IP-адрес	Маска	Шлюз	DNS
R1	Fa0/0	172.16.0.1	255.255.0.0	–	–
H1 (ПК)	FastEthernet 0	172.16.0.2	255.255.0.0	172.16.0.1	172.16.0.1

Подключите ПК (интерфейс *CiscoLAN*) перекрестным кабелем *Ethernet* к интерфейсу *Fa*0/0 маршрутизатора. Подключите последовательный интерфейс ПК (*COM*) к консольному порту (*CON*) маршрутизатора).

Задайте статический *IP*-адрес узла, используя данные из таблицы 5.

Запустите программу *HyperTerminal* и подключитесь к консоли маршрутизатора.

Войдите в конфигурационный режим и задайте имя узла:

```
Router>enable
Router#configure terminal
Router(config)#hostname R1
```

Задайте пароль ***cisco*** для консоли и включите вход с консоли:

```
R1(config)#line console 0
R1(config-line)#password cisco
R1(config-line)#login
R1(config-line)#exit
R1(config)#
```

Задайте пароль линий виртуального терминала:

```
R1(config)#line vty 0 4
R1(config-line)#password cisco
R1(config-line)#login
R1(config-line)#exit
R1(config)#
```

Задайте простой и секретный пароли привилегированного режима:

```
R1(config)#enable password cisco
R1(config)#enable secret class
R1(config)#exit
```

Секретный пароль привилегированного режима, в отличие от простого пароля, при просмотре конфигурации выводится в зашифрованном виде. Секретный пароль привилегированного режима имеет приоритет перед простым паролем привилегированного режима. После установки секретного пароля привилегированного режима простой пароль привилегированного режима больше не принимается. Можно установить только секретный пароль привилегированного режима.

Задайте заголовок новостей дня:

```
R1(config)#banner motd #Unauthorized Use Prohibited#
```

При подключении пользователя к маршрутизатору заголовок *MOTD* (*message of the day*) отображается перед запросом имени пользователя и пароля. В приведенном примере символ номера (#) используется для обозначения начала и конца сообщения. При отображении текущей конфигурации символ # превращается в ^C.

Настройте маршрутизатор таким образом, чтобы он не пытался выполнять преобразование имен узлов при использовании сервера *DNS*:

```
R1(config)#no ip domain-lookup
```

Если не выполнить эту настройку, маршрутизатор будет рассматривать команды, введенные с ошибками, как имена узлов и пытаться преобразовать их, используя сервер *DNS*. Может пройти много времени, прежде чем приглашение командной строки появится снова.

Настройте маршрутизатор таким образом, чтобы сообщения консоли не мешали введению команд:

```
R1(config)#line console 0
R1(config-line)#logging synchronous
```

Эта возможность удобна при выходе из режима конфигурации, поскольку она возвращает пользователя к командной строке

и блокирует появление сообщений в командной строке.

В командной строке привилегированного режима *EXEC* введите команду ***show running-config***. (эту команду можно сократить до ***sh run***):

```
R1#show running-config

Building configuration...
Current configuration : 605 bytes
!
hostname R1
!

*** часть выходных данных опущена ***
```

Просмотрите вывод конфигурации маршрутизатора. Есть ли зашифрованный пароль? Есть ли другие пароли? Есть ли среди других паролей зашифрованные?

Настройте интерфейс *Fast Ethernet* 0/0 на маршрутизаторе *R*1 в режиме глобальной конфигурации:

```
R1(config)#interface FastEthernet 0/0
R1(config-if)#description R1 LAN Default Gateway
R1(config-if)#ip address 172.16.0.1 255.255.0.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#exit
```

Для отображения сведений об интерфейсе введите команду ***show interfaces FastEthernet* 0/0**:

```
R1#show interfaces FastEthernet 0/0

FastEthernet0/0 is up, line protocol is up
Hardware is AmdFE, address is 000c.3076.8460 (bia
000c.3076.8460) Description: R1 LAN Default Gateway Internet
address is 172.16.0.1/16
*** часть выходных данных опущена ***
```

Просмотрите результат команды *show interfaces*. Какое состояние интерфейса *Ethernet* 0/0? Какой у него протокол канального уровня? Какой интернет-адрес? Какая инкапсуляция? На какой уровень OSI ссылается инкапсуляция?

Для проверки начальной конфигурации интерфейса можно воспользоваться командой ping.

Откройте командную строку на ПК. Для этого выберите *Start* > *Run* (Пуск > Выполнить) и введите *cmd*. Также можно выбрать

Start > *All programs* > *Accessories* > *Command Prompt* (Пуск > Все программы > Стандартные > Командная строка).

С помощью команды *ping* протестируйте подключение. Пошлите от ПК эхо-запрос интерфейсу *Fast Ethernet* маршрутизатора *R*1:

```
C:\>ping 172.16.0.1
```

Пошлите от маршрутизатора эхо-запрос на ПК:

```
R1#ping 172.16.0.2
```

Если какой-либо из пингов не успешен, найдите и устраните ошибки в конфигурации маршрутизатора или ПК, пока оба пинга не станут успешными.

Сохраните текущую конфигурацию в качестве начальной конфигурации:

```
R1#copy running-config startup-config
```

Настройка и проверка *DHCP*

Чтобы предотвратить назначение определенных адресов, их нужно исключить из пула. К таким адресам относится *IP*-адрес интерфейса *Fast Ethernet* 0/0 маршрутизатора (шлюз по умолчанию). Также будут исключены адреса в диапазоне от 172.16.0.101 до 172.16.0.254, чтобы зарезервировать их для других целей, например для назначения их серверам и принтерам, которые должны иметь фиксированные *IP*-адреса.

Для исключения адресов введите команду ***ip dhcp excluded-address***.

```
R1(config)#ip dhcp excluded-address 172.16.0.1
R1(config)#ip dhcp excluded-address 172.16.0.101 172.16.0.254
```

Для чего необходимо исключить адреса еще до того, как создан пул *DHCP*?

На маршрутизаторе настройте конфигурацию пула *DHCP* для внутренних клиентов:

```
R1(config)#ip dhcp pool INTERNAL
R1(dhcp-config)#network 172.16.0.0 255.255.0.0
R1(dhcp-config)#domain-name cisco.povt.npi-tu.ru
R1(dhcp-config)#default-router 172.16.0.1
R1(dhcp-config)#dns-server 172.16.0.1
```

Для тестирования настройки *DHCP* откройте настройки *IP*-адреса ПК и переключите на динамическую настройку. Затем откройте командную строку и введите команды ***ipconfig /release*** и ***ipconfig /renew***.

Введите в командной строке команду ***ipconfig /all***. Какой *IP*-адрес назначен узлу? Какая маска подсети? Какой шлюз по умолчанию назначен узлу? Какой *DNS*-суффикс (имя домена), зависящий от соединения, имеет узел? Какой *IP*-адрес у сервера *DHCP*? Какой *IP*-адрес у сервера *DNS*? Какой *MAC*-адрес имеет ПК?

Чтобы увидеть комбинацию *IP*-адреса и адреса аппаратного обеспечения (*MAC*), присвоенную *DHCP*-сервером, введите команду ***show ip dhcp binding*** на маршрутизаторе:

```
R1#show ip dhcp binding
```

Какой выделен адрес для ПК? Какой *MAC* адрес ПК? Через какое время заканчивается выделение *IP*-адреса?

На маршрутизаторе отобразите свойства пула *DHCP*, используя команду ***show ip dhcp pool***:

```
R1#show ip dhcp pool
```

Сколько адресов было выделено? Что означает ***Current Index*** в выходных данных команды?

Сохранение конфигурации на внешнем сервере

Запустите на ПК *Open TFTP Server*. Вы увидите текст, свидетельствующий, что сервер запущен. Сервер работает с каталогом документов пользователя на запущенном ПК.

Из сеанса *HyperTerminal* на маршрутизаторе *R*1 введите команду ***copy running-config tftp***, чтобы начать загрузку *TFTP* на *TFTP*-сервер. Отвечайте на запросы, как показано ниже. Имя файла назначения по умолчанию имеет следующий формат: имя устройства («*r*1»), тире и «*confg*». При успешном выполнении команды в окне терминала маршрутизатора отобразится сообщение, содержащее восклицательные знаки и количество скопированных байт.

```
R1#copy running-config tftp
Address or name of remote host []? 172.16.0.2
Destination filename [r1-confg]? <ENTER>
!!
1078 bytes copied in 1.188 secs (907 bytes/sec)
R1#
```

Откройте каталог пользователя на ПК, найдите файл *r*1-*confg*, просмотрите его содержимое. Что содержится в данном файле? В каком виде представлено?

Измените текущую конфигурацию *R*1. В режиме конфигурации смените имя маршрутизатора на *Router*. После этого приглашение консоли командной строки *iOS* должно смениться.

```
R1>enable
R1#configure terminal
R1(config)#hostname Router
Router(config)#end
```

Загрузите конфигурационный файл с *TFTP*-сервера с помощью команды ***copy tftp startup-config***. Отвечайте на запросы, как показано ниже. При успешном выполнении команды в окне терминала маршрутизатора отобразится сообщение, содержащее восклицательные знаки и количество скопированных байт.

```
Router#copy tftp startup-config
Address or name of remote host [172.16.0.2]? <ENTER>
Source filename [r1-confg]? <ENTER>
Destination filename [startup-config]? <ENTER>
Accessing tftp://172.16.0.2/r1-confg...
Loading r1-confg from 172.16.0.2 (via FastEthernet0/0):
!
[OK - 1078 bytes]
[OK]
1078 bytes copied in 12.780 secs (84 bytes/sec)
Router#
*Feb 17 02:18:33.551: %SYS-5-CONFIG_NV_I: Nonvolatile storage
configured from tftp://172.16.0.2/r1-confg by console
Router#
```

Просмотрите настройки в *NVRAM*, чтобы проверить, правильно ли выполнено преобразование, с помощью команды ***show startup-config*.**

Перезагрузите маршрутизатор и выберите ***No*** в ответ на запрос о сохранении измененной конфигурации.

```
Router#reload
```

Проследите процесс загрузки маршрутизатора до конца. После завершения загрузки имя должно быть *R*1. Проверьте *ping* между ПК и маршрутизатором, чтобы убедиться, что интерфейсы работают. Просмотрите текущую конфигурацию (***show running-config***), чтобы убедиться, что конфигурация загружена успешно.

Внутриполосное подключение к маршрутизатору

Запустите на ПК в командной строке клиентское приложение *telnet* с указанием IP-адреса маршрутизатора:

```
C:\>telnet.exe 172.16.0.1
```

Вы должны увидеть сообщение при входе на маршрутизатор, которое ранее было настроено командой ***banner motd*** и запрос на ввод пароля. При запросе пароля введите ***cisco*** (при вводе символа не отображаются!). Вы должны попасть в командную строку, аналогичную подключению к консоли.

Просмотрите текущую конфигурацию маршрутизатора. Отличается ли чем-нибудь интерфейс *CLI* через консоль и через *Telnet*? В каких случаях используется внутриполосное управление маршрутизатором? В каких случаях используется внеполосное управление маршрутизатором?

Для отключения от маршрутизатора по протоколу *Telnet*, наберите команду ***exit*** в *CLI*. Программа клиента *telnet* будет закрыта.

Удаление конфигурации оборудованиям

Перейдите в привилегированный режим *EXEC* с помощью команды ***enable***.

```
R1>enable
```

В привилегированном режиме *EXEC* введите команду ***erase startup-config***.

```
Router#erase startup-config
```

После этого отобразится следующее сообщение:

```
Erasing the nvram filesystem will remove all files! Continue?
[confirm]
```

Для подтверждения нажмите клавишу **ВВОД**.

После этого отобразится:

```
Erase of nvram: complete
```

В привилегированном режиме *EXEC* введите команду ***reload***.

```
R1#reload
```

После этого отобразится следующее сообщение:

```
System configuration has been modified. Save? [yes/no]:
```

Введите ***n***, а затем нажмите клавишу **ВВОД**.

Отобразится следующее сообщение:

```
Proceed with reload? [confirm]
```

Для подтверждения нажмите клавишу **ВВОД**.

Сначала отобразится следующий ответ:

```
Reload requested by console.
```

Проследите процесс начальной загрузки маршрутизатора. Какие этапы проходит маршрутизатор при включении?

После перезагрузки маршрутизатора отобразится следующее сообщение:

```
Would you like to enter the initial configuration dialog?
[yes/no]:
```

Введите ***n***, а затем нажмите клавишу **ВВОД**.

После этого отобразится следующее сообщение:

```
Press RETURN to get started!
```

Нажмите клавишу **ВВОД.**

АНАЛИЗ СЕТЕВОГО ТРАФИКА

Wireshark – это программа для анализа протоколов (анализатор пакетов), которая используется для поиска и устранения неполадок в сети, анализа, разработки программного обеспечения и протоколов, а также обучения. По мере движения потоков данных по сети анализатор перехватывает каждый протокольный блок данных (*PDU*), после чего расшифровывает или анализирует его содержание согласно соответствующему документу *RFC* или другим спецификациям.

Сбор и анализ данных протокола *ICMP*

Рассмотрим пример, как можно отправить эхо-запрос с помощью команды *ping* на другой ПК в локальной сети, перехватить *ICMP*-запросы и отклики в программе *Wireshark*, найти необходимую информацию в собранных кадрах. Этот анализ поможет понять, как используются заголовки пакетов для передачи данных по месту назначения.

Первым шагом необходимо узнать *IP*-адрес компьютера и физический адрес сетевого адаптера, который называется *MAC*-адресом.

Откройте окно командной строки, введите команду ***ipconfig /all*** и нажмите клавишу ВВОД. Пример вывода команды *ipconfig* показан на рисунке 32. Запишите *IP*-адрес интерфейса ПК и *MAC*-адрес.

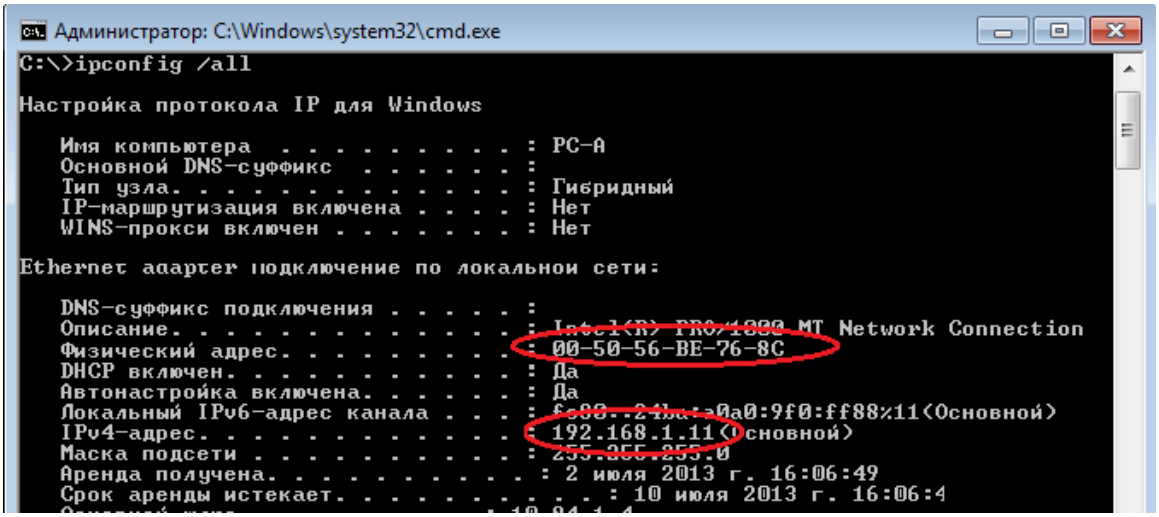


Рисунок 32 – Определение адреса компьютера

Запустите программу *Wireshark*, нажмите на параметр ***Interface list*** (Список интерфейсов, рисунок 33). Список интерфейсов можно также открыть, нажав на значок первого интерфейса в ряду значков.

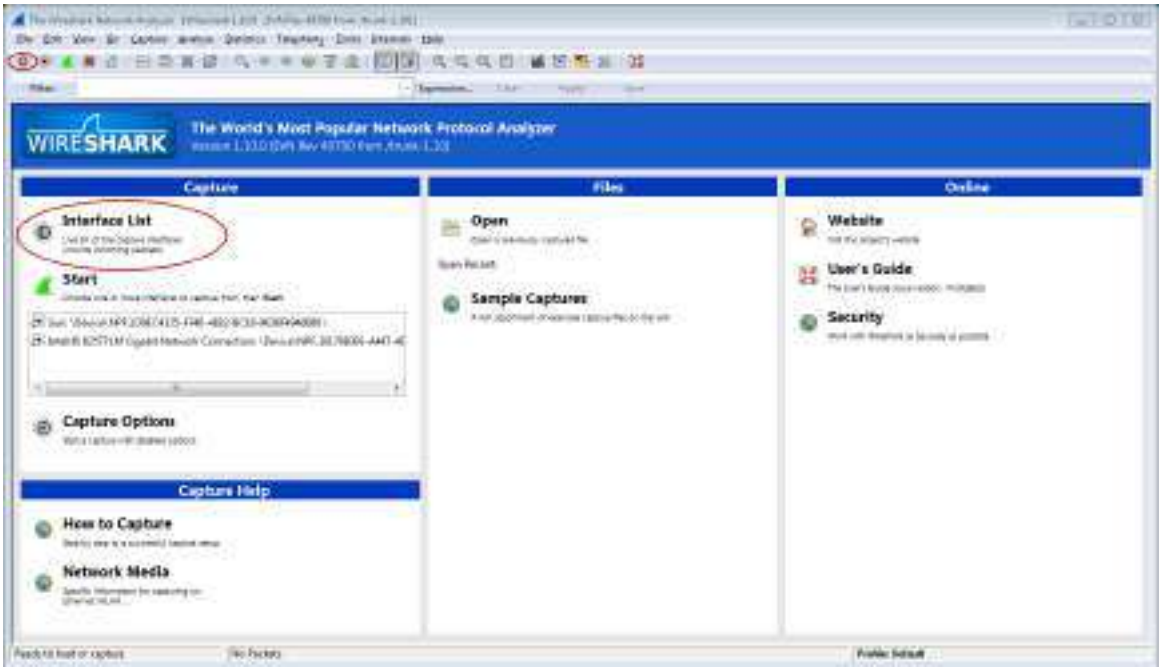


Рисунок 33 – Окно программы Wireshark

В окне ***Capture Interfaces*** (Перехват интерфейсов) установите флажок рядом с интерфейсом, подключённым к вашей локальной сети (рисунок 34).

Рисунок 34 – Выбор интерфейсов

Если перечислено несколько интерфейсов и вы не уверены в том, какой из них нужно выбрать, нажмите кнопку ***Details*** (Подробнее) и откройте вкладку **802.3 (*Ethernet*).** Убедитесь в том, что *MAC*-адрес соответствует нужному интерфейсу (рисунок 35).

После этого нажмите кнопку ***Start*** (Начать), чтобы начать перехват данных (рисунок 36).

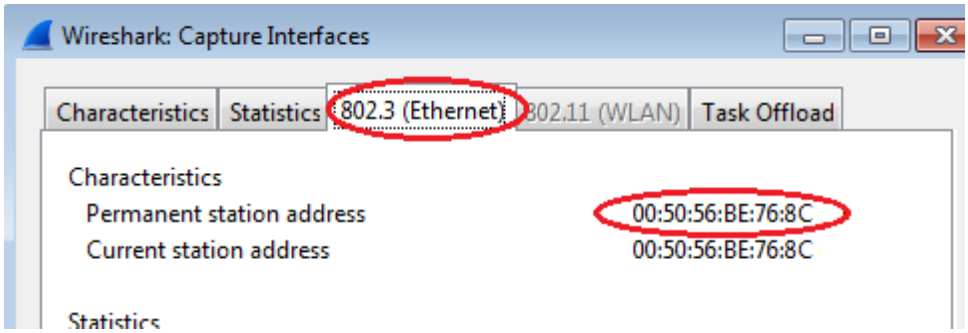


Рисунок 35 – Просмотр информации об интерфейсах

Рисунок 36 – Запуск захвата трафика

В верхней части окна программы *Wireshark* начнёт прокручиваться информация. Строки данных выделяются различными цветами в зависимости от протокола (рисунок 37).

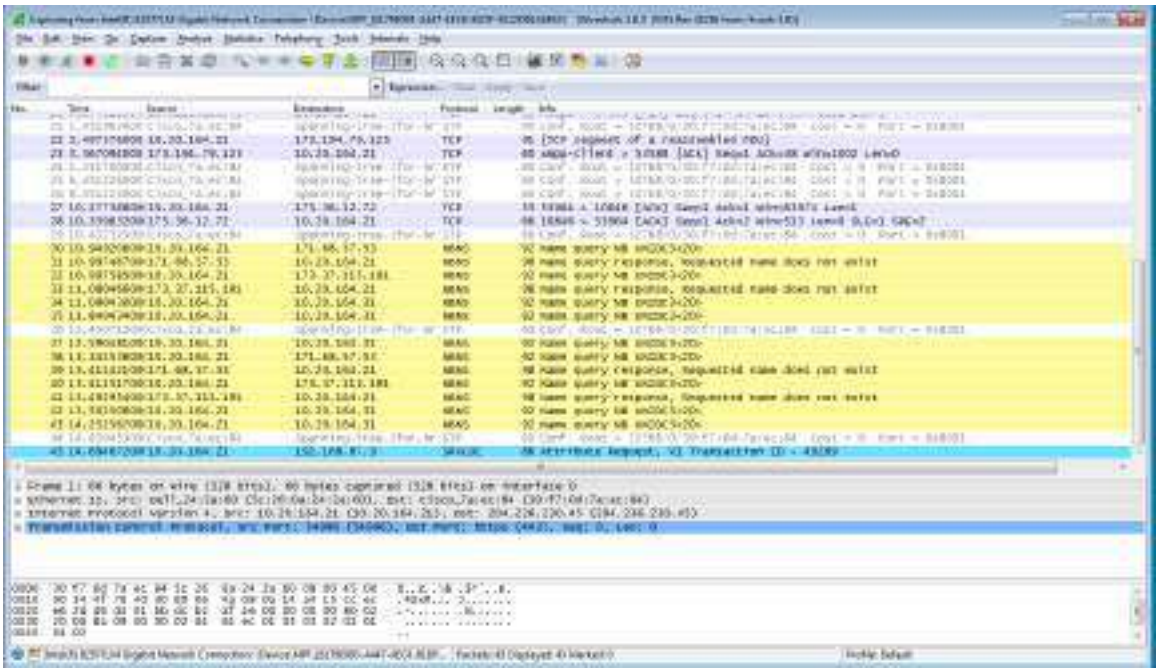


Рисунок 37 – Захваченные сетевые пакеты

Информация может прокручиваться очень быстро в зависимости от интенсивности сетевого обмена. Чтобы облегчить просмотр и работу с данными, собранными программой *Wireshark*, можно применить фильтр. Чтобы вывести на экран только протокольные блоки данных *ICMP* (эхо-запрос с помощью команды *ping*), в поле

фильтра в верхней части окна программы *Wireshark* (рисунок 38) введите ***icmp*** и нажмите клавишу ВВОД или кнопку ***Apply*** (Применить).

Рисунок 38 – Окно фильтрации

После этого все данные в верхнем окне исчезнут, однако перехват трафика в интерфейсе продолжится. Откройте окно командной строки и отправьте эхо-запрос с помощью команды *ping* на *IP*-адрес другого компьютера. В верхней части окна программы *Wireshark* появятся данные (рисунок 39).

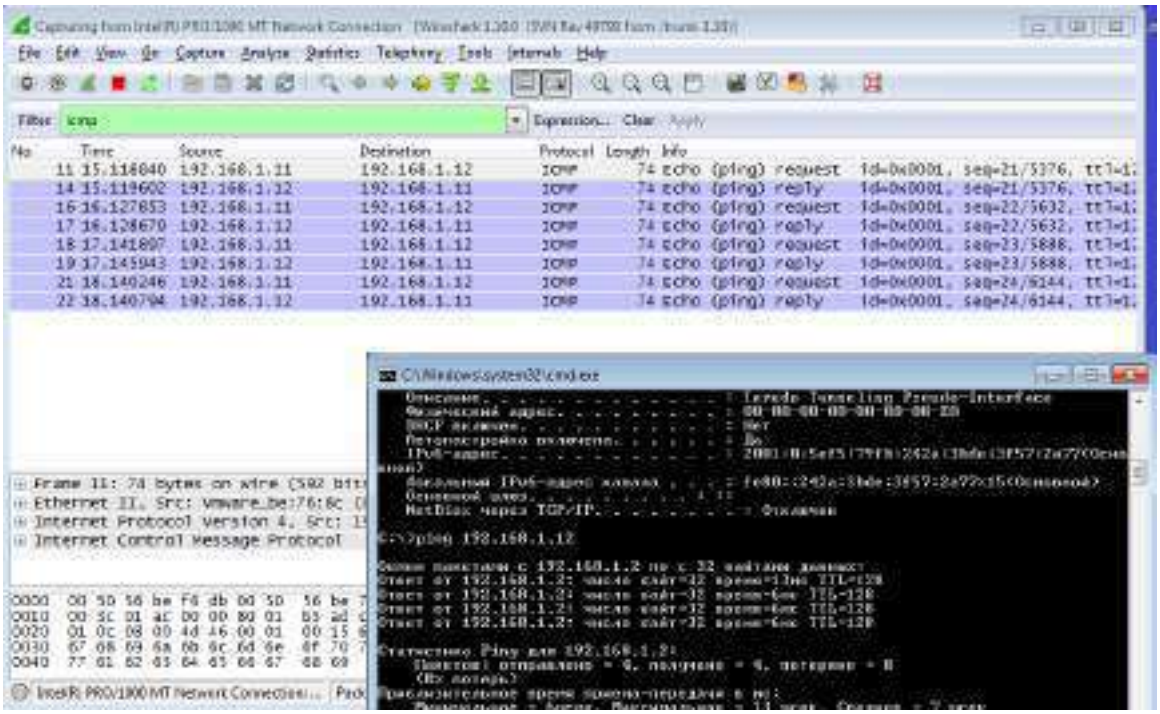


Рисунок 39 – Захваченные пакеты ICMP

Остановите перехват данных, нажав на значок ***Stop Capture*** (Остановить перехват), как показано на рисунке 40.

Изучим полученные пакеты протокола *ICMP*. Программа *Wireshark* отображает данные в трёх разделах (рисунок 41):

1) В верхнем разделе отображается список полученных кадров *PDU* со сводной информацией об *IP*-пакетах.

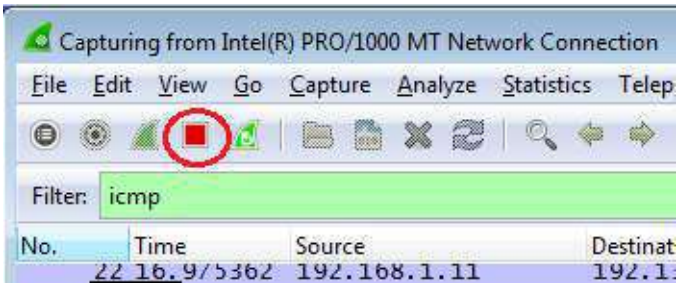


Рисунок 40 – Остановка захвата трафика

2) В среднем разделе приводится информация о *PDU* для кадра, выбранного в верхнем разделе экрана, и деление кадра *PDU* на слои протоколов.

3) В нижнем разделе показываются необработанные данные каждого уровня, как в шестнадцатеричном, так и десятичном форматах.

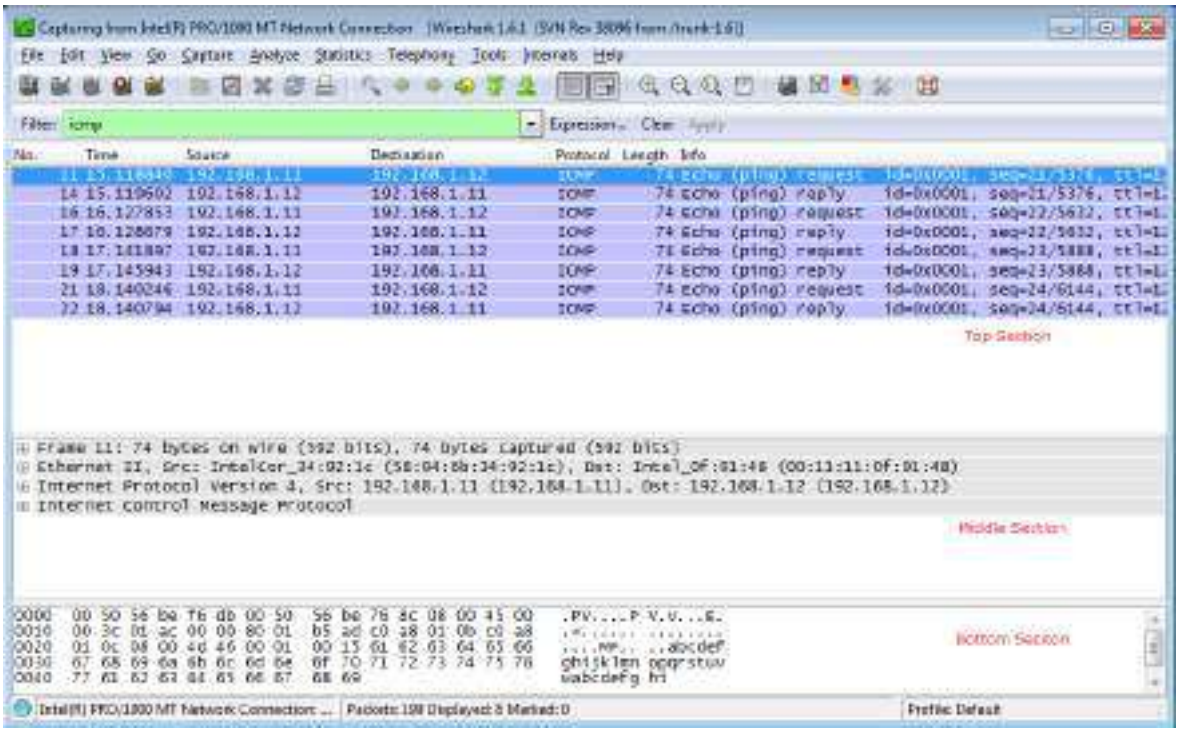


Рисунок 41 – Просмотр информации пакета

Выберите *PDU*-кадры первого запроса *ICMP* в верхнем разделе окна программы *Wireshark* (рисунок 42). В столбце *Source* (Источник) указывается *IP*-адрес вашего компьютера, а в столбце «*Destination*» (Назначение) – *IP*-адрес ПК, которому вы отправили эхо-запрос с помощью команды *ping*. Не меняя выбор *PDU*-кадра в верхнем разделе программы, перейдите в средний раздел. Нажмите на символ «+» слева от строки «*Ethernet* II», чтобы увидеть *MAC*-адреса источника и назначения (рисунок 43).

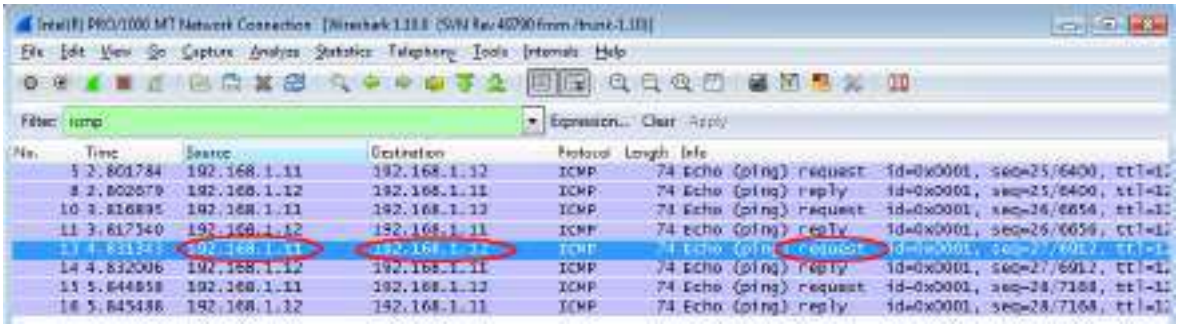


Рисунок 42 – Выбор нужного пакета *ICMP*

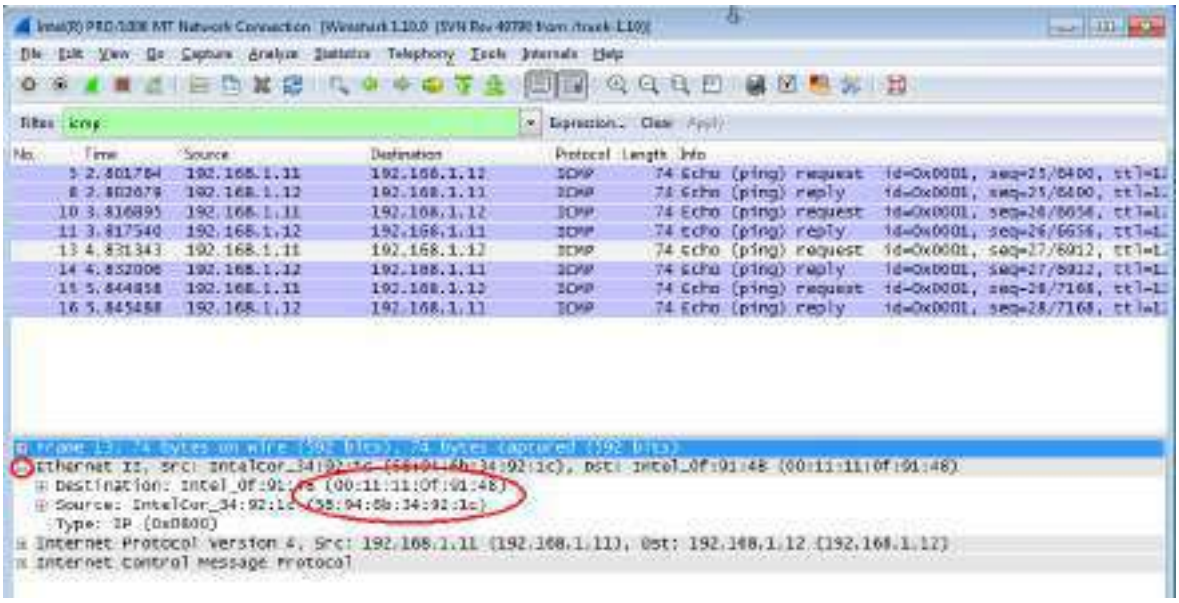


Рисунок 43 – Определение *MAC*-адресов

MAC-адрес источника должен совпадать с интерфейсом вашего компьютера, а *MAC*-адрес назначения – с *MAC*-адресом другого компьютера.

Как ваш ПК определил *MAC*-адрес ПК, на который был отправлен эхо-запрос с помощью команды *ping*?

Анализ обмена сообщениями *ARP*

Начните захват пакетов в программе *Wireshark*. С помощью фильтра отобразите только пакеты *ARP*.

Очистите *ARP*-кэш, набрав в командной строке команду:

```
arp –d *
```

Убедитесь в том, что *ARP*-кэш очищен:

```
arp -a
```

Отправьте эхо-запрос с помощью команды *ping* на шлюз по умолчанию с помощью команды:

```
ping 192.168.100.1.
```

После отправки эхо-запроса на шлюз по умолчанию остановите захват данных программой *Wireshark*.

В захваченных данных найдите сообщения *ARP* в панели сведений о пакетах. Найдите первый пакет, он является *ARP*-запросом (рисунок 44).

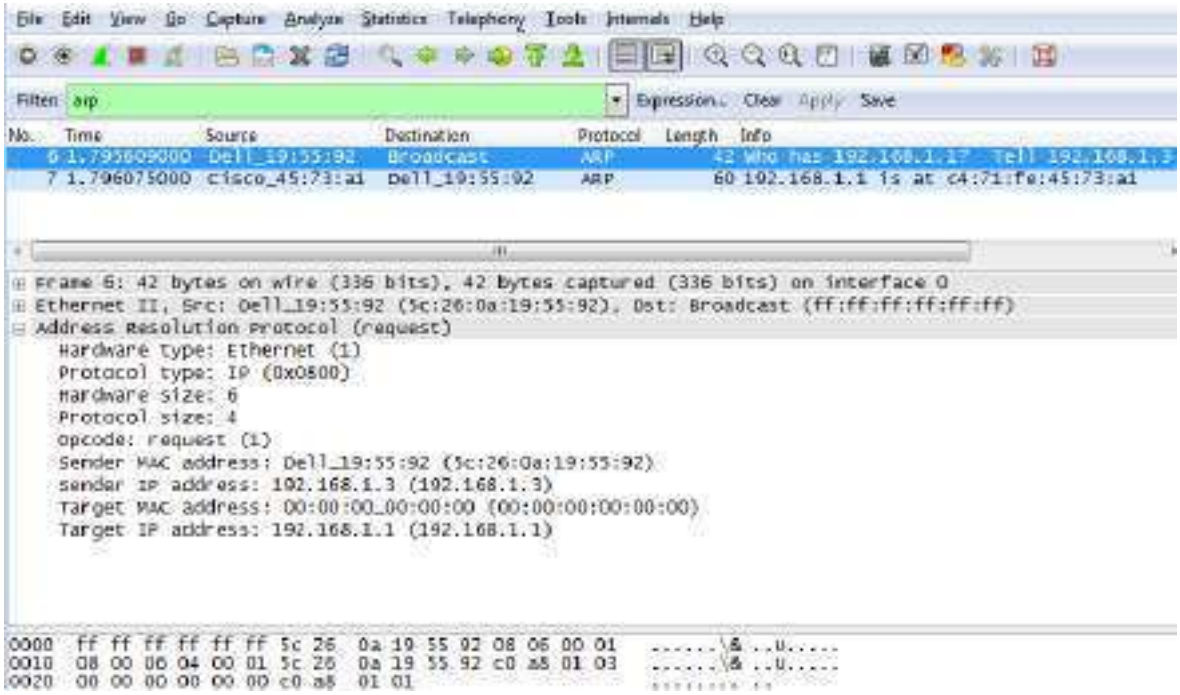


Рисунок 44 – Захват *ARP*-запроса

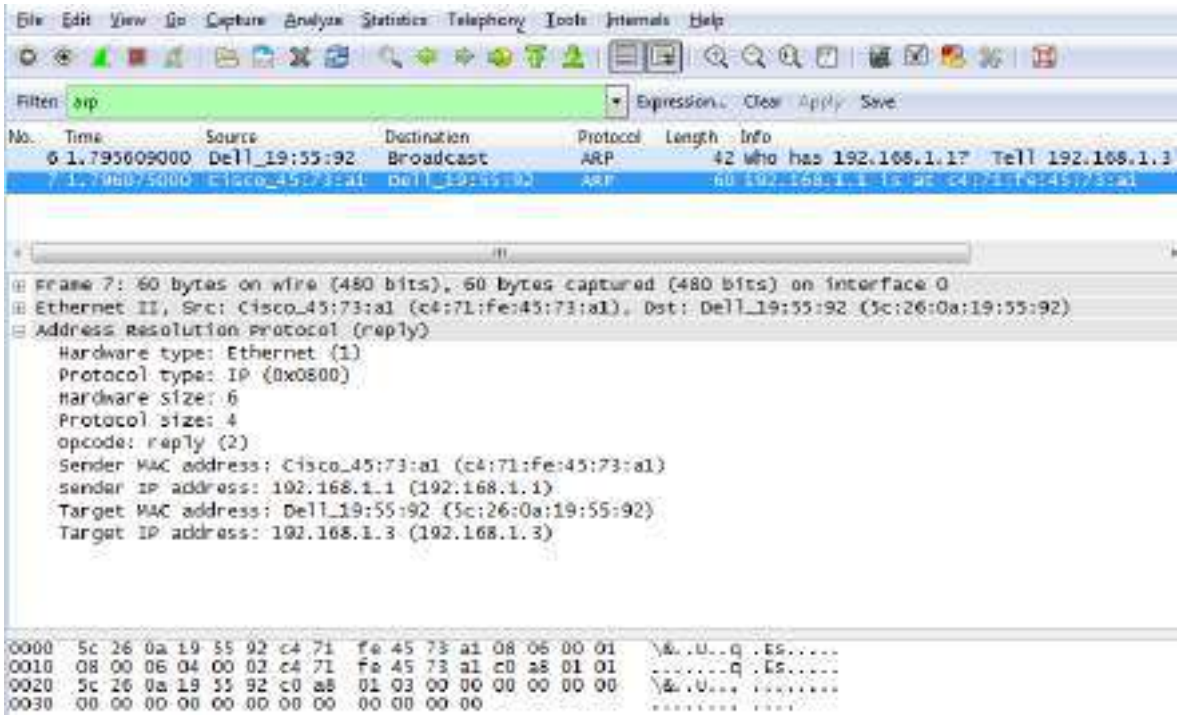


Рисунок 45 – Захват *ARP*-ответа

У *ARP*-запроса *MAC*-адрес отправителя является адресом вашего компьютера, а *MAC*-адрес получателя нулевой. Этот пакет является широковещательным, поскольку компьютер-отправитель не знает адреса получателя. Найдите второй пакет, он будет являться *ARP*-ответом (рисунок 45).

У *ARP*-ответа *MAC*-адресом отправителя является реальный адрес второго компьютера, а *MAC*-адресом получается – адрес первого компьютера, который отправлял *ARP*-запрос. Этот пакет уже не является широковещательным, он направляется первому компьютеру, который отправлял запрос.

Таким образом первый компьютер узнал *MAC*-адрес второго компьютера. Эти данные мы перехватили с помощью программы *Wireshark*.

Перехват установки сессии TCP

Запустите программу *Wireshark*. Выберите сетевой интерфейс, который будете использовать для захвата. Нажмите кнопку *Start* (Старт), чтобы начать захват данных.

Откройте в любом браузере веб-сайт, например *www.npi-tu.ru*. Сверните окно браузера и вернитесь в программу *Wireshark*. Остановите процесс захвата данных. Вы увидите захваченный трафик (рисунок 46). Найдите столбцы ***Source*** (Источник), ***Destination*** (Назначение) и ***Protocol*** (Протокол).

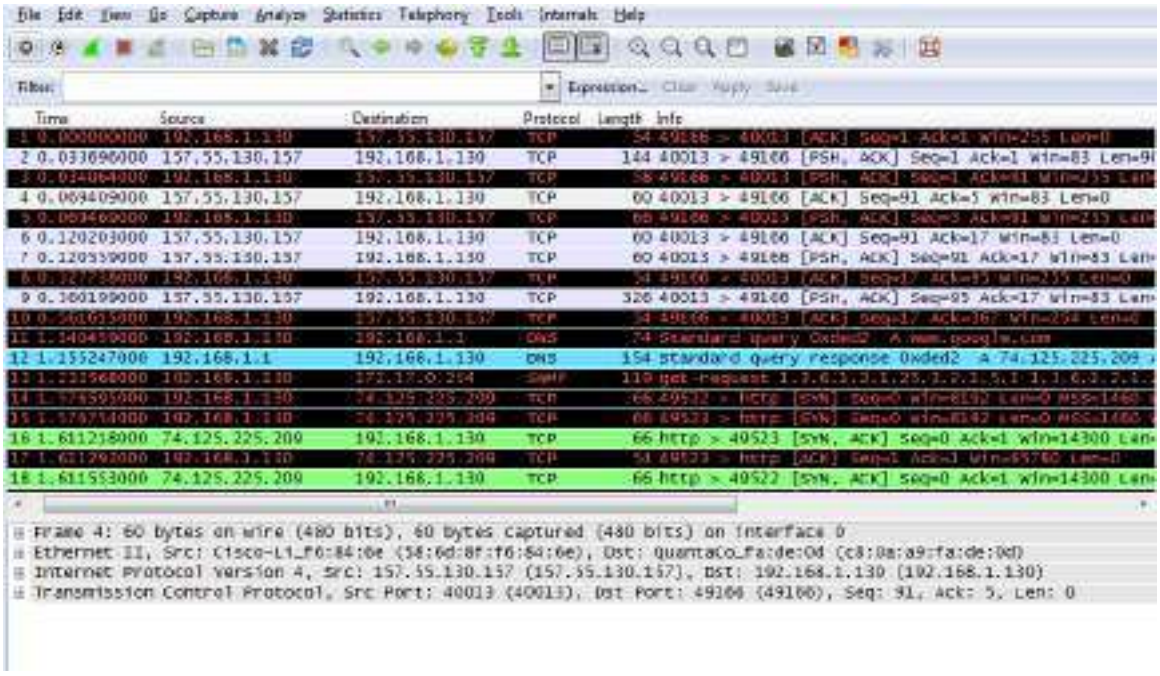


Рисунок 46 – Захват трафика при обращении к веб-сайту

В захваченных данных вы сможете увидеть весь процесс, включая протокол разрешения адресов (*ARP*), службу доменных имен (*DNS*) и трёхстороннее рукопожатие *TCP*, либо только часть этих данных.

Определите из рисунка 46 *IP*-адрес *DNS*-сервера, запрошенного компьютером.

Чтобы просмотреть только пакеты, связанные с *TCP*-соединением, воспользуйтесь фильтрами программы *Wireshark*. В поле фильтра программы *Wireshark* введите ***tcp*** и нажмите клавишу ВВОД (рисунок 47).

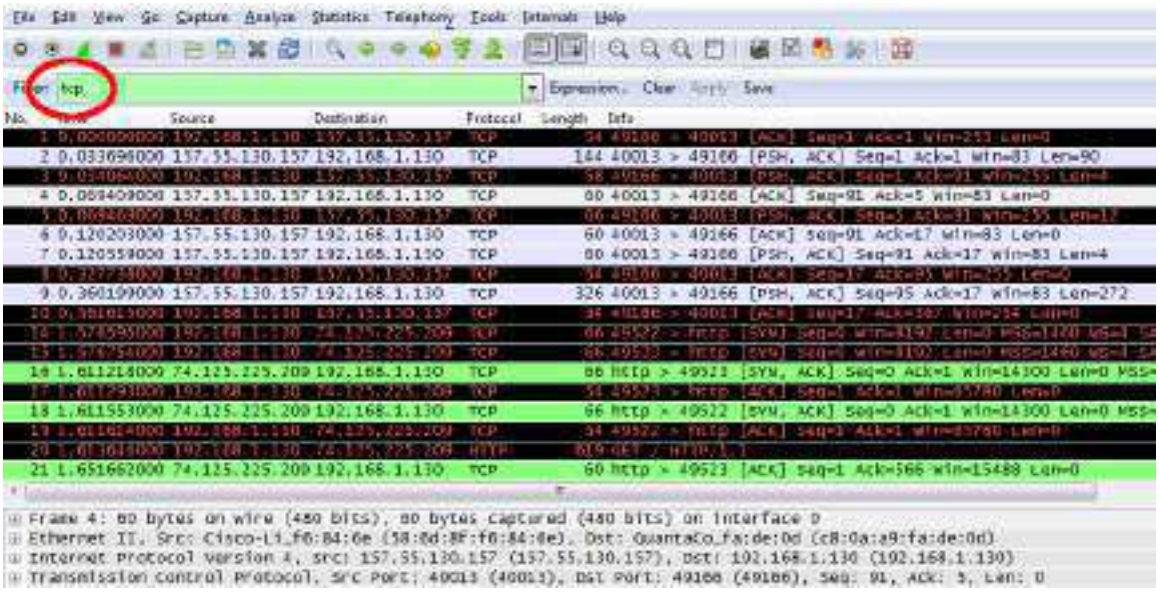


Рисунок 47 – Фильтрация протокола *TCP*

На панели списка пакетов (верхний раздел основного окна) выберите кадр, первый кадр должен иметь флаги [*SYN*] (рисунок 48). После этого будет выделена строка и отображена зашифрованная информация из пакета в двух нижних панелях. На панели нажмите на значок «+» слева от строки *Transmission Control Protocol* (Протокол управления передачей данных), чтобы увидеть подробную информацию о *TCP*. Слева от флажков нажмите на значок «+», чтобы увидеть подробную информацию о флагах *TCP*. Чтобы выбрать следующий кадр установки соединения, в меню программы *Wireshark* выберите параметр ***Go*** (Перейти), а затем ***Next Packet In Conversation*** (Следующий пакет коммуникации). Это ответ веб-сервера на исходный запрос начала сеанса (рисунок 49).

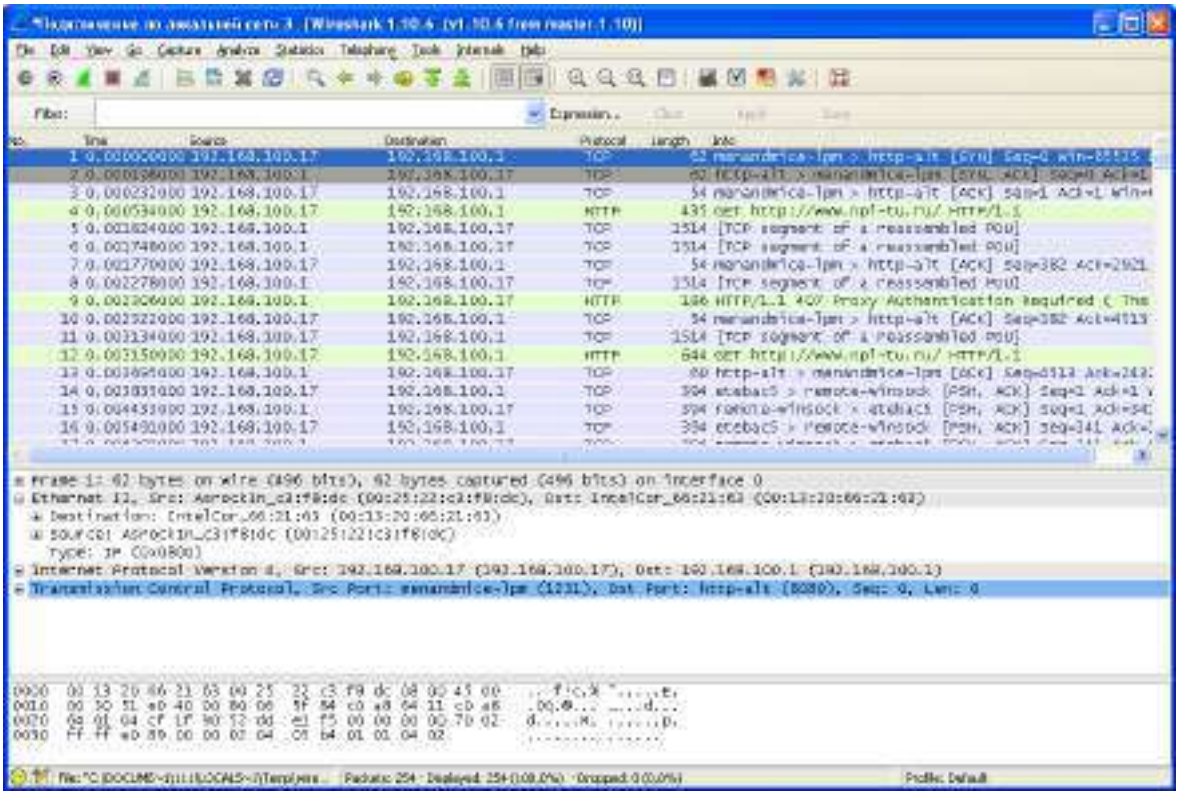


Рисунок 48 – Первый пакет установки связи *TCP*

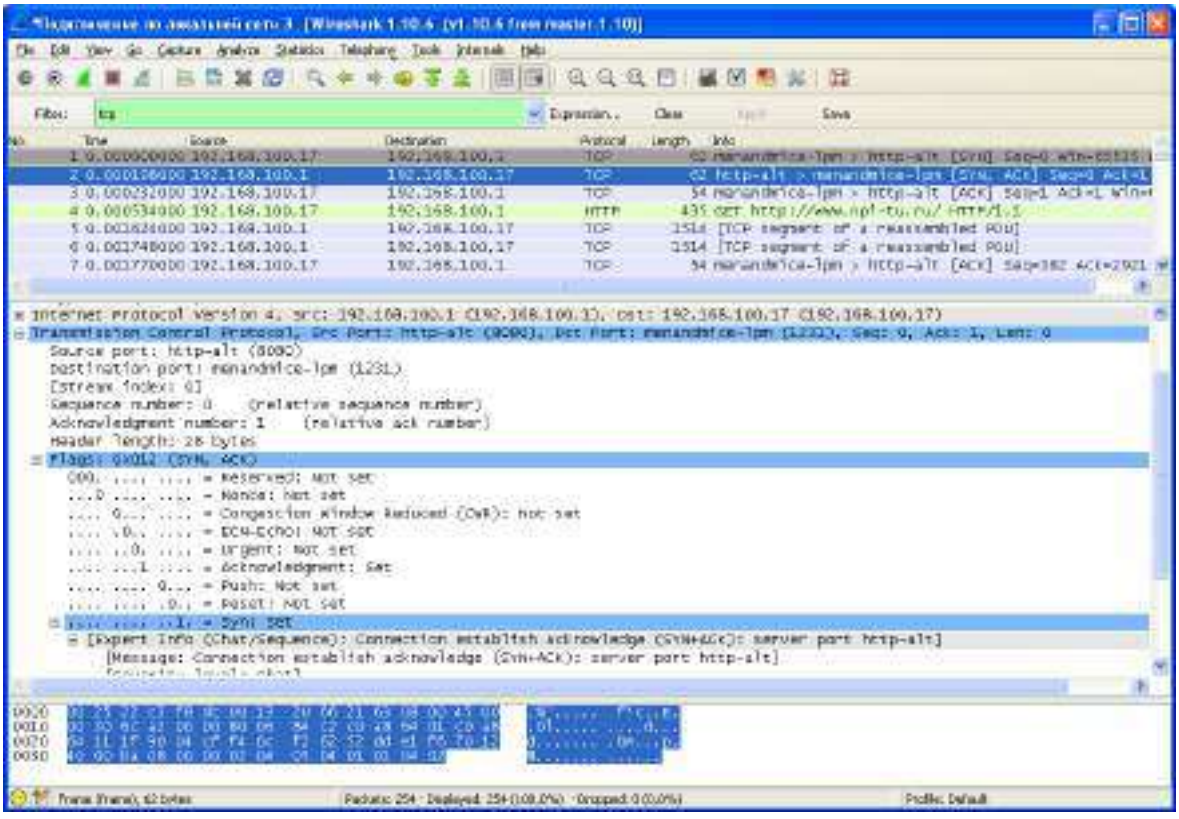


Рисунок 49 – Второй пакет установки связи *TCP*

Найдите и изучите третий пакет установки соединения. Какие установлены флаги *TCP*?

НАСТРОЙКА ПРОТОКОЛА МАРШРУТИЗАЦИИ *RIP*

Тестовая сеть

RIP – это один из самых распространенных и пользующихся широкой поддержкой протоколов маршрутизации в сетевой индустрии. Рассмотрим настройку *RIP* на примере небольшой сети, состоящей из трех подсетей. Топология примерной сети представлена на рисунке 50, а таблица адресации в таблице 6.

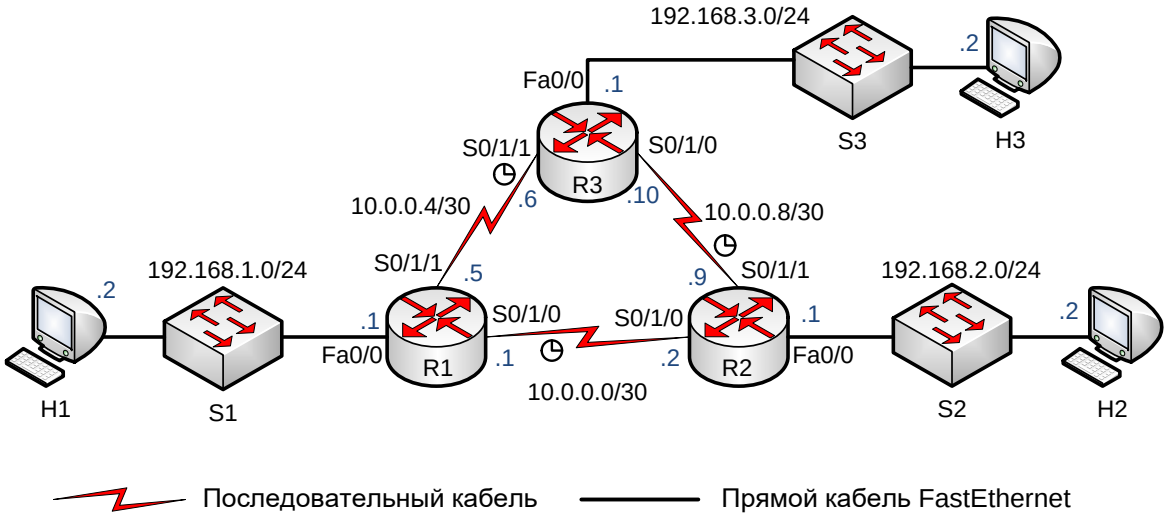


Рисунок 50 – Топология сети

Таблица 6 – Таблица адресации

Узел	Интерфейс	IP адрес	Маска	Шлюз
R1	FastEthernet0/0	192.168.1.1	255.255.255.0	–
	Serial0/1/0 (DCE)	10.0.0.1	255.255.255.252	–
	Serial0/1/1 (DTE)	10.0.0.5	255.255.255.252	–
R2	FastEthernet0/0	192.168.2.1	255.255.255.0	–
	Serial0/1/0 (DTE)	10.0.0.2	255.255.255.252	–
	Serial0/1/1 (DCE)	10.0.0.9	255.255.255.252	–
R3	FastEthernet0/0	192.168.3.1	255.255.255.0	–
	Serial0/1/0 (DTE)	10.0.0.10	255.255.255.252	–
	Serial0/1/1 (DCE)	10.0.0.6	255.255.255.252	–
H1	CiscoLAN	192.168.1.2	255.255.255.0	192.168.1.1
H2	CiscoLAN	192.168.2.2	255.255.255.0	192.168.2.1
H3	CiscoLAN	192.168.3.1	255.255.255.0	192.168.3.1

Подключите сетевые устройства и узлы, как показано на схеме топологии, используя последовательные кабели и прямые кабели

FastEthernet. Последовательные кабели подключаются соответствующими сторонами в зависимости от назначения интерфейса (*DCE*/*DTE*).

Подключите консольные кабели к консольным портам маршрутизаторов и ПК. Для доступа к консоли маршрутизатора будем использовать программу *Hyper Terminal*.

В программе *HyperTerminal* выбирите новое подключение, задаем любое имя, выбираем последовательный порт, скорость задаем 9600 бод, остальные параметры по умолчанию.

На узлах *H*1, *H*2, *H*3 настройте параметры *IP* протокола в соответствии с таблицей адресации.

Базовая настройка маршрутизатора

В этом параграфе описана последовательность действий по базовой настройке маршрутизатора *R*1. Для маршрутизаторов *R*2 и *R*3 действия аналогичны. Отличаются названия интерфейсов и *IP*-адреса, которые необходимо сверять с топологией и таблицей адресации.

В программе *HyperTerminal* для маршрутизатора переведите консоль в привилегированный режим, войдите в режим конфигурирования устройства и задайте ему имя:

```
Router>enable
Router#configure terminal
Router(config)#hostname R1
R1(config)#
```

Настройте интерфейс *FastEthernet* в соответствии с таблицей адресации:

```
R1(config)#interface FastEthernet 0/0
R1(config-if)#description R1 LAN Default Gateway
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
```

Настройте последовательные интерфейсы в соответствии с таблицей адресации:

```
R1(config)#interface Serial 0/1/0
R1(config-if)#description R1 to R2 link
R1(config-if)#ip address 10.0.0.1 255.255.255.252
```

Команда *clock rate* 64000 подается только для интерфейсов, помеченных как *DCE*:

```
R1(config-if)#clock rate 64000
R1(config-if)#no shutdown
R1(config-if)#exit
```

```
R1(config)#interface Serial 0/1/1
R1(config-if)#description R1 to R3 link
R1(config-if)#ip address 10.0.0.5 255.255.255.252
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#exit
```

Проверьте правильность настройка вашей подсети из общей топологии. Откройте на узле *H*1 окно командной строки *Windows* (Пуск -> Выполнить -> *cmd*). Выполните команду:

```
C:\>ping 192.168.1.1
```

Проверка достижимости маршрутизатора из узла должна быть успешной.

Проверка общей связности в сети

Проверьте правильность настройки интерфейсов на маршрутизаторе. В консоли маршрутизатора из привилегированного режима подайте команду:

```
R1#show ip interfaces brief
```

Изучите вывод, который показывает маршрутизатор. Сколько интерфейсов всего выводится? Сколько из них активны и правильно работают (статус *up* и протокол *up*)? Правильные ли *IP* адреса у всех интерфейсов?

Проверьте связь с другими маршрутизаторами в топологии. В консоли маршрутизатора *R*1 из привилегированного режима для проверки связи с маршрутизатором *R*2 подайте команду:

```
R1#ping 10.0.0.2
```

Для проверки связи с маршрутизатором *R*3:

```
R1#ping 10.0.0.6
```

Проверка достижимости должна быть успешной.

Посмотрите таблицу маршрутизации вашего маршрутизатора:

```
R1#show ip route
```

Сейчас в таблице маршрутизации три сети и все они непосредственно подключены к маршрутизатору.

Проверьте связи между узлами сети. В командной строке узла H1 для проверки связи с узлом *H*2 выполните команду:

```
C:\>ping 192.168.2.2
```

Для проверки связи с узлом *H*3 выполните команду:

```
C:\>ping 192.168.3.2
```

Эти проверки связи являются безуспешными, поскольку в таб-

лице маршрутизации нет сведений о пути до сетей этих узлов. Решим эту проблему путем настройки динамической маршрутизации по протоколу *RIP*.

Настройка динамической маршрутизации

Включите протокол динамической маршрутизации *RIP*. Для этого зайдите в режим конфигурирования маршрутизатора и подайте команды:

```
R1(config)#router rip
R1(config)#version 2
R1(config)#no auto-summary
R1(config)#network 10.0.0.0
R1(config)#network 192.168.1.0
```

Подождите 30 секунд. Посмотрите таблицу маршрутизации маршрутизатора:

```
R1#show ip route
```

В таблицу добавились сети по протоколу *RIP* (обозначены буквой *R*).

Проверьте связи между узлами сети. В командной строке узла *H*1 для проверки связи с узлом *H*2 выполните команду:

```
C:\>ping 192.168.2.2
```

Для проверки связи с узлом *H*3 выполните команду:

```
C:\>ping 192.168.3.2
```

Теперь проверки связи между узлами сети являются успешными, потому что маршрутизаторы знаю путь до сетей этих узлов.

Опишите путь пакета от узла *H*1 до узла *H*2. Воспользуйтесь программой *tracert.exe* для определения пути.

ПЛАНИРОВАНИЕ АДРЕСАЦИЯ *IPV4* В КОМПЬЮТЕРНЫХ СЕТЯХ

Двоичные побитовые операции над *IP*-адресами

Планирование сетевой адресации является достаточно простым процессом, однако понимание этого процесса у многих начинающих администраторов вызывает проблемы, в основном, из-за малого опыта выполнения булевых операций над 32-битными двоичными числами. Данная глава содержит множество упражнений, призванных дать опыт, необходимый для усвоения процесса планирования сетевой адресации и начинается с параграфа, посвященному самым базовым операциям над *IP*-адресами.

Заполните таблицу 7, преобразовав десятичное число в 8-битное двоичное значение. Первое число уже преобразовано для примера. Помните, что восемь двоичных битовых значений в октете имеют основание 2 и слева направо выглядят как 128, 64, 32, 16, 8, 4, 2 и 1.

Таблица 7 – Перевод чисел в двоичную систему исчисления

Десятичное	Двоичное
192	11000000
168	
10	
255	
2	

*IPv*4-адрес преобразуются в двоичный формат точно так же, как было проделано выше, но состоит из четырех октетов (байт).

Заполните таблицу 8 двоичными эквивалентами указанных *IP*-адресов.

Для того, чтобы определить сетевой адрес для заданного *IP*-адреса, сначала вам необходимо перевести десятичный *IPv*4-адрес и маску подсети в их двоичный эквивалент. Затем применяется операция двоичного сложения «И» для *IP*-адреса и маски. Полученное 32-битное значение представляет собой сетевой адрес.

Таблица 8 – Перевод *IP*-адресов в двоичную систему исчисления

Десятичное	Двоичное
192.168.10.10	11000000.10101000.00001010.00001010
209.165.200.229	
172.16.18.183	
10.86.252.17	
255.255.255.128	
255.255.192.0	

При использовании операции «И» десятичное значение в каждой битовой позиции 32-битного *IP*-адреса узла сравнивается с соответствующей позицией в 32-битной маске подсети. При наличии двух нулей или 0 и 1 результатом операции «И» будет 0. При наличии двух единиц результатом будет 1, как показано в таблице 9.

Таблица 9 – Определение адреса сети

Описание	Десятичное	Двоичное
IP-адрес	192.168.10.131	11000000.10101000.00001010.10000011
Маска подсети	255.255.255.192	11111111.11111111.11111111.11000000
Сетевой адрес	192.168.10.128	11000000.10101000.00001010.10000000

Заполните таблицы 10, 11, 12, 13, 14.

Для того, чтобы определить, находятся ли два *IP*-адреса в одной сети, необходимо выполнить сравнение сетевых адресов для обоих *IP*-адресов. Сетевые адреса получаются применением операции «И» к *IP*-адресу и маске. Если у обоих *IP*-адресов одинаковый сетевой адрес, то они находятся в одной сети и могут непосредственно взаимодействовать, отправляя пакеты друг другу напрямую. Если же сетевые адреса не совпадают, то компьютеры находятся в разных сетях, для их взаимодействия требуется маршрутизирующее устройство, и пакеты необходимо отправлять на адрес основного шлюза ПК.

Таблица 10 – Задание по определению адреса сети, вариант А

Описание	Десятичное	Двоичное
IP-адрес	172.16.145.29	
Маска подсети	255.255.0.0	
Сетевой адрес		

Таблица 11 – Задание по определению адреса сети, вариант Б

Описание	Десятичное	Двоичное
IP-адрес	192.168.10.10	
Маска подсети	255.255.255.0	
Сетевой адрес		

Таблица 12 – Задание по определению адреса сети, вариант В

Описание	Десятичное	Двоичное
IP-адрес	192.168.68.210	
Маска подсети	255.255.255.128	
Сетевой адрес		

Таблица 13 – Задание по определению адреса сети, вариант Г

Описание	Десятичное	Двоичное
IP-адрес	172.16.188.15	
Маска подсети	255.255.240.0	
Сетевой адрес		

Таблица 14 – Задание по определению адреса сети, вариант Д

Описание	Десятичное	Двоичное
IP-адрес	10.172.2.8	
Маска подсети	255.224.0.0	
Сетевой адрес		

Компьютеру ПК-А присвоен *IP*-адрес 192.168.1.18, а компьютеру ПК-Б — *IP*-адрес 192.168.1.33. Маска подсети обоих компьютеров — 255.255.255.240. Какой сетевой адрес у ПК-А? Какой

сетевой адрес у ПК-Б? Смогут ли эти ПК взаимодействовать друг с другом напрямую?

Компьютеру ПК-А присвоен *IP*-адрес 10.0.0.16, а компьютеру ПК-Б — *IP*-адрес 10.1.14.68. Маска подсети обоих компьютеров — 255.254.0.0. Какой сетевой адрес у ПК-А? Какой сетевой адрес у ПК-Б? Смогут ли эти ПК взаимодействовать друг с другом напрямую? Какой наименьший адрес, присвоенный компьютеру ПК-Б, позволит ему находиться в одной сети с ПК-А?

Вычисление адресов для заданной сети

Маска сети определяет, какие биты *IP*-адреса относятся к адресу сети, а какие – к адресу узла внутри сети. Число битов в узловой части адреса сети определяет количество различных уникальных адресов в этой сети и широковещательный адрес.

В маске сети узловые биты всегда следуют за сетевыми, поэтому очень часто для сокращения вместо полной маски подсети указывают число сетевых бит. Например, для сети 192.168.10.0 с маской 255.255.255.0 можно указать 192.168.10.0/24.

IP-адреса внутри сети распределяются следующим образом, как представлено в таблице 15.

Таблица 15 – Специальные значения *IP*-адресов

Адрес	Биты	Назначение
Сетевой	000…000 (все нулевые)	Адрес сети, идентифицирует саму сеть и не выдается узлам.
Широковещательный	111…111 (все единичные)	Адресует все узлы внутри сети, используется для передачи информации сразу для всей сети.
Уникальные	xxx…xxx (все прочие)	Арес для отдельного узла в сети, должны быть уникальными.

Рассчитать количество узлов для каждой сети можно путём анализа маски подсети. Число отдельных IP-адресов можно определить по формуле:

$$\{\text{количество узлов}\} = 2^{\{\text{количество бит узла}\}} - 2$$

Маска подсети может быть представлена в десятичном формате с точкой-разделителем, например 255.255.192.0, или в формате сетевого префикса, например /18. *IPv*4-адрес всегда содержит 32 бита. Отняв количество битов, используемых сетевой частью (как показано в маске подсети), вы получите количество битов, используемых для узлов.

Например, маска подсети 255.255.192.0 равна /18 в префиксной записи. Вычитание 18 бит сети из 32 бит даст нам 14 бит, оставшихся для узловой части. Исходя из этого, можно выполнить простой расчёт:

$$2^{14} = 16\ 384 - 2 = 16\ 382 \text{ узла}$$

Проанализируйте таблицу 16 и определите сетевую и узловую части указанных *IPv*4-адресов. Первые две строки содержат примеры заполнения таблицы. Сокращения, используемые в таблице:

С = все 8 бит для октета содержатся в сетевой части адреса;

с = бит в сетевой части адреса;

У = все 8 бит для октета содержатся в узловой части адреса;

у = бит в узловой части адреса.

Таблица 16 – Определение сетевой и узловой части

IP-адрес/префикс	**Сеть/узел** С, с = сеть У, у = узел	**Маска подсети**	**Сетевой адрес**
192.168.10.10/24	С.С.С.У	255.255.255.0	192.168.10.0
10.101.99.17/23	С.С.ссссссу.У	255.255.254.0	10.101.98.0
209.165.200.227/27			
172.31.45.252/24			
10.1.8.200/26			
172.16.117.77/20			
10.1.1.101/25			
209.165.202.140/27			
192.168.28.45/28			

Проанализируйте таблицу 17 и укажите диапазон адресов узлов и широковещательных адресов. В первой строке приведён пример завершения таблицы.

Таблица 17 – Определение диапазона IP-адресов

Адрес IPv4/ префикс	Сетевой адрес	Адрес первого узла	Адрес последнего узла	Широковеща-тельный адрес	Nбп	Nбу	N
192.168.10.10/24	192.168.10.0	192.168.10.1	192.168.10.254	192.168.10.255	24	8	254
10.101.99.17/23							
209.165.200.227/27							
172.31.45.252/24							
10.1.8.200/26							
172.16.117.77/20							
10.1.1.101/25							
209.165.202.140/27							
192.168.28.45/28							
192.168.100.25/28							
172.30.10.130/30							
10.1.113.75/19							
198.133.219.250/24							
128.107.14.191/22							
172.16.104.99/27							

Классификация *IPv*4-адресов

Все множество существующих *IPv*4 адресов разбито на несколько классов.

Отдельно выделяются адреса кольцевого интерфейса, которые относятся к сети 127.0.0.0/8. Эти адреса предназначены для различных сервисных функций стека *TCP*/*IP*. Наиболее известен и часто используется адрес 127.0.0.1, который присутствует на каждом компьютере с установленным и правильно работающим стеком *TCP*/*IP*. Все адреса кольцевого интерфейса не адресуются в Интернет и не могут использоваться для передачи данных по сети.

Еще одним важным классом адресов являются частные адреса, которые предназначены для использования в частных локальных сетях. Эти адреса могут присваиваться узлам в локальной сети, но не могут использоваться с сети Интернет. Для обеспечения доступа к узлам из такой локальной сети с частными адресами в Интернет используются механизмы трансляции (преобразования, подмены) адресов – *NAT*. Частные адреса принадлежат диапазонам, представленным в таблице 18.

Определенный диапазон адресов в Интернет предназначен для многоадресной рассылки: 224.0.0.0 – 239.255.255.255. Многоадресной называют рассылку, получателями которой является несколько (от одного и более) узлов в сети Интернет. Любой узел

может подключиться к многоадресной рассылке с определенным адресом и получать ее пакеты, может отключиться в любой момент. Эти механизмы регулируются специальным протоколом *IGMP*. Адрес из многоадресного диапазона не может быть присвоен отдельному компьютеру как его *IP*-адрес.

Таблица 18 – Диапазоны частных *IP*-адресов

Название	**Значения**
Диапазон класса *A*	10.0.0.0 – 10.255.255.255
Диапазон класса *B*	172.16.0.0 – 172.31.255.255
Диапазон класса *C*	192.168.0.0 – 192.168.255.255

В диапазоне *IP*-адресов выделен специальный блок для экспериментальной исследовательской работы и тестирование сетей: 240.0.0.0 – 255.255.255.254. Эти адреса обычно не присваиваются узлам в работающих штатно сетях и могут не маршрутизироваться в сети Интернет, но их можно присвоить любому узлу.

В каждой сети есть отдельный адрес, называемый широковещательным (все узловые биты которого равны 1). Кроме этого, существует глобальный широковещательный адрес – 255.255.255.255. Получателями такого адреса теоретически могли бы быть все узлы в сети Интернет, но в целях безопасности и защиты от перегрузки в глобальной сети, областью действия такого адреса обычно является только сеть, в которой он был отправлен. То есть он мало отличается от широковещательного адреса самой сети. В своих целях администраторы могут организовывать передачу пакетов глобального широковещательного адреса за пределы одного сетевого адреса, но в сети Интернет передача такого адреса запрещена.

Все прочие *IP*-адреса называют общими, они могут работать в сети Интернет без ограничений.

Таким образом, допустимыми для присвоения отдельному узлу являются адреса, которые не являются сетевыми, широковещательными или многоадресными, адресами кольцевого интерфейса.

Таблица 19 – Определение типа *IP*-адреса

IP-адрес	Маска подсети	Тип адреса
10.1.1.1	255.255.255.252	узел
192.168.33.63	255.255.255.192	
239.192.1.100	255.252.0.0	
172.25.12.52	255.255.255.0	
10.255.0.0	255.0.0.0	
172.16.128.48	255.255.255.240	
209.165.202.159	255.255.255.224	
172.16.0.255	255.255.0.0	
224.10.1.11	255.255.255.0	

Таблица 20 – Определение частных *IP*-адресов

IP-адрес/префикс	Общий или частный
209.165.201.30/27	
192.168.255.253/24	
10.100.11.103/16	
172.30.1.100/28	
192.31.7.11/24	
172.20.18.150/22	
128.107.10.1/16	
192.135.250.10/24	
64.104.0.11/16	

Проанализируйте таблицу 19 и определите тип адреса (адрес сети, узла, многоадресной или широковещательной рассылки). В первой строке приведён пример завершения таблицы.

Проанализируйте таблицу 20 и определите тип адреса – общий или частный.

Проанализируйте таблицу 21 и определите, является ли пара адреса и префикса допустимым адресом узла.

Таблица 21 – Определение допустимых *IP*-адресов

IP-адрес/префикс	**Допустимый или нет адрес узла**	**Причина**
127.1.0.10/24		
172.16.255.0/16		
241.19.10.100/24		
192.168.0.254/24		
192.31.7.255/24		
64.102.255.255/14		
224.0.0.5/16		
10.0.255.255/8		
198.133.219.8/24		

Определение количества подсетей

Компьютерную сеть, состоящую из множества узлов, можно разбивать на подсети. Делается это по множеству причин, самыми главными из которых являются:

- из соображений безопасности – для каждого отдела организации можно выделить отдельную подсеть и ограничить доступ этих подсетей друг к другу во избежание утечки конфиденциальной информации или распространения вредоносных программ;
- из соображений производительности – каждая сеть для своей работы использует множество механизмов, основанных на использовании широковещательных пакетов. Чем больше число узлов в сети, тем больше таких пакетов. В сети размером в несколько сотен узлов приличная часть пропускной способности каналов связи занята передачей этих служебных широковещательных пакетов, уменьшая при этом полезную работу сети по передаче пользовательских данных. В масштабах всего Интернет использовать широковещательные пакеты уже попросту невозможно.

Понятие сети и подсети близки друг другу. Когда говорят о крупной сети, то *подсетью* называют какую-либо ее обособленную часть. Упоминается также понятие *суперсети*, которое используют при объединении сетей со смежными сетевыми адресами с уменьшением сетевой маски (для облегчения задачи маршрутизации уменьшением строк в таблице маршрутов устройств).

Отдельной подсетью считается сеть, в которой все узлы имеют одинаковый сетевой адрес и маску. Они могут без ограничений связываться друг с другом напрямую и посылать широковещательные пакеты. Совокупность таких узлов называют *доменом широковещательной рассылки*.

Узлы, которые имеют различные сетевые адреса, уже не являются частью одной подсети и не могут напрямую взаимодействовать друг с другом. Для связи узлов из двух различных подсетей используется специальное сетевой устройство – маршрутизатор, основными задачами которого как раз и являются связь различных подсетей и определение направления передачи пакета данных, исходя из сетевого адреса получателя этого пакета.

В крупной корпоративной сети, состоящей из множества маршрутизаторов, каждая подсеть отделена от другой маршрутизатором. Подсчитать число подсетей можно, определив, сколько подсетей в топологии разделены друг от друга маршрутизатором.

При определении сетевой адресации также важно определить, сколько узлов находится в каждой подсети, чтобы найти для этой подсети оптимальную маску. Количество узлов можно подсчитать, при этом необходимо помнить:

- каждый узел требует один *IP* адрес;
- концентраторы не требуют *IP* адресов, коммутаторы, как правило, их не требуют, но им может быть присвоен один *IP* адрес (*Vlan* 1), так что его тоже необходимо учитывать;
- интерфейс (порт) маршрутизатора, к которому подключена подсеть требует один *IP* адрес, который одновременно является адресом шлюза по умолчанию (*default gateway*) для этой подсети.

Зная количество узлов, можно определить, какая сетевая маска необходима этой подсети. Число *IP*-адресов определяется по формуле:

$$\{\text{количество бит}\} = \lceil \log_2(\{\text{количество узлов}\} + 1) \rceil$$

Операция $\lceil x \rceil$ означает округление числа x до целого вверх.

То есть необходимо найти такое число, чтобы 2 в степени этого числа было не меньше требуемого количества узлов плюс 2.

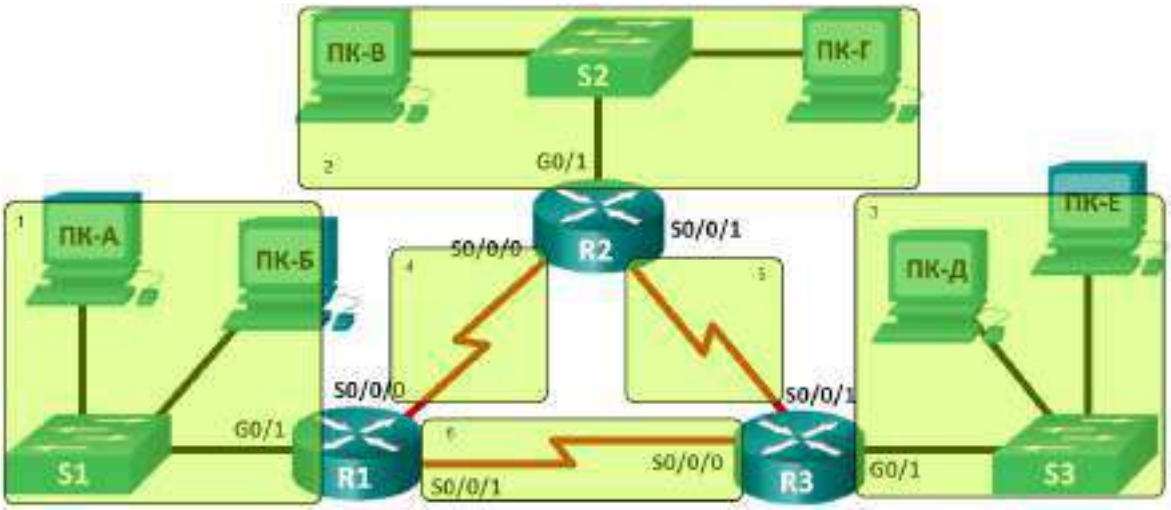


Рисунок 51 – Определение количества подсетей

Таблица 22 – Вычисление количества бит подсети

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0	S1, ПК-А, ПК-Б, R1(G0/1)	4	3
1	S2, ПК-В, ПК-Г, R2(G0/1)	4	3
2	S3, ПК-Д, ПК-Е, R3(G0/1)	4	3
3	R1(S0/0/0), R2(S0/0/0)	2	2
4	R2(S0/0/1), R3(S0/0/1)	2	2
5	R1(S0/0/1), R3 (S0/0/0)	2	2

На рисунке 51 показано разбиение сети на шесть подсетей. Подсети пронумерованы и выделены цветом. В таблице 22 представлен пример расчета количества бит, необходимых для сетевой части каждой подсети.

Определите количество подсетей в топологии на рисунке 52 и заполните таблицу 23.

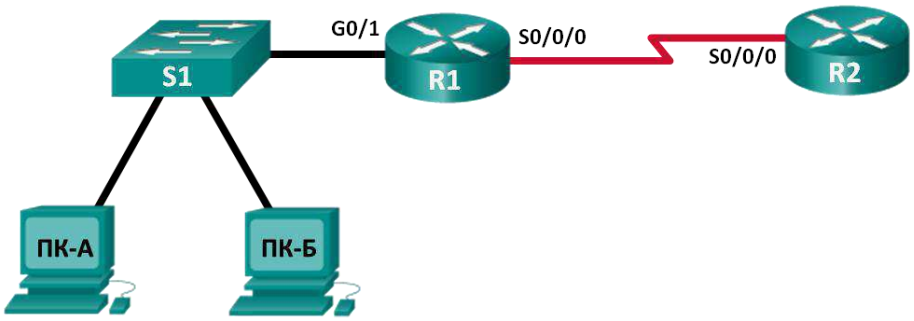


Рисунок 52 – Определение количества подсетей, вариант А

Таблица 23 – Вычисление количества бит подсети, вариант А

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			

Определите количество подсетей в топологии на рисунке 53 и заполните таблицу 24.

Определите количество подсетей в топологии на рисунке 54 и заполните таблицу 25.

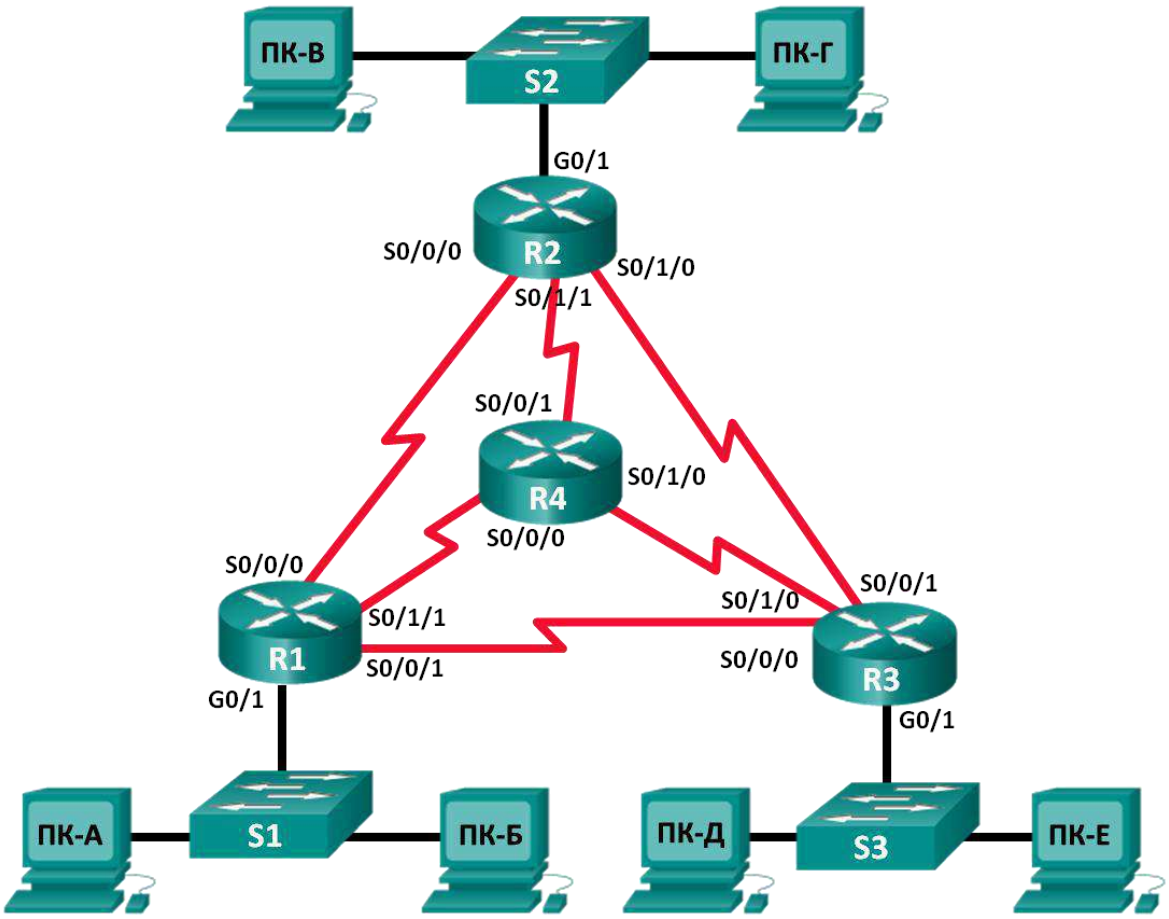


Рисунок 53 – Определение количества подсетей, вариант Б

Таблица 24 – Вычисление количества бит подсети, вариант Б

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

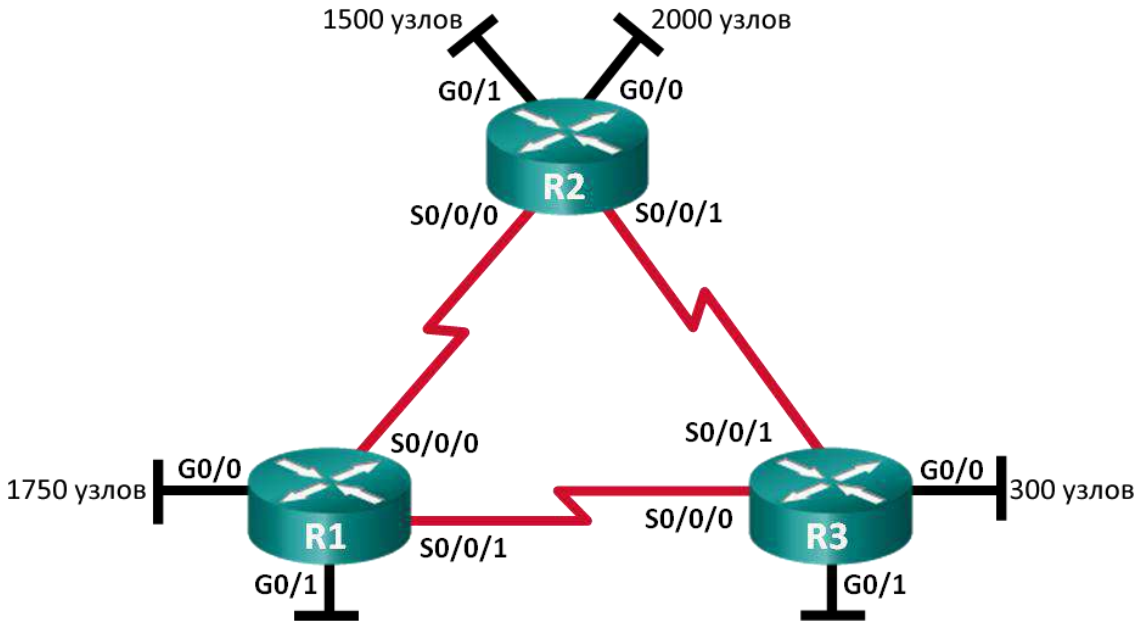


Рисунок 54 – Определение количества подсетей, вариант В

Таблица 25 – Вычисление количества бит подсети, вариант В

№ п/сети	Узлы, интерфейсы	Узлов	Бит
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

Разбиение сети на подсети

Одной из часто решаемых задач при проектировании компьютерной сети является разбиение сети на подсети и выделение диапазонов сетевых адресов для соответствующих подсетей.

Обычно проектировщику сети дается один диапазон адресов, либо сеть, заданная в виде адрес/маска, на всю проектируемую сеть. Его задачей является определить количество подсетей в сети, количество узлов в каждой подсети, а затем, исходя из этих данных, спланировать разбиение сети на подсети.

При разбиении маска каждой подсети становится больше, чем исходная маска всей сети. Число бит, на которые увеличивается маска, зависит от числа подсетей, и определяется по формуле:

$$\{\text{количество добавочных бит}\} = \lceil \log_2\{\text{количество подсетей}\} \rceil$$

То есть необходимо найти такое число, чтобы 2 в степени этого числа было не меньше требуемого количества подсетей.

Например, дана сеть 172.16.0.0/16. При анализе топологии сети было определено, что она состоит из 12 подсетей. По формуле для разбиения сети на подсети необходимо 4 добавочных бита.

Добавочные биты увеличивают маску в подсети, при этом уменьшают на то же число количество узловых бит. Из-за этого максимально допустимое число узлов в подсети снижается. Поэтому необходимо убедиться, что получаемое разбиение на подсети позволяет оставить в каждой подсети требуемое число узлов.

Такой подход к разбиению сети на подсети называется *разбиением с фиксированной маской*, потому что во всех подсетях при нем получаются маски одинаковой длины, и, следовательно, одинаковое число максимально допустимых узлов.

Зная *IPv*4-адрес, а также исходную и новую маски подсети, можно определить следующие параметры:

- сетевой адрес этой подсети;
- широковещательный адрес этой подсети;
- диапазон адресов узлов этой подсети;
- количество созданных подсетей;
- количество узлов в подсети.

В таблице 26 показан пример решения одной из задач разбиения сети на подсети с фиксированной длиной маски.

Таблица 26 – Определение разбиения на подсети, пример

Дано:	
IP-адрес узла	172.16.77.120
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.240.0
Найти:	
Количество битов подсети	4
Количество созданных подсетей	16
Количество битов узлов в подсети	12
Количество узлов в подсети	4094
Сетевой адрес этой подсети	172.16.64.0
Адрес IPv4 первого узла в этой подсети	172.16.64.1
Адрес IPv4 последнего узла в этой подсети	172.16.79.254
Широковещательный адрес IPv4 в этой подсети	172.16.79.255

Рассмотрим, как была получена таблица 26.

Исходная маска подсети имела вид 255.255.0.0 или /16. Новая маска подсети — 255.255.240.0 или /20. Полученная разница составляет 4 бита. Так как 4 бита были заимствованы, мы можем определить, что были созданы 16 подсетей, так как 2^4= 16.

В новой маске, равной 255.255.240.0 или /20, остаётся 12 бит для узлов. Воспользуемся следующей формулой:

$$2^{12}= 4096–2=4094$$

Таким образом, 4094 узла в каждой подсети.

Бинарная операция «И» поможет определить сетевой адрес подсети для заданного изначально *IP*-адреса, в результате чего мы получим сеть 172.16.64.0.

В заключение необходимо установить первый узел, последний узел и широковещательный адрес для этой подсети. В нашем примере узловая часть – это последние 12 бит адреса. В первом узле для всех старших битов будет установлено значение 0, а для младшего бита — значение 1. В последнем узле для всех старших битов будет установлено значение 1, а для младшего бита — значение 0.

В этом примере узловая часть адреса находится в третьем и четвёртом октетах. Таблица 27 наглядно демонстрирует расположение бит в сетевой и узловой частях.

Таблица 27 – Определение адресов подсети

1-й октет	**2-й октет**	**3-й октет**	**4-й октет**	**Описание**
cccccccc	**cccccccc**	**ccccуууу**	уууууууу	Маска подсети
10101100 172	**00010000** 16	**0100**0000 64	00000001 1	Первый узел
10101100 172	**00010000** 16	**0100**1111 79	11111110 254	Последний узел
10101100 172	**00010000** 16	**0100**1111 79	11111111 255	Широковещательный

Заполните таблицу 28, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Таблица 28 – Определение разбиения на подсети, вариант А

Дано:	
IP-адрес узла	192.168.200.139
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.224
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 29 – Определение разбиения на подсети, вариант Б

Дано:	
IP-адрес узла	10.101.99.228
Исходная маска подсети	255.0.0.0
Новая маска подсети	255.255.128.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 30 – Определение разбиения на подсети, вариант В

Дано:	
IP-адрес узла	172.22.32.12
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.224.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 31 – Определение разбиения на подсети, вариант Г

Дано:	
IP-адрес узла	192.168.1.245
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.252
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 32 – Определение разбиения на подсети, вариант Д

Дано:	
IP-адрес узла	128.107.0.55
Исходная маска подсети	255.255.0.0
Новая маска подсети	255.255.255.0
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Таблица 33 – Определение разбиения на подсети, вариант E

Дано:	
IP-адрес узла	192.135.250.180
Исходная маска подсети	255.255.255.0
Новая маска подсети	255.255.255.248
Найти:	
Количество битов подсети	
Количество созданных подсетей	
Количество битов узлов в подсети	
Количество узлов в подсети	
Сетевой адрес этой подсети	
Адрес IPv4 первого узла в этой подсети	
Адрес IPv4 последнего узла в этой подсети	
Широковещательный адрес IPv4 в этой подсети	

Заполните таблицу 29, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 30, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 31, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 32, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Заполните таблицу 33, указав необходимые значения для указанного IPv4-адреса, а также исходной и новой масок подсети.

Планирование сетевой адресации

Окончательным этапом полного планирование сетевой адресации с фиксированной маской для готовой топологии сети является присвоение IP-адресов отдельным важным узлам сети, таким как маршрутизаторы и коммутаторы, серверы и выделенные узлы сети. Таким образом, последовательность действий включает:

- определение числа подсетей и вычисление числа добавочных бит;
- разбиение сети на подсети и составление схемы адресации;
- присвоение адресов узлам сети и интерфейсам сетевых устройств.

Для всех остальных обычных узлов сети отдельные адреса обычно не задаются при планировании, а составляется схема адресации аналогично, как это делалось ранее для одной сети, но теперь это выполняется для всех подсетей.

Например, имея сеть 172.22.0.0 с маской 255.255.0.0 или /16, мы разбиваем ее на 4 подсети. Для такого разбиения нам необходимо 2 добавочных бита, таким образом маска становится /18 или 255.255.192.0.

Адрес каждой подсети будет изменяться в пределах добавочных бит и принимать двоичные значения 00, 01, 10, 11. Сетевая часть адреса, предшествующая добавочным битам, не изменяется.

Таблица 34 – Определение добавочных бит подсетей

№ п/с	Первый октет	Второй октет	Третий октет	Четвертый октет	Сетевой адрес
0	**10101100** 172	**00010110** 22	**00**000000 0	00000000 0	172.22.0.0/18
1	**10101100** 172	**00010110** 22	**01**000000 64	00000000 0	172.22.64.0/18
2	**10101100** 172	**00010110** 22	**10**000000 128	00000000 0	172.22.128.0/18
3	**10101100** 172	**00010110** 22	**11**000000 192	00000000 0	172.22.192.0/18

В таблице 34 приведен пример такого расчета. В двоичном виде добавочные биты подсети выделены подчеркиванием, вся сетевая часть адреса выделена жирным, узловая обычным шрифтом. Ниже двоичного представления дан десятичный эквивалент. У всех подсетей маска одинаковая 255.255.192.0 или /18.

Как видно из таблицы 34, диапазоны всех подсетей не пересекаются друг с другом, а все вместе образуют исходную сеть 172.22.0.0/16.

Для присвоение сетевого адреса узлу или интерфейсу устройства можно выбрать любой допустимый узловой *IP*-адрес из подсети. Обычно интерфейсам маршрутизаторов присваивают первый допустимый адрес в подсети, поскольку он является шлюзом по умолчанию для всех узлов подсети и так его легче запомнить. Адреса узлов при необходимости могут выбираться совершенно произвольно.

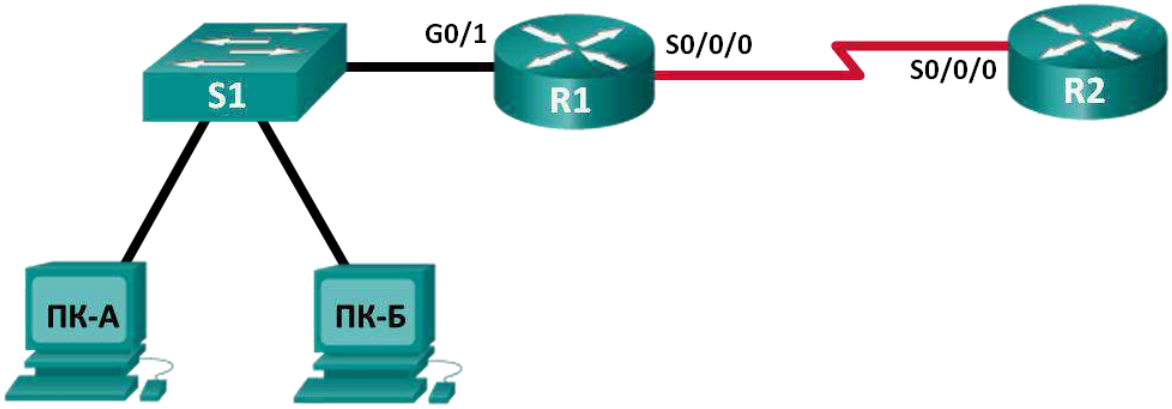


Рисунок 55 – Планирование сетевой адресации, вариант А

Таблица 35 – Планирование подсетей, вариант А

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				

Таблица 36 – Схема адресации, вариант А

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
R2	Serial 0/0/0		
S1	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		

Для сети с топологией на рисунке 55 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 35.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 36.

Для сети с топологией на рисунке 56 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 37.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 38.

Для сети с топологией на рисунке 57 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 39.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 40.

Для сети с топологией на рисунке 58 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 41.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 42.

Для сети с топологией на рисунке 59 дается диапазон адресов 192.168.10.0/24. Определите необходимое количество подсетей и заполните таблицу 43.

Сколько бит необходимо использовать для создания полученного количества подсетей?

Сколько имеется полезных адресов узла в подсети в данной структуре адресации?

Как будет выглядеть новая маска подсети в десятичном формате с точкой-разделителем?

Сколько подсетей останутся свободны для использования в будущем?

Придумайте допустимые IP адреса для узлов и устройств сети, а затем составьте схему адресации для полученной сети и заполните таблицу 44.

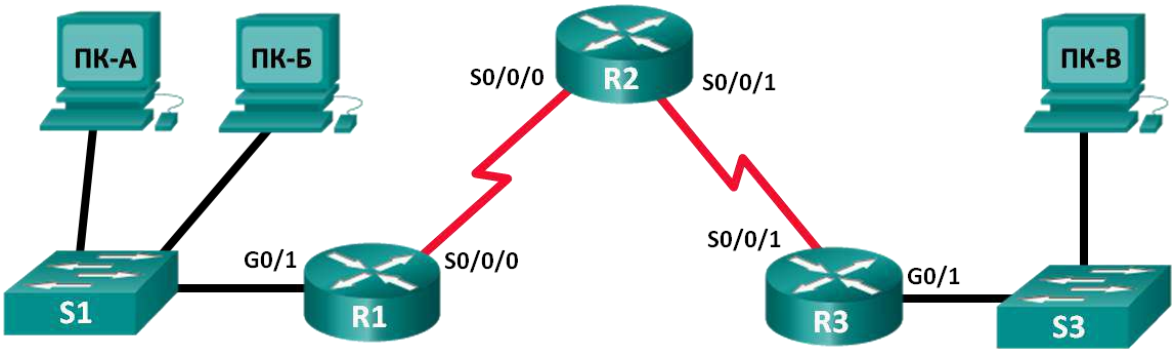


Рисунок 56 – Планирование сетевой адресации, вариант Б

Таблица 37 – Планирование подсетей, вариант Б

Номер подсети	Адрес подсети	Первый используемый адрес узла	Последний используемый адрес узла	Широковещательный адрес
0				
1				
2				
3				

Таблица 38 – Схема адресации, вариант Б

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
R2	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/1		
S1	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		

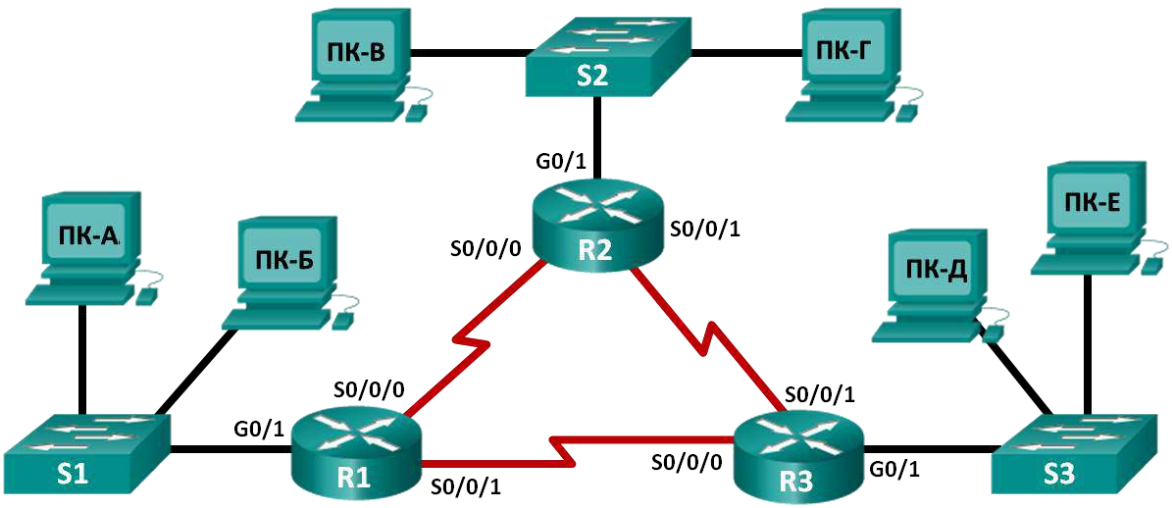


Рисунок 57 – Планирование сетевой адресации, вариант В

Таблица 39 – Планирование подсетей, вариант В

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				
2				
3				
4				
5				

Таблица 40 – Схема адресации, вариант В

Устройство	**Интерфейс**	**IP-адрес**	**Маска подсети**
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R2	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
S1	Vlan 1		
S2	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		
ПК-Г	GigabitEthernet		
ПК-Д	GigabitEthernet		
ПК-Е	GigabitEthernet		

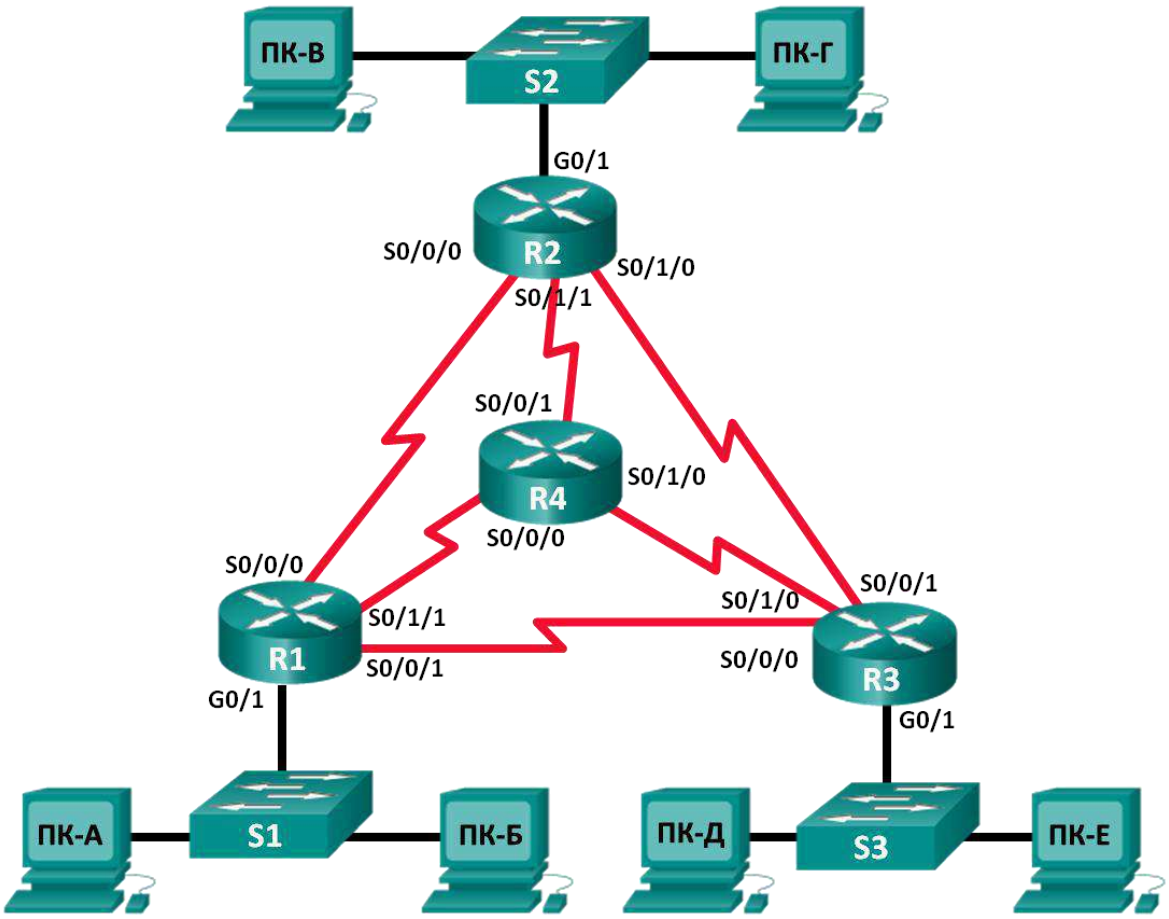


Рисунок 58 – Планирование сетевой адресации, вариант Г

Таблица 41 – Планирование подсетей, вариант Г

Номер подсети	Адрес подсети	Первый используемый адрес узла	Последний используемый адрес узла	Широковещательный адрес
0				
1				
2				
3				
4				
5				
6				
7				
8				

Таблица 42 – Схема адресации, вариант Г

Устрой-ство	**Интерфейс**	**IP-адрес**	**Маска подсети**
R1	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/1		
R2	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/1/0		
	Serial 0/1/1		
R3	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/0		
R4	Serial 0/0/0		
	Serial 0/0/1		
	Serial 0/1/0		
S1	Vlan 1		
S2	Vlan 1		
S3	Vlan 1		
ПК-А	GigabitEthernet		
ПК-Б	GigabitEthernet		
ПК-В	GigabitEthernet		
ПК-Г	GigabitEthernet		
ПК-Д	GigabitEthernet		
ПК-Е	GigabitEthernet		

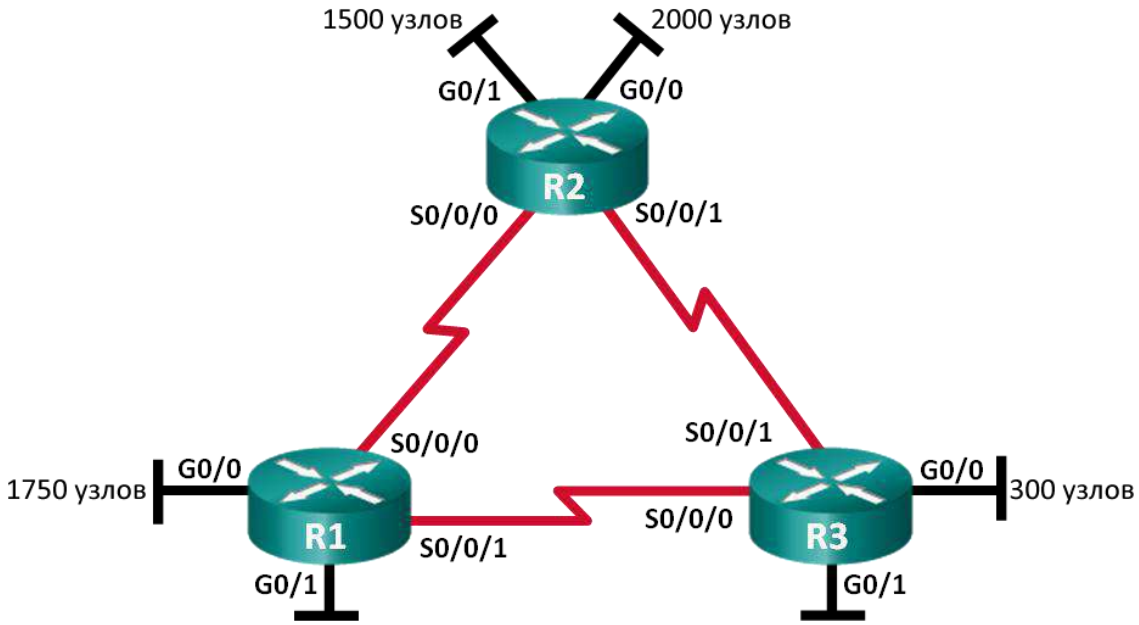


Рисунок 59 – Планирование сетевой адресации, вариант Д

Таблица 43 – Планирование подсетей, вариант Д

Номер подсети	**Адрес подсети**	**Первый используемый адрес узла**	**Последний используемый адрес узла**	**Широковещательный адрес**
0				
1				
2				
3				
4				
5				
6				
7				
8				

Таблица 44 – Схема адресации, вариант Д

Устройство	Интерфейс	IP-адрес	Маска подсети
R1	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R2	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		
R3	GigabitEthernet 0/0		
	GigabitEthernet 0/1		
	Serial 0/0/0		
	Serial 0/0/1		

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Cisco Networking Academy. CCNA: Введение в сетевые технологии [Электронный ресурс] – Режим доступа: https://www.netacad.com/ru/courses/networking/ccna-introduction-networks
2. Cisco Networking Academy. CCNA 7: Switching, Routing, and Wireless Essentials [Электронный ресурс] – Режим доступа: https://www.netacad.com/ru/courses/networking/ccna-switching-routing-wireless-essentials
3. Синецкий Р.М. Компьютерные сети [Электронный ресурс]: электрон. образ. ресурс. – ЮРГПУ (НПИ), 2020. – Режим доступа: https://sdo.srspu.ru/enrol/index.php?id=276
4. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: учеб. пособие для вузов. - СПб: Питер, 2010. - 944 с.
5. Таненбаум Э. Компьютерные сети: перевод с английского. - СПб: Питер, 2008. - 992 с.
6. Айвенс К. Компьютерные сети:как победить глюки и заставить домашнюю сеть работать без сбоев. - СПб: Питер, 2006. - 298 с.
7. Куроуз Д.Ф., Росс К. Компьютерные сети: многоуровневая архитектура Интернета. - СПб: Питер, 2004. - 765 с.
8. Закер К. Компьютерные сети. Модернизация и поиск неисправностей: перевод с английского. - СПб: БХВ-Петербург, 2004. - 1008 с.
9. Суворов А.Б. Телекоммуникационные системы, компьютерные сети и интернет: учеб. пособие для вузов. - Ростов-н/Д: Феникс, 2007. - 384 с.
10. Синецкий Р.М. Компьютерные сети: метод. указ. к практ. занятиям, лаб. работам и самостоят. раб. студ. заоч. формы обуч. бакалавр. "Информатика и вычисл. техника". - Новочеркасск: ЮРГПУ(НПИ), 2017. - 20 с.
11. Синецкий Р. М. Компьютерные сети: метод. указания к практ. занятиям, лаб. раб. и самостоят. работе студентов бакалавриата напрвлений подгот. 02.03.03, 09.03.01, 09.03.04 очной формы обучения. - Новочеркасск: ЮРГПУ (НПИ), 2017. - 19 с.

Учебное издание

Синецкий Роман Михайлович

АДМИНИСТРИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Учебное пособие

Редактор *Я.В. Максименко*

Подписано в печать 13.08.2020
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 5,11. Уч.- изд. л. 5,5. Тираж 100 экз. Заказ 46-0380.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ(НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
ipd-npi@mail.ru

Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

АЛГОРИТМЫ ОБУЧЕНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

**Учебно-методическое пособие
по изучению дисциплины
«Искусственный интеллект и мягкие вычисления»**

**Новочеркасск
ЮРГПУ (НПИ)
2020**

УДК 004.8 (076.5)
ББК 32.813

Рецензент – кандидат технических наук, доцент, доцент кафедры
«Программное обеспечение вычислительной техники»
Гавриков Михаил Михайлович

Кузнецова А.В., Мохов В.А., Алгазали С.М.М.

Алгоритмы обучения искусственных нейронных сетей: учебно-методическое пособие по изучению дисциплины Искусственный интеллект и мягкие вычисления / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2020. – 72 с.

Теоретический материал и приведенные примеры направлены на изучение принципов обучения искусственных нейронных сетей двух базовых типов архитектур – слоистых и однослойных. Контрольные вопросы к темам и практические задания для ручного расчёта направлены на углубленное изучение алгоритмов обучения с целью их последующей реализации в виде программ для ЭВМ и создания различных модификаций.

Учебно-методическое пособие предназначено для студентов, обучающихся по программам бакалавриата направлений подготовки 09.03.01 «Информатика и вычислительная техника» очной и заочной форм обучения, 09.03.04 «Программная инженерия», 02.03.03 «Математические основы и алгоритмизация информационных систем».

УДК 004.8 (076.5)
ББК 32.813

ВВЕДЕНИЕ

Искусственные нейронные сети (ИНС) работают на тех же принципах, что и их биологический прототип – мозг. С точки зрения реализации – это один из программных механизмов, который позволяет программе обучаться, учитывать имеющийся опыт, самостоятельно находить закономерности в исходных данных, запоминать информацию и восстанавливать её по отдельным фрагментам, учитывать предысторию наблюдаемых процессов и накапливать информацию для выработки правильной стратегии управления, и многое другое. В целом, это семейство технологий для решения трудно формализуемых задач, которые требуют аналитических способностей, присущих человеку.

Нейросети могут иметь различную структуру и в зависимости от задачи, которую необходимо решить, наилучшим образом подходит та или иная архитектура. Например, для классификации используют многослойные персептроны, для анализа временных рядов, распознавания речи и прочих событий, растянутых во времени, – рекуррентные сети, для распознавания изображений – свёрточные сети. Однако есть и универсальные архитектуры, которые предназначены для решения нескольких типов задач. В этом случае всё зависит от способа представления данных для обучения ИНС.

По общему мнению, наиболее подходящим языком для реализации нейронных сетей является язык *python*. Для создания и обучения нейронных сетей можно воспользоваться его готовыми фреймворками типа *tensorflow, keras, theano* и надстройками над ними. Как правило, это мощные средства, изначально разрабатывавшиеся в рамках крупных исследовательских проектов, переданные программистскому сообществу в бесплатное пользование.

Однако, создавать нейронные сети можно и на *Java*, и на *C++*, и вообще на любом общедоступном процедурном или объектно-ориентированном языке программирования. В этом случае придётся самостоятельно заниматься моделированием системы нейронов и алгоритмом обучения.

Предлагаемое учебное пособие направлено на формирование базовых представлений и объяснение принципов работы наиболее распространённых нейронных сетей – слоистых сетей прямого распространения и самоорганизующихся сетей Кохонена.

Тема 1. МОДЕЛЬ ИСКУССТВЕННОГО НЕЙРОНА И ЕГО МОДИФИКАЦИИ

Искусственный нейрон является основой любой искусственной нейронной сети. Нейроны представляют собой относительно простые, однотипные элементы, имитирующие работу биологических нейронов мозга.

Искусственные нейроны до определенной степени отражают динамику передачи сигналов в реальных биологических нейронах. Биологические нервные клетки (нейроны) состоят из тела клетки, входных отростков – дендритов и выхода – аксона. Очень упрощая, работу нейрона можно описать следующим образом. Дендриты получают сигналы от других клеток через особые биологические образования – синапсы (или синаптические щели). Сигналы поступают в тело клетки, где они суммируются с другими такими же входными сигналами. Если суммарный сигнал в течение короткого промежутка времени является достаточно большим, то клетка возбуждается, вырабатывая в аксоне импульс, который передается на следующие клетки. Не вдаваясь в подробности, подчеркнем, что модели нейронов только очень грубо отражают работу биологических живых нервных клеток.

Искусственный нейрон, так же, как и его естественный прототип, имеет группу входов, которые соединены с выходами других нейронов, и один выход, через который сигнал возбуждения или торможения поступает на входы других нейронов. На рис. 1.1 показана схема искусственного нейрона.

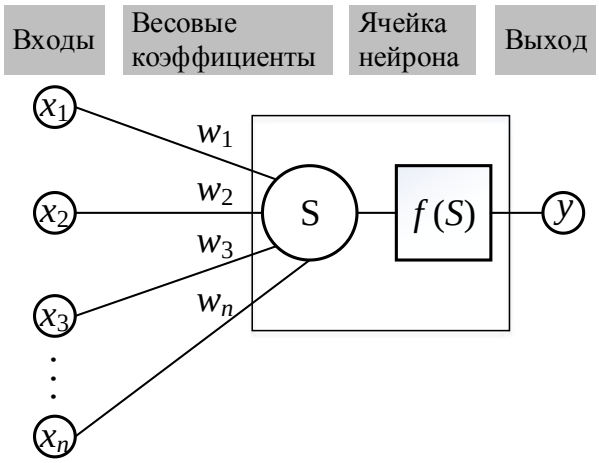


Рис. 1.1. Графическая модель искусственного нейрона

Каждый вход характеризуется весовым коэффициентом или просто весом w_i, который по своему физическому смыслу отражает силу синаптической связи и эквивалентен электрической проводимости. Текущее состояние нейрона определяется как взвешенная сумма его входов:

$$S = \sum_{i=1}^{n} x_i \cdot w_i, \qquad (1.1)$$

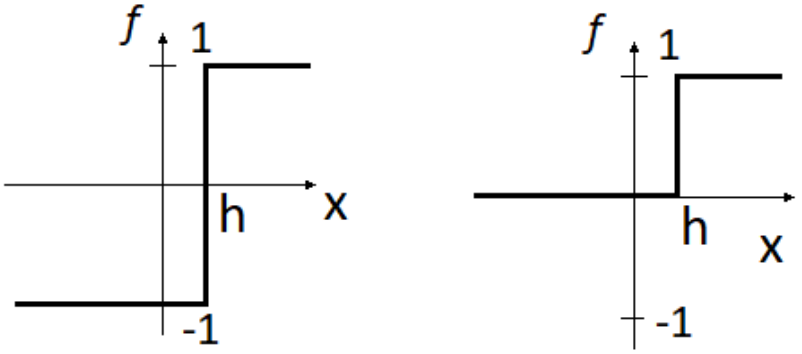
где n – число входов нейрона; x_i – значение i-го входа нейрона; w_i – вес i-го синапса.

Выход нейрона определяется значением некоторой функции от его состояния, т.е.:

$$y = f(S). \qquad (1.2)$$

Эту функцию называют активационной, сжимающей функцией или функцией возбуждения нейрона.

Первой функцией в модели искусственного нейрона была пороговая функция, разновидности которой представлены на рис. 1.2.



а *б*

Рис. 1.2. Пороговые активационные функции
а – пороговая биполярная; *б* – пороговая бинарная

Для рассмотрения работы нейрона неважно, приходит сигнал к нейрону из внешнего мира или с другого нейрона, и неважно, куда отправляется сигнал с нейрона.

В том случае, когда на вход искусственного нейрона подаются двоичные сигналы, а активационная функция – пороговая (функции Хевисайда) с параметром h, мы имеем дело с моделью *формального нейрона*, предложенного Мак-Каллоком и Питтсом в 1943 году. Математическое представление модели с n входами имеет вид:

$$y = \begin{cases} 1, \ \sum_{i=1}^{n} x_i * w_i \geq h; \\ 0, \ \sum_{i=1}^{n} x_i * w_i < h, \end{cases} \qquad (1.3)$$

где x – входной сигнал; y – выходной сигнал; h – значение порога.

Например, для нейрона со следующими значениями весов его трёх входов: w_1 = 0.30, w_2 = 0.40, w_3 = 0.25, значением порога h = 0.45 и входным двоичным сигналом 1 0 1 выходное значение будет определяться выражением:

$$0.3*1 + 0.4*0+0.25*1 = 0.55; \ 0.55 > 0.45 \ \Rightarrow \ y=1.$$

Для вычисления выходного сигнала модифицированного формального нейрона использовать правило:

$$\frac{(Сумма\ активных\ весов)}{(Полная\ сумма\ входных\ весов)} \geq (Порог)$$

Для этого нейрона справедливо то, что если есть в наличии два любых активных входа, выход будет всегда активным. Формула расчета значения на выходе модифицированного нейрона будет иметь следующий вид:

$$\frac{0.3*1+0.25*1}{|0.3|+|0.4|+|0.25|} = 0.58; \quad 0.58 > 0.45 \Rightarrow y = 1.$$

Обучим нейрон, функционирующий по формуле (1.3), распознаванию идеального графического образа, представленного битовыми значениями (рис. 1.3). Входной вектор, соответствующий символу «F», можно закодировать, например, построчно:

1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0.

Обучение заключается в подборе параметров нейрона – весовых коэффициентов и значения порога, таких, что только при заданном входном сигнале на выходе нейрона будет единица. Очевидно, что таких наборов параметров может быть сколь угодно много.

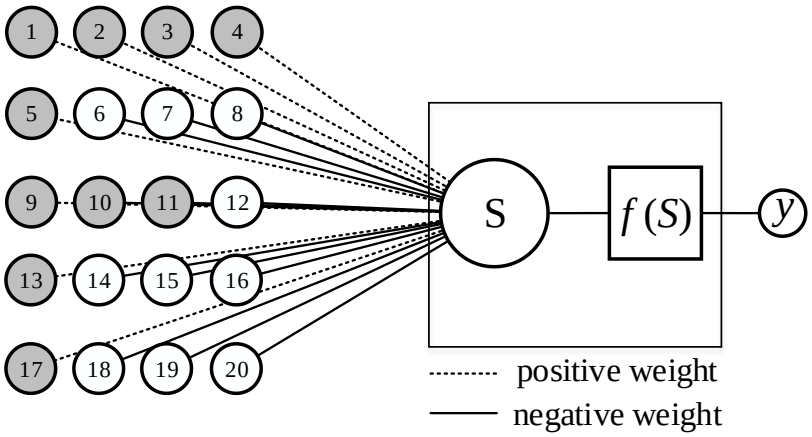


Рис. 1.3. Модель нейрона для распознавания символа «F»

Выберем для активных входов, т.е. входов, на которые подаются 1, малые положительные значения:

w_1=0.11, w_2=0.14, w_3=0.12, w_4=0.10, w_5=0.12,

w_9=0.11, w_{10}=0.13, w_{10}=0.10, w_{13}=0.11, w_{17}=0.14.

Для неактивных входов – малые случайные отрицательные значения в диапазоне [-1.0 .. -0.2].

Подсчитаем взвешенную сумму для входного сигнала, соответствующего символу «F»:

0.11+0.14+0.12+0.10+0.12+0.11+0.13+0.10+0.11+ 0.14=1.18.

Значение порога h следует выбрать таким, чтобы при отсутствии хотя бы одной единицы на входе, взвешенная сумма оказалась меньше порога. Поскольку минимальное значение веса для активного входа равно 0.1, то величина порога должна находиться в диапазоне [1.09 .. 1.18]. При отсутствии хотя бы одной единицы на определенном выше множестве активных входов, взвешенная сумма буде заведомо меньше значения 1.09. В наилучшем случае она может быть равной 1.08, если ноль окажется на входе с номером 4 или с номером 10. Если же на активных входах будут присутствовать единицы, но появятся «лишние» единицы на тех входах, где весовые коэффициенты отрицательные, то они заведомо уменьшат взвешенную сумму 1.18 на величину -0.2. Итоговая сумма снова станет меньше 1.09.

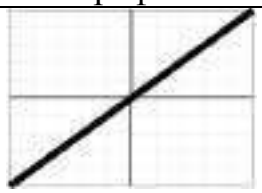
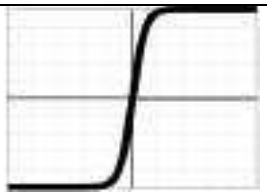
Значения входов x_i и выхода y формального нейрона могут быть как бинарными, т.е. равными 0 или 1, так и бинарными биполярными, равными ±1. В основополагающей работе Мак-Каллока и Питтса [1] входы и выходы нейронов предполагались бинарными,

веса синапсов считались бинарными биполярными, а активационная функция – пороговой. Исследования нейросетей в [1] проводились с точки зрения анализа логических исчислений, которые могут быть построены на базе формальных нейронов. В частности, было показано, что «для всякого логического выражения, удовлетворяющего некоторым условиям, можно найти сеть, имеющую описываемое этим выражением поведение».

Заменив простую пороговую функцию на гладкую ограниченную функцию, можно перейти к модели *градуального нейрона*. В такой модели нейрона информация, передаваемая от нейрона к нейрону, представлена не строгими бинарными, а действительными числовыми значениями.

В табл. 1.1 представлены наиболее часто употребляемые гладкие функции активации. К желательным свойствам функций активации относятся: нелинейность, непрерывная дифференцируемость, бесконечная область значений, монотонность.

Таблица 1.1 Гладкие функции активации искусственного нейрона

Название	График	Уравнение
Линейная		$f(x) = x$
Логистическая или сигмоидальная		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
Гиперболический тангенс		$f(x) = th(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$

Наиболее часто в качестве активационной функции используется так называемая *сигмоидальная функция*, которая в общем случае имеет следующий вид:

$$f(x) = \frac{1}{1+e^{-ax}}\ , \qquad (1.4)$$

При уменьшении параметра α функция становится более пологой, и в пределе, при $\alpha = 0$, вырождается в горизонтальную линию на уровне 0.5. При увеличении α сигмоид приближается по внешнему виду к функции единичного скачка с порогом $h = 0$. Из выражения (1.4) следует, что выходное значение нейрона лежит в диапазоне [0..1]. Одно из ценных свойств сигмоидальной функции – простое выражение для ее производной:

$$f'(x) = af(x)(1 - f(x)) \ , \qquad (1.5)$$

Следует отметить, что сигмоидальная функция дифференцируема на всей оси абсцисс, что используется в большинстве алгоритмов обучения. Вторым замечательным свойством этой функции является её способность усиливать слабые сигналы лучше, чем большие, и предотвращать насыщение от больших сигналов, поскольку они соответствуют областям аргументов, где сигмоида имеет пологий наклон.

Вопросы для самоконтроля

1. Что представляет собой модель формального нейрона?
2. Какой смысл несут в себе весовые коэффициенты нейрона?
3. Что представляет собой взвешенная сумма входного сигнала?
4. Каково назначение пороговой функции нейрона?
5. Чем отличается модифицированная модель формального нейрона от классической модели?
6. Каким образом производится ручная подстройка нейрона сети?
7. Что понимается под активационной функцией нейрона? Какой смысл она в себе заключает?
8. В чем преимущества гладких функций перед пороговой функцией активации?
9. Опишите особенности и достоинства сигмоидальной функции активации.

Тема 2. ВЫЧИСЛЕНИЕ ОШИБКИ В СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ СИГНАЛА. МЕТОДЫ ВЫЧИСЛЕНИЯ ОШИБКИ

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами сети при прямом проходе входного сигнала (образца) по нейронной сети – от входов к выходам.

Различают ошибку для образца (или ошибку за один проход) и ошибку для эпохи. Ошибка, формируемая в каждой эпохе, при верном обучении должна идти на спад. Если этого не происходит, значит, алгоритм обучения не верен.

Ошибку можно вычислить разными методами, но мы рассмотрим лишь три основных: *mean squared error* (далее *MSE*), *root MSE* (*RMSE*) и *Arctan*:

$$E_{MSE} = \frac{(d_1\text{-}y_1)^2 + (d_2\text{-}y_2)^2 + \ldots + (d_n\text{-}y_n)^2}{n}, \qquad (2.1)$$

$$E_{RMSE} = \sqrt{\frac{(d_1\text{-}y_1)^2 + (d_2\text{-}y_2)^2 + \ldots + (d_n\text{-}y_n)^2}{n}}, \qquad (2.2)$$

$$E_{arctan} = \frac{\arctan^2(d_1\text{-}y_1) + \ldots + \arctan^2(d_n\text{-}y_n)}{n}. \qquad (2.3)$$

Принцип подсчета ошибки во всех случаях одинаков. За каждый проход вычисляется ошибка путем вычитания от полученного ответа (d_i) от идеального ответа (y_i). Далее, результат либо возводится в квадрат, либо вычисляется квадратный тангенс из этой разности. В случае одного прохода $n = 1$. В случае эпохи n – число образцов в обучающей выборке. По завершении эпохи полученная сумма делится на количество наборов данных в обучающей выборке. При вычислении ошибки на всей эпохе можно определить наихудшую и наилучшую ошибки на определенных образцах.

Для вычисления ошибки сети нет какого-либо ограничения. Стоит лишь учитывать, что каждый метод считает ошибки поразному. У *Arctan*, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У *root MSE* будет наименьшая ошибка, поэтому, чаще всего, используют *MSE*, которая сохраняет баланс в вычислении ошибки.

Пример расчета ошибки для обучения на примере логической функции *XOR*.

Пусть задана нейронная сеть со случайно заданными значениями весовых коэффициентов (рис. 2.1). Функции активации нейронов – сигмоидальные.

Параметры сети:
w_1=0.45, w_2=0.78, w_3= -0.12,w_4=0.13, w_5=1.50, w_6= -2.3.

Обучающая выборка:

x_1	x_2	y
1	0	1
0	1	1
0	0	0
1	1	0

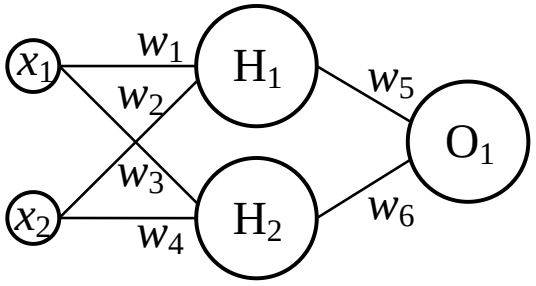


Рис. 2.1. Архитектура сети

Расчет выходных значений нейронов скрытого слоя для первого элемента обучающей выборки (x_1=1, x_2=0, y=1):

H_{1inp} = 1*0.45+0*(-0.12)=0.45 H_{1out} = sigmoid(0.45)=0.61
H_{2inp} = 1*0.78+0*0.13=0.78 H_{2out} = sigmoid(0.78)=0.69

Расчет выходных значений нейронов выходного слоя:

O_{1inp} = 0.61*1.5+0.69*-2.3=-0.67 O_{1out} = sigmoid(-0.67)=0.34

Абсолютная ошибка: Δ_1 = 1-0.34 = 0.66

Для остальных элементов обучающей выборки расчеты аналогичные. Среднеквадратическая ошибка сети на всей выборке:

$$E_{MSE} = ((1 - 0.34)^2+(1 - 0.37)^2+(0 - 0.40)^2+(0 - 0.32)^2)/4 =$$
$$= (0.66^2+0.63^2+(-0.4)^2+(-0.32)^2)/4 = 0.27 = 27\%.$$

Вопросы для самоконтроля

1. Что понимается под ошибкой сети?
2. Какие методы вычисления ошибки являются наиболее распространенными? В чем их отличие?
3. Приведите примеры других способов и методов вычисления ошибки, отличных от изученных на практическом занятии.
4. С какой целью вычисляется ошибка сети?

Тема 3. ОБУЧЕНИЕ СЕТЕЙ ПРЯМОГО РАСПРОСТРАНЕНИЯ СИГНАЛА. АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Нейронные сети «прямого распространения сигнала и обратного распространения ошибки» – это мощнейший инструмент поиска закономерностей, прогнозирования, качественного анализа. Более краткими названиями являются синонимы: *сети прямого распространения сигнала* (*feed forward*) или *сети обратного распространения ошибки (back propagation)*. Такие названия сети получили из-за используемого алгоритма обучения, в котором уменьшение ошибки производится от выходного слоя к входному, т. е. в направлении, противоположном направлению распространения сигнала при нормальном функционировании сети (алгоритм *Back Propagation*).

Нейронная сеть прямого распространения сигнала состоит из нескольких слоев нейронов, причем каждый нейрон слоя i связан с каждым нейроном слоя $i+1$, т. е. речь идет о *полносвязной* НС.

Задача обучения сети сводится к нахождению некой функциональной зависимости $\mathbf{Y}=F(\mathbf{X})$ где $\mathbf{X}$ – входной, а $\mathbf{Y}$ – выходной векторы. В общем случае такая задача, при ограниченном наборе входных данных, имеет бесконечное множество решений. Для ограничения пространства поиска при обучении ставится задача минимизации целевой функции ошибки, которая находится по методу наименьших квадратов (2.1). Обучение нейросети производится методом градиентного спуска, т. е. на каждой итерации изменение веса производится по формуле:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \qquad (3.1)$$

где η – параметр, определяющий скорость обучения.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} * \frac{\partial S_j}{\partial w_{ij}} \qquad (3.2)$$

где y_i – значение выхода j-го нейрона; S_j – взвешенная сумма входных сигналов. При этом множитель

$$\frac{\partial S_j}{\partial w_{ij}} = x_i, \tag{3.3}$$

где x_i – значение i-го входа нейрона.

Рассмотрим определение первого множителя формулы (3.2).

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{\mathrm{d}y_k}{\mathrm{d}S_k} * \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{\mathrm{d}y_k}{\mathrm{d}S_k} * w_{jk}^{(n+1)}, \tag{3.4}$$

где k – число нейронов в слое n+1.

Введем вспомогательную переменную δ (дельта):

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_i} * \frac{\mathrm{d}y_i}{\mathrm{d}S_j}. \tag{3.5}$$

Тогда можно определить рекурсивную формулу для определения δ n-го слоя, если нам известно δ следующего (n+1)-го слоя:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} * w_{jk}^{(n+1)} \right] * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} \tag{3.6}$$

Нахождение же δ для последнего N-го слоя не представляет трудности, так как целевой вектор известен, т. е. вектор тех значений, которые должна выдавать нейронная сеть при данном наборе входных значений.

$$\delta_j^{(N)} = \left(y_i^{(N)} - d_i \right) * \frac{\mathrm{d}y_i}{\mathrm{d}S_j} \tag{3.7}$$

Вернёмся к формуле (3.1) в раскрытом виде:

$$\Delta w_{ij}^{(n)} = -\eta * \delta_j^{(n)} * x_i^n. \tag{3.8}$$

Сомножитель $\delta_j^{(n)} * x_i^n$ называется *градиентом*. В данном случае это – элемент, включающий в себя производную функции ошибки относительно одного весового коэффициента.

Полный алгоритм обучения нейронной сети состоит из следующих этапов:

1. подать на вход сети случайно выбранный входной вектор и определить для него значения выходов (прямой проход сигнала);
2. рассчитать δ для выходного слоя по формуле (3.7) и для скрытых слоев сети по формуле (3.6);
3. рассчитать изменения весов $\Delta w_{ij}^{(n)}$ всех нейронов сети по формуле (3.8);
4. скорректировать веса сети:

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}; \qquad (3.9)$$

5. повторить шаги 1 – 4 для всех векторов обучающей выборки, т.е. для всей *эпохи*;
6. если ошибка E_{MSE} на эпохе существенна, то перейти на шаг 1 (к новой эпохе), иначе – алгоритм завершен.

На рисунке 3.1 проиллюстрирован прямой проход сигнала по нейронной сети с одним скрытым слоем. Начальные значения весовых коэффициентов и входной вектор обучающей выборки те же, что и в предыдущем примере (см. тему 2).

Рассмотрим подробный расчёт новых весовых коэффициентов для одного элемента обучающей выборки.

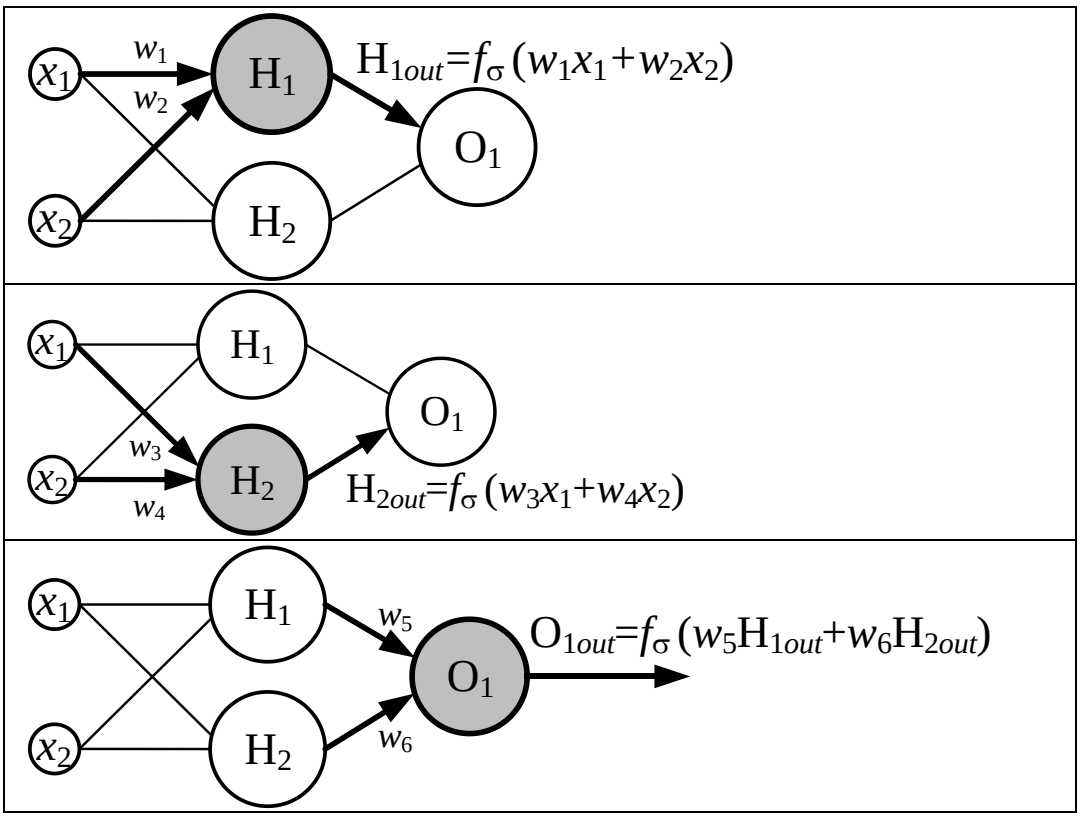


Рис. 3.1. Иллюстрация прямого прохода сигнала по НС

Расчет величины δ для единственного нейрона выходного слоя осуществляется на основе выходного значения O_{1out}, идеального значения выхода y_{ideal}, соответствующее входному вектору в обучающей выборке, и производной от выходного значения:

$$\delta_O = (O_{1out} - y_{ideal}) * f'(O_{1out}) \qquad (3.10)$$

Обратите внимание на производную. Во-первых, с алгоритмом нужно использовать только те функции активации, которые могут быть дифференцированы. Во-вторых, чтобы не делать лишних вычислений, формулу производной можно заменить на более дружелюбную и простую формула вида:

$$f'(OUT) = \begin{vmatrix} f_{sigmoid} = (1 - OUT) * OUT; \\ f_{th} = 1 - OUT^2. \end{vmatrix} \qquad (3.11)$$

Для выходного нейрона имеем:
Данные: $O_{1out} = 0.33$ $O_{1ideal} = 1$ $E_{MSE} = 0.449$
Решение: $\delta_{O_1} = (1 - 0.33) * ((1 - 0.33) * 0.33) = 0.148$

Для расчёта величины дельта в нейронах скрытого слоя вместо сомножителя ($O_{1out} - y_{ideal}$) в формуле (3.10) должна использоваться сумма $\sum(w_i * \delta_{next})$, т.е. сумма произведений всех дельт и весов нейронов последующего слоя, с которыми этот нейрон связан:

$$\delta_H = \sum(w_i * \delta_i) * f'(OUT) \qquad (3.12)$$

Но поскольку в рассматриваемой сети нейрон H_1 связан только с одним выходным нейроном O_1, производная от выхода H_1 умножается на единственны величину, равную $w_5 * \delta_{O1}$:

Данные: $H_{1out} = 0.61 \quad w_5 = 1.5 \quad \delta_{O1} = 0.148$
Решение: $\delta_{H1} = ((1 - 0.61) * 0.61) * (1.5 * 0.148) = 0.053$

На рисунке 3.2 проиллюстрирован процесс вычисления значений δ для нейронов выходного и скрытого слоёв.

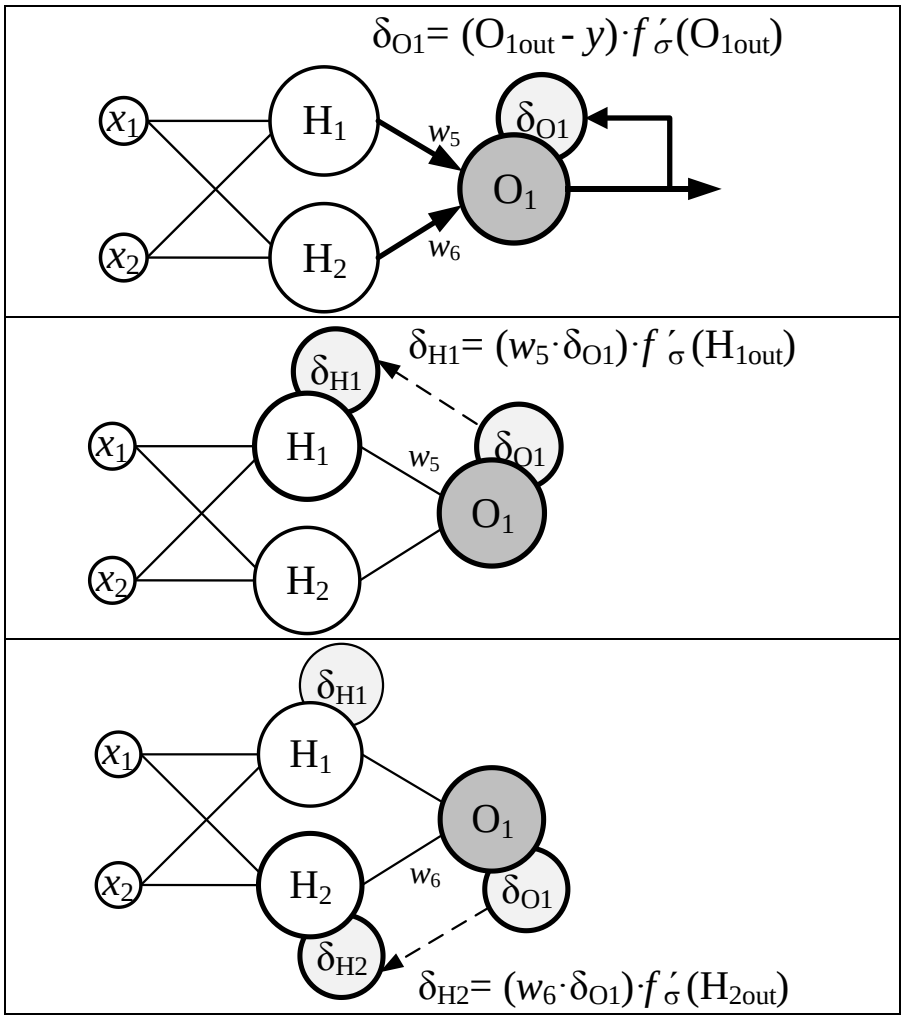


Рис. 3.2. Иллюстрация вычисления величины δ

Вычисление δ_{H2} производится аналогично: δ_{H2} = -0.073.

Для расчета подстроечных значений весов Δw следует найти градиент для каждого входа нейрона. В рассматриваемом методе формула нахождения градиента имеет вид:

$$GRAD_B^A = OUT_A * \delta_B . \qquad (3.13)$$

Здесь точка A – это точка в начале синапса (выход предыдущего нейрона), а точка B – на конце синапса (дельта самого нейрона). Таким образом можно подсчитать градиент w_5 следующим образом:

Данные: H_{1out} = 0.61 δ_{O1} = 0.148
Решение: $GRAD_{w5}$ = 0.61 * 0.148 = 0.09

Расчет подстроечных значений весовых коэффициентов осуществляется по формуле:

$$\Delta w_i = \eta * GRAD_{w_i} + \alpha * \Delta w_{i(pred)} . \qquad (3.14)$$

В отличие от формулы (3.8) здесь присутствует добавочный член. Переводя формулу в слова получим: изменение веса синапса равно коэффициенту скорости обучения η, умноженному на градиент этого веса, плюс момент α, умноженный на предыдущее изменение этого же веса. На первой итерации этот член равен 0. Настоятельно рекомендуется использовать момент α, так как это позволит избежать проблем с локальными минимумами.

Для весового коэффициента w_5 это будет выглядеть следующим образом.

Данные: η = 0.7 α= 0.3 w_5 = 1.5 $GRAD_{w5}$ = 0.09
Решение: $\Delta_{w5}(i\text{-}1)$ = 0 (предыдущих итераций не было)
Δ_{w5} = 0.7 * 0.09 + 0.3 * 0 = 0.063

Новое значение весового коэффициента w_5 получается путём сложения его значения с величиной подстройки:

$$w_5 = w_5 + \Delta w_5 = 1.5 + 0.063 = 1.563$$

Таким образом, после применения алгоритма весовой коэффициент w_5 изменился с 1.5 до 1.563 на малую величину 0.063.

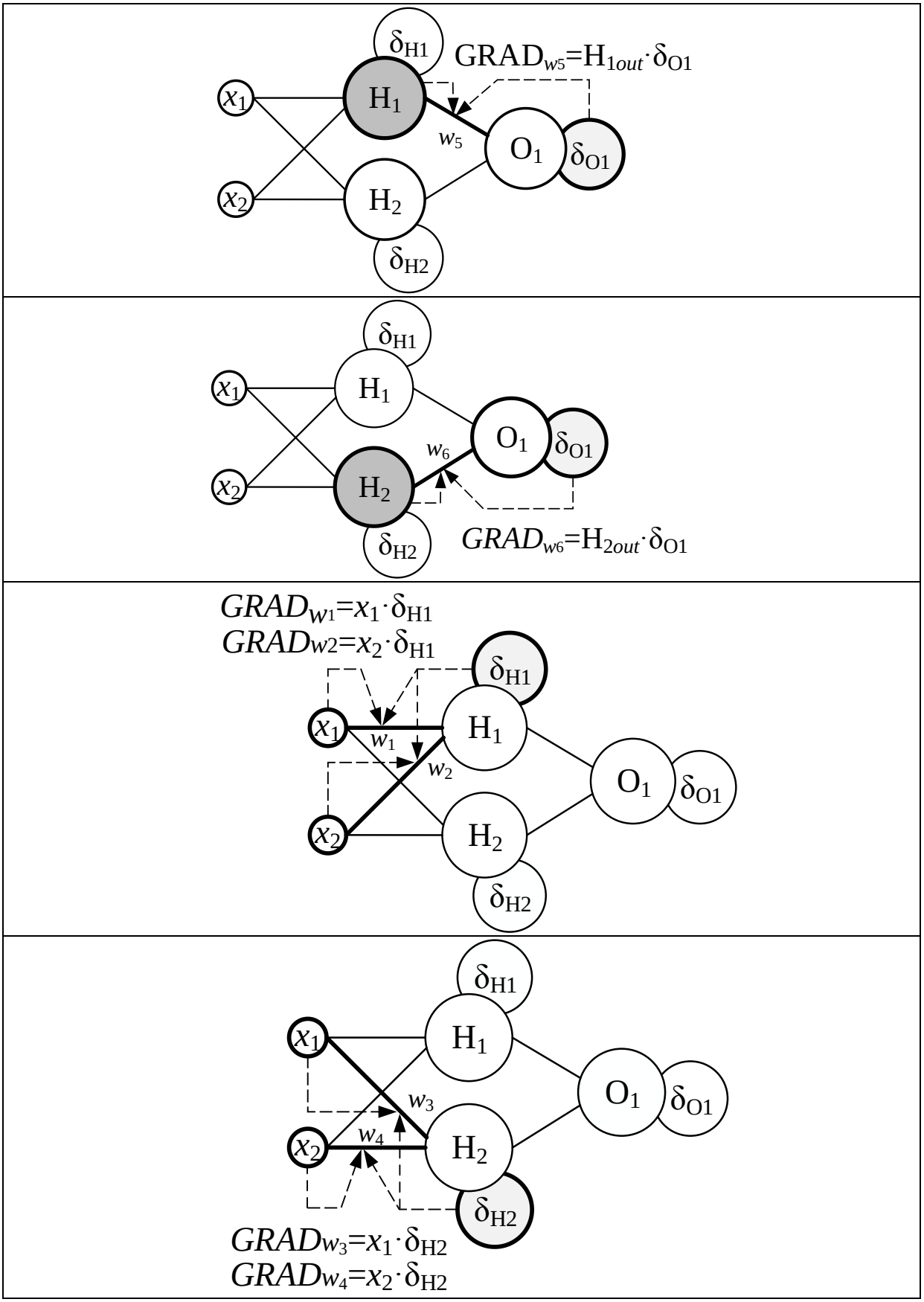


Рис. 3.3. Иллюстрация вычисления градиентов

Иллюстрация вычисления новых весовых коэффициентов представлена на рисунке 3.4.

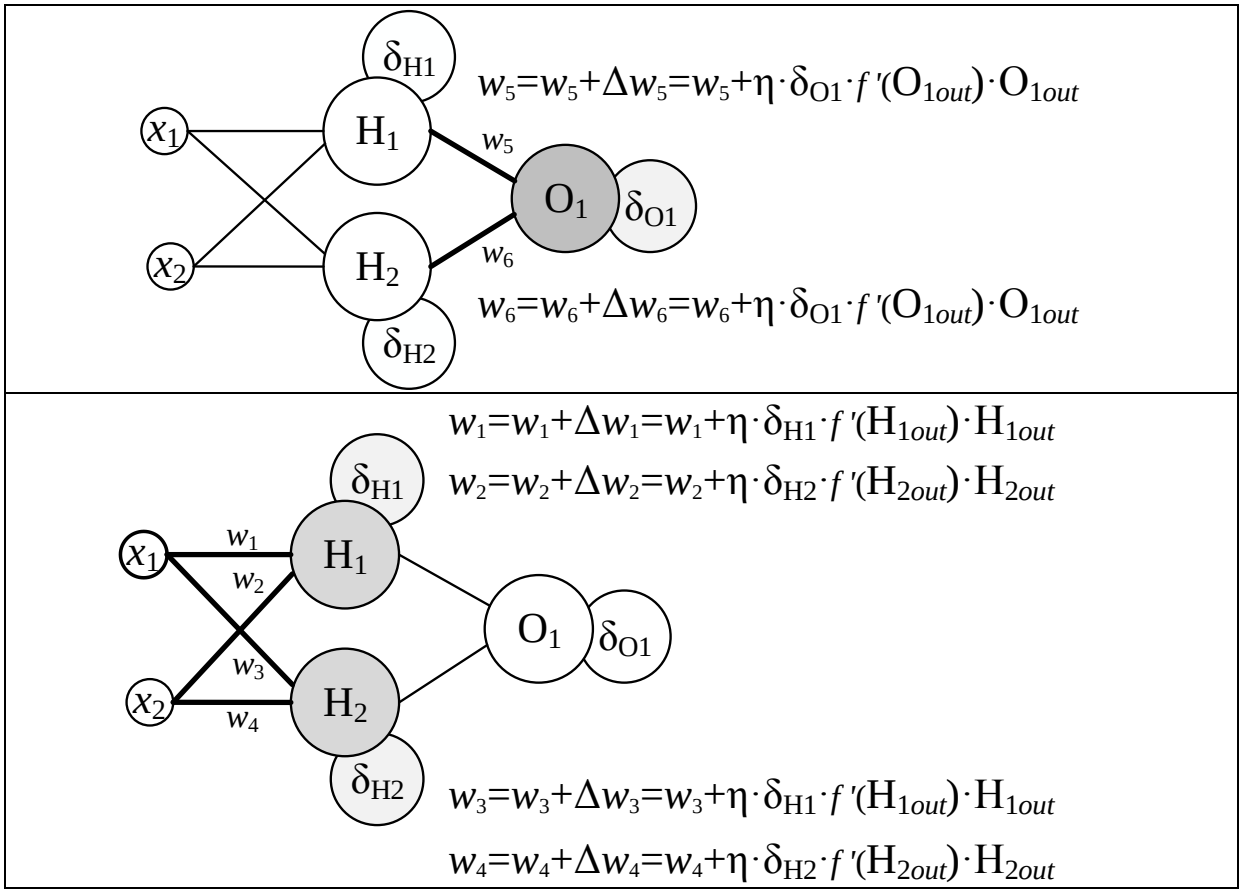


Рис. 3.4. Иллюстрация подстройки весовых коэффициентов

Величины градиентов, подстроечных коэффициентов и собственно новых значений весов для всех остальных входов нейронов вычисляются аналогично вычислениям для w_5. Результаты расчётов представлены в таблице 3.1. Из таблицы видно, как «колеблются» значения весовых коэффициентов после подстройки в сторону векторов обучающей выборки:

Таблица 3.1 Результаты расчётов весовых коэффициентов

	GRAD	Δw	$w^{(t)}$	$w^{(t+1)}$	$w^{(t+2)}$	$w^{(t+3)}$	$w^{(t+4)}$
w_6	0.090	0.071	-2.300	-2.229	-2.153	-2.168	-2.215
w_5	0.102	0.063	1.500	1.563	1.630	1.613	1.570
w_4	0.000	0.000	0.130	0.130	0.074	0.057	0.081
w_3	0.000	0.000	-0.120	-0.120	-0.080	-0.069	-0.089
w_2	-0.073	-0.052	0.780	0.729	0.713	0.709	0.736
w_1	0.053	0.037	0.450	0.487	0.498	0.501	0.479

Последовательная подстройка весовых коэффициентов способствует снижению ошибки сети. Ошибка для всей эпохи, полученная на первоначальных весовых коэффициентах, составила 0.273 (27.3%). После завершения первой эпохи ошибка уменьшилась до величины 0.264 (26.4%).

Вопросы для самоконтроля

1. Опишите суть алгоритма обратного распространения ошибки.
2. Какие задачи решаются на каждом из двух проходов алгоритма?
3. Что понимается под градиентным спуском и как используется градиент при подстройке коэффициентов сети?
4. Что понимается под скоростью обучения? Какие значения скорости рекомендуется выбирать?
5. Какую роль в обучении играет использование момента?
6. в процессе обучения циклически решаются однокритериальные задачи оптимизации.
7. Поясните смысл высказывания: «в процессе обучения многослойного персептрона циклически решаются однокритериальные задачи оптимизации».
8. Какие модификации базового алгоритма распространения Вам известны? Опишите их.

Тема 4. МАТРИЧНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Для вычисления преобразований сигналов между слоями нейронной сети можно использовать произведение матриц. Первая матрица содержит веса нейронов одного слоя. Вторая матрица – вектор – содержит поступившие на этот слой сети сигналы. Результат умножения (×) этих двух матриц – вектор с суммами взвешенных входных сигналов. Затем значения вектора используются в качестве аргумента функции активации (сигмоиды) нейронов слоя.

Рассмотрим прямое прохождение сигнала для нейронной сети с тремя входами и тремя выходами и двумя слоями (рис. 4.1). Первый слой – скрытый с нейронами H1, H2, H3. Второй слой – выходной с нейронами O1, O2, O3.

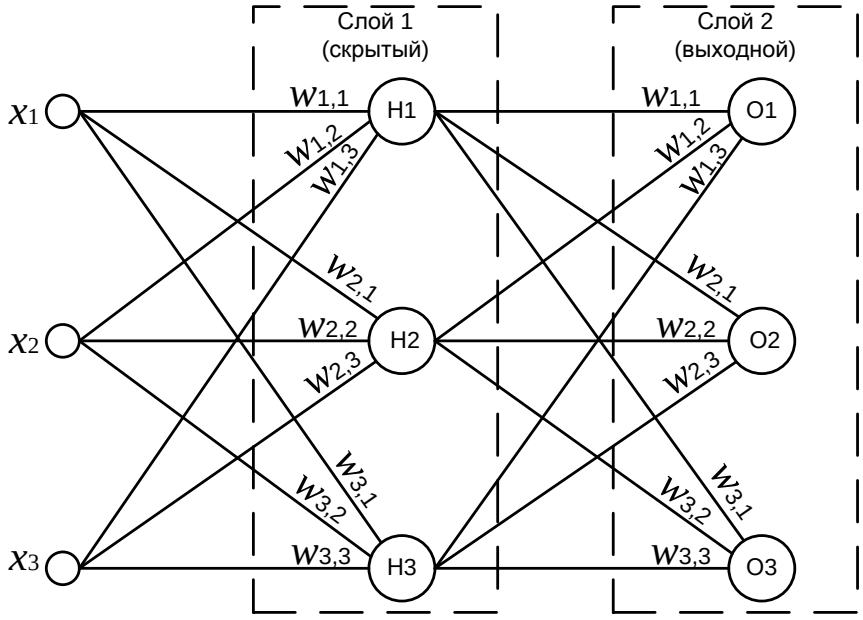


Рис. 4.1. Двухслойная нейронная сеть для реализации матричных операций

Для расчёта выходных значений скрытого слоя используется матрица весовых коэффициентов нейронов скрытого слоя **W1** и вектор входного сигнала **x**:

$$\begin{vmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{vmatrix} \times \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} x_1 \times w_{1,1} + x_2 \times w_{1,2} + x_3 \times w_{1,3} \\ x_1 \times w_{2,1} + x_2 \times w_{2,2} + x_2 \times w_{2,3} \\ x_1 \times w_{3,1} + x_2 \times w_{3,2} + x_3 \times w_{3,3} \end{vmatrix},$$

$$sigmoid\left(\begin{vmatrix} x_1 \times w_{1,1} + x_2 \times w_{1,2} + x_3 \times w_{1,3} \\ x_1 \times w_{2,1} + x_2 \times w_{2,2} + x_2 \times w_{2,3} \\ x_1 \times w_{3,1} + x_2 \times w_{3,2} + x_3 \times w_{3,3} \end{vmatrix} \right) = \begin{vmatrix} d_1^{(H1)} \\ d_2^{(H2)} \\ d_3^{(H3)} \end{vmatrix}$$

Для слоя 2 используется матрица весовых коэффициентов нейронов выходного слоя **W2**, а в качестве вектора входов – вектор, полученный на предыдущем шаге, т.е. вектор выходов нейронов скрытого слоя $\mathbf{d_H}$:

$$\begin{vmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{vmatrix} \times \begin{vmatrix} d_1^{(H1)} \\ d_2^{(H2)} \\ d_3^{(H3)} \end{vmatrix} = \begin{vmatrix} d_1^{(H1)} \times w_{1,1} + d_2^{(H2)} \times w_{1,2} + d_3^{(H3)} \times w_{1,3} \\ d_1^{(H1)} \times w_{2,1} + d_2^{(H2)} \times w_{2,2} + d_3^{(H3)} \times w_{2,3} \\ d_1^{(H1)} \times w_{3,1} + d_2^{(H2)} \times w_{3,2} + d_3^{(H3)} \times w_{3,3} \end{vmatrix},$$

$$sigmoid\left(\begin{vmatrix} d_1^{(H1)} \times w_{1,1} + d_2^{(H2)} \times w_{1,2} + d_3^{(H3)} \times w_{1,3} \\ d_1^{(H1)} \times w_{2,1} + d_2^{(H2)} \times w_{2,2} + d_3^{(H3)} \times w_{2,3} \\ d_1^{(H1)} \times w_{3,1} + d_2^{(H2)} \times w_{3,2} + d_3^{(H3)} \times w_{3,3} \end{vmatrix} \right) = \begin{vmatrix} d_1^{(O1)} \\ d_2^{(O2)} \\ d_3^{(O3)} \end{vmatrix}$$

В матричной форме обе эти операции записываются как:

$$\mathbf{W1} \times \mathbf{x} = \mathbf{d_H},$$
$$\mathbf{W2} \times \mathbf{d_H} = \mathbf{d_O}.$$

Пример. Зададим случайные значения весовым коэффициентам нейронов скрытого и выходного слоёв, т.е. элементам матриц **W1**, **W2**. Зададим входной вектор **x** и соответствующий ему выходной вектор **y**:

$$\mathbf{W1} = \begin{vmatrix} 0.90 & 0.30 & 0.40 \\ 0.20 & 0.80 & 0.20 \\ 0.10 & 0.50 & 0.60 \end{vmatrix} \quad \mathbf{W2} = \begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix}, \quad \mathbf{x} = \begin{vmatrix} 0.900 \\ 0.100 \\ 0.800 \end{vmatrix} \quad \mathbf{y} = \begin{vmatrix} 0.350 \\ 0.950 \\ 0.480 \end{vmatrix}$$

Дважды выполним матричный расчет выходов скрытого и выходного слоя:

$$\mathbf{W1} \times \mathbf{x} = \begin{vmatrix} 0.90 & 0.30 & 0.40 \\ 0.20 & 0.80 & 0.20 \\ 0.10 & 0.50 & 0.60 \end{vmatrix} \times \begin{vmatrix} 0.900 \\ 0.100 \\ 0.800 \end{vmatrix} = \begin{vmatrix} 1.160 \\ 0.420 \\ 0.620 \end{vmatrix} \quad \mathbf{d_H} = sigmoid\left(\begin{vmatrix} 1.160 \\ 0.420 \\ 0.620 \end{vmatrix} \right) = \begin{vmatrix} 0.761 \\ 0.603 \\ 0.650 \end{vmatrix}$$

$$\mathbf{W2} \times \mathbf{d_H} = \begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix} \times \begin{vmatrix} 0.761 \\ 0.603 \\ 0.650 \end{vmatrix} = \begin{vmatrix} 0.976 \\ 0.889 \\ 1.255 \end{vmatrix} \quad \mathbf{d_O} = sigmoid\left(\begin{vmatrix} 0.976 \\ 0.889 \\ 1.255 \end{vmatrix} \right) = \begin{vmatrix} 0.726 \\ 0.709 \\ 0.778 \end{vmatrix}$$

Разница **e** между желаемыми (**y**) и полученным ($\mathbf{d_O}$) выходом:

$$\mathbf{e} = \mathbf{y} - \mathbf{d_O} = \begin{vmatrix} 0.350 \\ 0.950 \\ 0.480 \end{vmatrix} - \begin{vmatrix} 0.726 \\ 0.709 \\ 0.778 \end{vmatrix} = \begin{vmatrix} -0.376 \\ 0.241 \\ -0.298 \end{vmatrix}$$

Ошибка сети:

$$\mathbf{E_{MSE}} = \frac{1}{3}\sum_{\text{i}} e_i^2 = \frac{(-0.376)^2 + 0.241^2 + (-0.298)^2}{3} = 0.096$$

С помощью полученной разности между желаемыми и реальными значениями выходов надо откалибровать весовые коэффициенты для улучшения нейронной сети.

Вычисление дельт для нейронов выходного слоя (см. формулу 3.10):

$$\boldsymbol{\delta_O} = (\mathbf{y} - \mathbf{d_O}) * f'(\mathbf{d_O}) = \mathbf{e} * f'(\mathbf{d_O}) = \begin{vmatrix} -0.376 \\ 0.241 \\ -0.298 \end{vmatrix} * \begin{vmatrix} (0.726 * (1 - 0.726)) \\ (0.709 * (1 - 0.708)) \\ (0.778 * (1 - 0.778)) \end{vmatrix} = \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix}$$

Вычисление дельт для нейронов скрытого слоя отличается отсутствием разности между выходными значениями этих нейронов и желаемыми значениями, поскольку желаемых значений нет и не может быть в принципе. Вычисление дельт производится по формуле (3.12). В матричных вычислениях первый член формулы представляет собой транспонированную матрицу весовых коэффициентов **W2**, умноженную на вектор дельт выходного слоя. Второй член формулы – по-прежнему производная функции активации от выходного значения скрытого нейрона:

$$\boldsymbol{\delta_H} = \left(\mathbf{W2}^{\mathrm{T}} \times \boldsymbol{\delta}_{\mathrm{O}}\right) * f'(\mathbf{d_H}) =$$

$$= \left(\begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix}^{\mathrm{T}} \times \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \right) * \begin{vmatrix} 0.761*(1-0.761) \\ 0.603*(1-0.603) \\ 0.650*(1-0.650) \end{vmatrix} = \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix}.$$

Вычисление градиента для каждого входа нейронов выходного и скрытого слоев осуществляется единообразно по формуле (3.13). Элементы матрицы градиентов выходного слоя определяются на основе векторов $\boldsymbol{\delta_O}$ и $\mathbf{d_O}$ следующим образом:

$$\mathbf{GRAD}_{\mathbf{W2}} = \boldsymbol{\delta_O} * \mathbf{d}_{\mathrm{H}}^{\mathrm{T}} =$$

$$= \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \times \begin{vmatrix} 0.761 \\ 0.603 \\ 0.650 \end{vmatrix}^{\mathrm{T}} = \begin{vmatrix} -0.075 \\ 0.050 \\ -0.051 \end{vmatrix} \times \begin{vmatrix} 0.761 & 0.603 & 0.650 \end{vmatrix} = \begin{vmatrix} -0.057 & -0.045 & -0.049 \\ 0.038 & 0.030 & 0.032 \\ -0.039 & -0.031 & -0.033 \end{vmatrix}.$$

Элементы матрицы градиентов скрытого слоя определяются на основе векторов $\boldsymbol{\delta_H}$ и **x**:

$$\mathbf{GRAD}_{\mathbf{W1}} = \boldsymbol{\delta_H} * \mathbf{x}^{\mathrm{T}} =$$

$$= \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix} \times \begin{vmatrix} 0.900 \\ 0.100 \\ 0.800 \end{vmatrix}^{\mathrm{T}} = \begin{vmatrix} -0.006 \\ -0.08 \\ -0.017 \end{vmatrix} \times \begin{vmatrix} 0.900 & 0.100 & 0.800 \end{vmatrix} = \begin{vmatrix} -0.006 & -0.001 & -0.005 \\ -0.007 & -0.001 & -0.006 \\ -0.015 & -0.002 & -0.013 \end{vmatrix}.$$

Для вычисления корректирующих значений весовых коэффициентов зададим параметр скорости обучения $\eta = 0{,}7$. Воспользуемся формулой:

$$\Delta \mathbf{W} = \mathbf{GRAD}_{\mathbf{W}} * \eta + \Delta \mathbf{W}(t-1) * \alpha . \qquad (4.1)$$

Но поскольку предыдущих итераций не было, о чём свидетельствует аргумент *t*-1, и предыдущие корректирующие значения еще не вычислялись, второе слагаемое с моментом α в формуле (4.1) будет нулевым.

$$\Delta_{\mathbf{W2}} = \begin{vmatrix} -0.057 & -0.045 & -0.049 \\ 0.038 & 0.030 & 0.032 \\ -0.039 & -0.031 & -0.033 \end{vmatrix} * 0{,}7 = \begin{vmatrix} -0.040 & -0.032 & -0.034 \\ 0{,}027 & 0{,}021 & 0.023 \\ -0.027 & -0.022 & -0{,}023 \end{vmatrix}$$

$$\Delta_{\mathbf{W1}} = \begin{vmatrix} -0.006 & -0.001 & -0.005 \\ -0.007 & -0.001 & -0.006 \\ -0.015 & -0.002 & -0.013 \end{vmatrix} * 0{,}7 = \begin{vmatrix} -0.004 & 0.000 & -0.003 \\ -0.005 & -0.001 & -0{,}004 \\ -0.011 & -0.001 & -0.009 \end{vmatrix}$$

Новые значения весовых коэффициентов получаются путем прибавления к старым значениям вычисленных поправочных значений:

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \Delta_{\mathbf{W}} \qquad (4.2)$$

$$\mathbf{W2} = \begin{vmatrix} 0.30 & 0.70 & 0.50 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.90 \end{vmatrix} + \begin{vmatrix} -0.040 & -0.032 & -0.034 \\ 0{,}027 & 0{,}021 & 0.023 \\ -0.027 & -0.022 & -0{,}023 \end{vmatrix} = \begin{vmatrix} 0.260 & 0.668 & 0.466 \\ 0.627 & 0.521 & 0.223 \\ 0.773 & 0.078 & 0.877 \end{vmatrix}$$

$$\mathbf{W1} = \begin{vmatrix} 0.90 & 0.30 & 0.40 \\ 0.20 & 0.80 & 0.20 \\ 0.10 & 0.50 & 0.60 \end{vmatrix} + \begin{vmatrix} -0.004 & 0.000 & -0.003 \\ -0.005 & -0.001 & -0{,}004 \\ -0.011 & -0.001 & -0.009 \end{vmatrix} = \begin{vmatrix} 0.896 & 0.000 & 0.397 \\ 0.195 & 0.799 & 0.196 \\ 0.089 & 0.499 & 0{,}591 \end{vmatrix}$$

Полученные значения весовых коэффициентов были использованы для того же самого входного вектора. При этом величина ошибки сети E_{MSE}=0.089 уменьшилась по сравнению с первоначальным значением (0.096).

Матричная запись алгоритма более компактна, так как она не зависит от размера сети. Что еще более важно, некоторые языки программирования могут работать с матрицами. Они могут эффективно и быстро посчитать произведение матриц вместо того, чтобы использовать циклы (*python*, встроенный язык *MatLab* и др.).

Вопросы для самоконтроля

1. Какие преимущества имеет матричный способ реализации алгоритма обратного распространения?
2. Чем отличаются операции «×» и «*» применительно к матрицам?
3. Где и почему в алгоритме используется операция транспонирования матриц?
4. Какие изменения произойдут в матричном алгоритме при изменении числа нейронов во входном слое, в скрытом слое, в выходном слое?
5. Как изменится алгоритм при добавлении второго скрытого слоя?

Тема 5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Библиотека линейной алгебры *numpy* языка *python* является очень удобным инструментом для реализации алгоритмов в области научных вычислений. Её *универсальные функции* предоставляют отличные решения для обучения нейронных сетей. Универсальная функция – это оболочка над обычной функцией, благодаря которой возможно выполнять определенные действия над скалярами, векторами и над целыми массивами. Данные функции выполняют поэлементные действия над массивами, поддерживают механизм транслирования, автоматически преобразуют типы данных и обеспечивают доступ к более тонким настройкам своей работы.

Ниже представлен список функций с кратким описанием в том контексте, в котором они используются в двух примерных программах:

- функция *array*() инициализирует массив входных данных в виде *numpy*-матрицы;
- функция *dot*() реализует скалярное произведение векторов, умножение матрицы на вектор, умножение матрицы на матрицу;
- функция *exp*() вычисляет экспоненту всех элементов массива;
- функция *square*() вычисляет квадрат элементов массива, т.е. каждый элемент массива умножается сам на себя;
- функция *subtract*() вычисляет поэлементную разность значений двух массивов;
- функция *mean*() вычисляет среднее арифметическое значений элементов массива.

Используемые операторы:

- оператор «.T» транспонирует *numpy*-матрицу;
- операторы «+, -, /» выполняют поэлементные операции сложения, вычитания и деления элементов векторов и матриц;
- оператор «*» выполняет поэлементные операции умножения векторов и матриц или умножение элементов векторов и матриц на скаляр.

Первая программа реализует матричный алгоритм обучения нейронной сети прямого распространения сигнала на обучающей выборке, содержащей три элемента:

входной вектор (**x**)	выходной вектор (**y**)
0.90	0.35
0.10	0.95
0.80	0.48

Архитектура сети и весовые коэффициенты матриц скрытого и выходного слоя совпадают с коэффициентами матриц из примера ручного расчёта (см. Тему 4). Каждое действие программы сопровождается выводом промежуточных результатов на экран. Все векторы для лаконичного представления выводятся транспонированными, т.е. в виде горизонтальной строки. Матрицы выводятся в традиционном представлении.

Программа_1.

```
import numpy as np

#================== ИСХОДНЫЕ ДАННЫЕ =================
# произвольные значения для входного и выходного векторов
x = np.array([[0.90, 0.10, 0.80]]).T
y = np.array([[0.35, 0.95, 0.48]]).T
print('====================================')
print('x: ',x.T)     #транспонируем вектор для лаконичного
print('y: ',y.T)     #вывода на экран **

# задание произвольных значений для весовых коэффициентов
# W1-Матрица весовых коэффициентов первого (скрытого) слоя
# W2-Матрица весовых коэффициентов второго (выходного) слоя
W1 = np.array([[0.9,0.3,0.4],[0.2,0.8,0.2],[0.1,0.5,0.6]])
W2 = np.array([[0.3,0.7,0.5],[0.6,0.5, 0.2],[0.8,0.1,0.9]])
print('W1:')
print(W1)
print('W2:')
print(W2)
```

```
# параметр скорости обучения
learn_rate =0.7
print('================================================')

#=================== ПРЯМОЙ ПРОХОД ==================
#расчёт выходных сигналов
L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
print('L1:',L1.T)                    #Выходы первого слоя
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
print('L2:',L2.T)                    #Выходы второго слоя

#расчёт средневадратической ошибки
MSE1=np.square(np.subtract(y, L2)).mean()
print('\nMSE =', MSE1)
print('================================================')

#=================== ОБРАТНЫЙ ПРОХОД ===============
#расчет абсолютной ошибки
Eabs=y-L2
print('E_abs:', Eabs.T)              #**

# расчет производные вых. сигналов второго и первого слоя
Deriv2=L2*(1-L2) #здесь 1 – единичная матрица
Deriv1=L1*(1-L1)

#расчет дельт второго и первого слоя
L2delta=Eabs*Deriv2
L1delta= np.dot( W2.T,L2delta,)*Deriv1
print('\nDelta_L2:', L2delta.T)       #**
print('Delta_L1:', L1delta.T)        #**

#расчет градиентов второго и первого слоя
GR2=L2delta.dot(L1.T)
GR1=L1delta.dot(x.T)
print('\nGrad_2:')
print(GR2.T)                          #**
print('Grad_1:')
print(GR1.T)                         #**

#расчет поправочных значений для весовых коэффициентов
deltaW2 = GR2*learn_rate
deltaW1 = GR1*learn_rate
```

```
print('\ndelta_W2:')
print(deltaW2.T)                        #**
print('delta_W1:')
print(deltaW1.T)                        #**

#расчет новых весовых коэффициентов
W1=W1+deltaW1
W2=W2+deltaW2
print('\nW1_new:')
print(W1)
print('W2_new:')
print(W2)

# ======= ПРОВЕРКА ОШИБКИ ПОСЛЕ 1-ГО ПРОХОДА =========
L1 = 1/(1+np.exp(-np.dot( W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE2=np.square(np.subtract(y, L2)).mean()
print('\nMSE_new =', MSE2)
print('=========================================')
```

Результат работы программы

На рис. 5.1 представлены входные и выходные векторы обучающей выборки и матрицы весовых коэффициентов первого (скрытого) и второго (выходного) слоя.

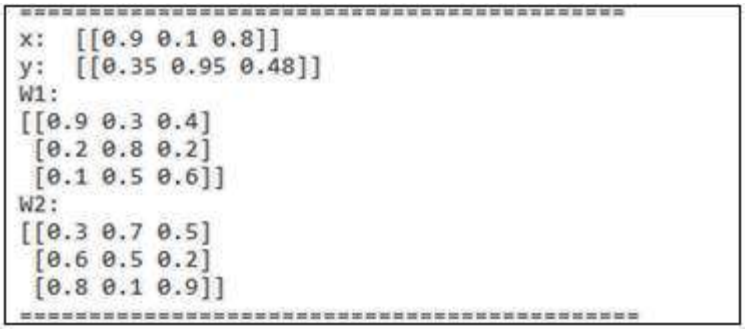


Рис. 5.1. Исходные данные алгоритма

Во время прямого прохода трех входных элементов обучающей выборки вычисляются выходные значения нейронов скрытого слоя L1 и нейронов выходного слоя L2. На основе реальных (L2) и желаемых (y) выходных значений вычисляется среднеквадратическая ошибка *MSE* на всей эпохе. Дополнительно выводятся абсо-

лютные ошибки сети для каждого из трех элементов обучающей выборки E_abs (рис. 5.2).

```
L1: [[0.76133271 0.60348325 0.65021855]]
L2: [[0.72630335 0.70859807 0.77809706]]

MSE = 0.09624698500823996
==========================================
E_abs: [[-0.37630335  0.24140193 -0.29809706]]
```

Рис. 5.2. – Результаты прямого прохода сигналов

На этапе обратного прохода происходит вычисление дельт каждого слоя сети Delta_L2 и Delta_L1. Эти вектора используются для расчёта матриц градиентов каждого синапса Grad_2 и Grad_1. Поскольку в каждом слое сети по три нейрона, а у каждого нейрона три входа, матрицы градиентов, как и матрицы весовых коэффициентов, имею размерности 3×3. Подстроечные значения весовых коэффициентов delta_W2 и delta_W1 получаются из матриц Grad_2 и Grad_1 умножением на константу скорости обучения learn_rate. Результаты промежуточных расчетов представлены на рисунке 5.3.

```
Delta_L2: [[-0.07480414  0.04984632 -0.05147004]]
Delta_L1: [[-0.00612519 -0.00779772 -0.01677461]]

Grad_2:
[[-0.05695084  0.03794964 -0.03918583]
 [-0.04514304  0.03008142 -0.03106131]
 [-0.04863904  0.032411   -0.03346678]]
Grad_1:
[[-0.00551267 -0.00701795 -0.01509715]
 [-0.00061252 -0.00077977 -0.00167746]
 [-0.00490015 -0.00623818 -0.01341969]]

delta_W2:
[[-0.03986559  0.02656475 -0.02743008]
 [-0.03160013  0.02105699 -0.02174292]
 [-0.03404733  0.0226877  -0.02342674]]
delta_W1:
[[-0.00385887 -0.00491256 -0.01056801]
 [-0.00042876 -0.00054584 -0.00117422]
 [-0.00343011 -0.00436672 -0.00939378]]
```

Рис. 5.3. Промежуточные результаты обратного прохода

Окончательные результаты обратного прохода – матрицы пересчитанных значений весов W1_new и W2_new. Их значения представлены на рис. 5.4. Для доказательства эффективности алго-

ритма производится прямой проход и вычисление ошибки сети с новыми весовыми коэффициентами. Уменьшенное значение ошибки MSE_new также выводится на экран.

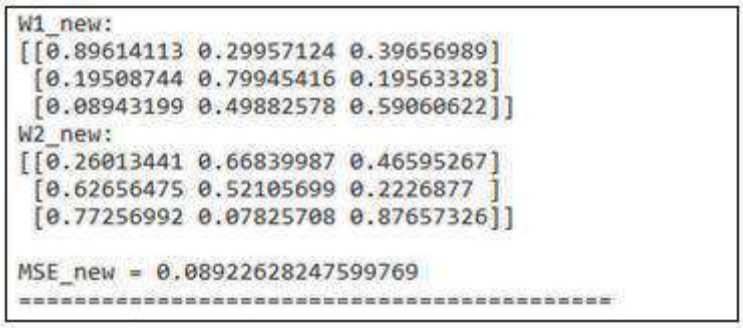


Рис. 5.4. Окончательные результаты обратного прохода и проверка эффективности произведенных вычислений

Вторая программа демонстрирует работу сети для трехмерных входных и выходных векторов. Обучение реализовано за счёт выполнения N=5000 циклов обучения. Печать промежуточных вычислений отсутствует. На экран выводятся только значения ошибок до и после обучения.

Для сокращения объёма кода во второй программе удалено вычисление ряда промежуточных переменных (векторов Eabs, Deriv1,2). Их расчёт перенесен в расчеты для основных переменных. Добавлено использование момента (momentum=0.3) при вычислении подстроечных значений коэффициентов. С этой цель введены массивы deltaW1_old и deltaW2_old. В них сохраняются значения delta_W1 и delta_W2 после каждой подстройки весовых коэффициентов сохраняются в массивах для использования на следующем шаге.

Программа выводит значения входного и выходного векторов обучающей выборки Input и Output, два значения ошибки – до и после обучения сети MSE и MSE_new, и значения, выдаваемые сетью на последнем шаге обучения Neural_Net_Output (рис. 5.5).

Программа_2.

```
import numpy as np
#================= ИСХОДНЫЕ ДАННЫЕ =================
# задание произвольных значений для входного (x)
# и выходного (y) векторов
x = np.array([[1, 0, 1], [0, 1, 1], [1, 1, 1]])
y = np.array([[1, 1, 1], [0, 0, 1], [1, 1, 0]])
# задание случайных значений для весовых коэффициентов
W1 = 2*np.random.random((3, 3))-1
W2 = 2*np.random.random((3, 3))-1
# параметры: скорость обучения и момент
learn_rate=0.7
momentum=0.3
# число эпох
N=5000

#расчёт выходных сигналов для вычисления начальной ошибки
L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE1=np.square(np.subtract(y, L2)).mean()
print('\nMSE =', MSE1)

deltaW2_old = 0
deltaW1_old=0

for j in range(N):
#=================== ПРЯМОЙ ПРОХОД ====================
#расчёт выходных сигналов
    L1 = 1 / (1 + np.exp(-np.dot(W1,x)))
    L2 = 1/(1+np.exp(-np.dot( W2,L1)))

#=================== ОБРАТНЫЙ ПРОХОД ====================
#расчет дельт второго и первого слоя
    L2delta=(y-L2)*(L2*(1-L2))
    L1delta= np.dot( W2.T,L2delta,)*L1*(1-L1)
#расчет градиентов второго и первого слоя
    GR2=L2delta.dot(L1.T)
    GR1=L1delta.dot(x.T)
#расчет поправочных значений для весовых коэффициентов
    deltaW2 = GR2*learn_rate + deltaW2_old*momentum
    deltaW1 = GR1*learn_rate + deltaW1_old*momentum
```

```
#расчет новых весовых коэффициентов
    W1=W1+deltaW1
    W2=W2+deltaW2
#сохранение поправочных коэффициентов текущего шага
    deltaW2_old=deltaW2
    deltaW1_old=deltaW1

# ======= ПРОВЕРКА ОШИБКИ ПОСЛЕ ОБУЧЕНИЯ  =========
L1 = 1/(1+np.exp(-np.dot( W1,x)))
L2 = 1/(1+np.exp(-np.dot( W2,L1)))
MSE2=np.square(np.subtract(y, L2)).mean()

print('\nInput:')
print(x)
print('\nOutput:')
print(y)
print('\nNeural_Net_Output:')
print(L2)
print('\nCycles = ',N)
print('MSE_new = ',MSE2)
```

```
MSE = 0.2017459125377166
Input:
[[1 0 1]
 [0 1 1]
 [1 1 1]]
Output:
[[1 1 1]
 [0 0 1]
 [1 1 0]]
Neural_Net_Output:
[[0.99890246 0.99601532 0.99007858]
 [0.00897447 0.00681352 0.98491968]
 [0.99091445 0.99313959 0.01521282]]
Cycles =  5000
MSE_new = 9.232668049558024e-05
```

Рис. 5.5. Результаты работы программы_2

Анализ результатов показывает, что сеть научилась распознавать данные с высокой степенью точности. Среднеквадратическая ошибка обученной сети составляет 9.23e-5, а максимальная абсо-

лютная ошибка (разница между элементами матриц Output и Neural_Net_Output) составляет 1.5%.

Вопросы для самоконтроля

1. Дайте краткую характеристику языку *python*. В каких областях рекомендуется его применение?
2. Каково назначение библиотеки *numpy*? Чем оправдано её применение для обучения нейронных сетей?
3. Что представляют собой массивы *numpy*?
4. Из каких переменных программы_1 можно получить представление об архитектуре нейронной сети?
5. Поясните, как в программном коде вычисляется ошибке сети на эпохе.
6. Чем обусловлено добавление момента при обучении сети в программе_2?
7. Проведите эксперименты с программой, закомментировав участок кода, осуществляющий использование момента, и поясните разницу в полученных результатах.
8. Проведите сравнение результатов программы_2 при одном и том же числе циклов, но с разными значениями скорости обучения. Какие выводы Вы можете сделать?

Тема 6. СЕТИ КОХОНЕНА

Термин «сети Кохонена» (англ.: *Kohonen network, KCN*) был введен в 1982 г. финским ученым Тойво Кохоненом. Хотя изначально они применялись для обработки изображений и звука, в настоящее время сети Кохонена являются эффективным средством кластерного анализа, когда требуется обнаружить скрытые закономерности в больших массивах данных.

Сеть Кохонена состоит из узлов-нейронов, которые в результате обучения сдвигаются в области, где сосредоточены исследуемые объекты. На рисунке 6.1 представлена архитектура сети Кохонена, каждый нейрон которой имеет два входа – по числу признаков или свойств кластеризуемых объектов.

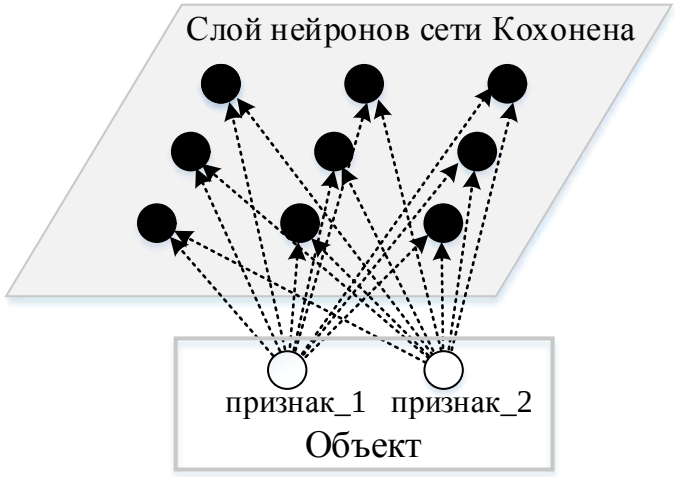


Рис. 6.1. Архитектура сети Кохонена

В отличие от большинства нейронных сетей, сеть Кохонена не имеет скрытых слоев: данные с входного слоя передаются непосредственно на единственный и он же выходной слой сети. Как и в обычной нейронной сети, элементы входного слоя не являются нейронами. Их задача просто передать значения входных полей исходной выборки на нейроны выходного слоя. Нейроны выходного слоя называются *кластерными элементами*, их количество определят максимальное количество групп, на которые сеть может разделить входные данные. Увеличивая количество нейронов, можно увеличивать детализацию результатов кластеризации. Исследователю следует знать, что могут иметь место ситуации, когда часть нейронов окажется незадействованной, либо несколько нейронов

диагностируют объекты одного и того же реально существующего кластера.

Каждая связь между входными элементами и нейронами сети имеет некоторый вес, который устанавливается в процессе инициализации. В большинстве случаев эти значения устанавливаются случайным образом, хотя существуют и другие способы инициализации. Значение весов, как правило, небольшие и лежат в интервале от 0 до 1.

Процесс обучения сети Кохонена, как и сетей прямой передачи сигнала, заключается в подстройке весов. В основе построения сети Кохонена лежит *конкурентное обучение*, при котором выходные узлы (нейроны) конкурируют между собой за право стать «победителем». Говорят, что нейроны *избирательно настраиваются* для различных входных примеров или классов входных примеров в ходе соревновательного процесса обучения. Побеждает тот нейрон, чей вектор весов ближе всего к входному вектору сигналов. Весовые коэффициенты нейрона-победителя подстраиваются таким образом, чтобы он стал ещё ближе к текущему объекту.

Исходные значения признаков объектов должны быть предварительно пронормированы. Это следует делать для того, чтобы признаки с большими абсолютными значениями не преобладали над признаками, величины которых малы.

Алгоритм обучения Кохонена

Пусть имеется набор из n экземпляров m-мерных векторов $\mathbf{x}$ исходной выборки. Каждый вектор $\mathbf{x}_i$ содержит оцифрованный набор признаков исследуемых объектов $\mathbf{x}_i=\{x_{i1}, x_{i2}, \ldots, x_{im}\}$, $i=1..n$. Векторы весов $\mathbf{w}_j$ каждого из q нейронов сети Кохонена тоже имеют размерность m: $\mathbf{w}_j=\{w_{1j}, w_{2j}, \ldots, w_{mj}\}$, $j=1..q$. За меру близости двух векторов можно взять квадрат евклидова расстояния:

$$\rho(\mathbf{x1}, \mathbf{x2}) = \sum_{j=1}^{m} (x1_j - x2_j)^2 . \qquad (6.1)$$

На каждом шаге обучения из исходного набора данных случайно выбирается один вектор. Затем производится поиск нейрона выходного слоя, для которого расстояние между его вектором весов и входным вектором минимально.

В обучении по Кохонену сам нейрон-победитель и нейроны, которые являются соседями нейрона-победителя, подстраивают свои веса, используя линейную комбинацию входных векторов и текущих векторов весов:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_{ni} - w_{ij}(t)), \qquad (6.2)$$

где $0<\eta\leq 1$ – коэффициент скорости обучения; индексы t, t+1 обозначают текущее и новое значение весового коэффициента.

Кохонен показал, что скорость обучения должна быть уменьшающейся функцией от числа итераций обучения. Согласно его рекомендациям, процесс обучения сети можно разделить на две фазы – фазу точной и фазу грубой подстройки. На первой фазе скорость обучения велика, и веса нейронов корректируются на большие величины, что позволяет приблизительно настроить их в соответствии с распределением значений признаков исследуемых объектов. Это так называемая грубая кластеризация. В процессе точной настройки скорость обучения резко уменьшается, что позволяет подстраивать веса на каждом шаге более точно.

По завершении эпохи вычисляется значение критерия останова алгоритма обучения. Если критерий не достигнут, то осуществляется переход к новой эпохе. В качестве критериев останова процесса обучения можно использовать следующие:

– количество полных циклов обучения ограничено константой, например, количество циклов равно количеству элементов во входном множестве;

– выход сети стабилизируется, т.е. входные вектора не переходят между кластерными элементами;

– изменения весов становятся незначительными.

В целях повышения эффективности алгоритма нейроны, которые слабо участвуют в конкуренции (не получают достаточного количества побед), могут быть сокращены.

Рассмотрим процесс обучения на примере сети Кохонена размером 2×2, предназначенной для кластеризации объектов с двумя признаками. Случайные начальные значения весовых коэффициентов:

$w_{1,1} = 0.9$ $w_{2,1} = 0.8$ (нейрон 1)
$w_{1,2} = 0.9$ $w_{2,2} = 0.2$ (нейрон 2)
$w_{1,3} = 0.1$ $w_{2,3} = 0.8$ (нейрон 3)
$w_{1,4} = 0.1$ $w_{2,4} = 0.2$ (нейрон 4)

В качестве объектов выступают абоненты, потребляющие трафик сети Интернет. Первый признак – возраст, второй – месячный объем трафика. Значения признаков пронормированы:

$x_{1,1} = 0.8$ $x_{1,2} = 0.8$ (пожилой человек, высокое потребление)
$x_{2,1} = 0.8$ $x_{2,2} = 0.1$ (пожилой человек, низкое потребление)
$x_{3,1} = 0.2$ $x_{3,2} = 0.9$ (молодой человек, высокое потребление)
$x_{4,1} = 0.1$ $x_{4,2} = 0.1$ (молодой человек, низкое потребление)

Для упрощения расчётов радиус обучения примем равным нулю ($\sigma = 0$), поэтому возможность подстройки весов будет предоставлена только нейрону-победителю. Коэффициент скорости обучения установим $h = 0{,}5$.

Вычисляем евклидово расстояние между входным вектором $\mathbf{x}_1 = (0.8; 0.8)$ и векторами весов всех четырех нейронов выходного слоя.

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_1) = \sqrt{(w_{11} - x_{11})^2 + (w_{21} - x_{12})^2} = \sqrt{(0.9 - 0.8)^2 + (0.8 - 0.8)^2} = 0.10 \quad \text{(min)}$$

$$D(\mathbf{w}_2, \mathbf{x}_1) = \sqrt{(w_{12} - x_{11})^2 + (w_{22} - x_{12})^2} = \sqrt{(0.9 - 0.8)^2 + (0.2 - 0.8)^2} = 0.61$$

$$D(\mathbf{w}_3, \mathbf{x}_1) = \sqrt{(w_{13} - x_{11})^2 + (w_{23} - x_{12})^2} = \sqrt{(0.1 - 0.8)^2 + (0.8 - 0.8)^2} = 0.70$$

$$D(\mathbf{w}_4, \mathbf{x}_1) = \sqrt{(w_{14} - x_{11})^2 + (w_{24} - x_{12})^2} = \sqrt{(0.1 - 0.8)^2 + (0.2 - 0.8)^2} = 0.92$$

Для первого объекта победителем стал нейрон с номером 1, для которого расстояние между вектором его весов $\mathbf{w}_1 = (0.9; 0.8)$ и входным вектором $\mathbf{x}_1 = (0.8; 0.8)$ оказалось минимальным. Видно, что вектор весов нейрона-победителя более «похож» на входной вектор $\mathbf{x}_1$, чем векторы весов других нейронов. Можно предположить, что нейрон 1 «открывает» кластер, в котором будут содержаться пожилые люди с высоким объёмом потреблённого трафика.

Подстройка:

Согласно формуле (6.2) для первого нейрона имеем:

$$w_{11}(t+1) = w_{11}(t) + 0.5 \times (x_{11} - w_{11}(t)) = 0.9 + 0.5 \times (0.8 - 0.9) = 0.85$$

$$w_{21}(t+1) = w_{21}(t) + 0.5 \times (x_{12} - w_{21}(t)) = 0.8 + 0.5 \times (0.8 - 0.8) = 0.8$$

Видим, что весовые коэффициенты нейронов «подталкиваются» в направлении значений входного вектора, т.е. w_{11} нейрона-победителя, соответствующий признаку «возраст», изначально был равен 0,9, но в результате подстройки был изменен в сторону нормированного значения «возраст» первого входного вектора, равного 0,8. Поскольку коэффициент скорости обучения h = 0,5, то величина подстройки равна половине расстояния между текущим весом и значением поля. Данная подстройка позволит нейрону 1 стать более «успешным» и в «захвате» других возможных объектов, содержащих информацию о пожилых людях с высокой интернет-активностью.

Второй входной вектор $\mathbf{x}_2$ = (0.8; 0.1).

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_2) = \sqrt{(w_{11} - x_{21})^2 + (w_{21} - x_{22})^2} = \sqrt{(0.85 - 0.8)^2 + (0.8 - 0.1)^2} = 0.70$$

$$D(\mathbf{w}_2, \mathbf{x}_2) = \sqrt{(w_{12} - x_{21})^2 + (w_{22} - x_{22})^2} = \sqrt{(0.9 - 0.8)^2 + (0.2 - 0.1)^2} = 0.14 \text{ (min)}$$

$$D(\mathbf{w}_3, \mathbf{x}_2) = \sqrt{(w_{13} - x_{21})^2 + (w_{23} - x_{22})^2} = \sqrt{(0.1 - 0.8)^2 + (0.8 - 0.1)^2} = 0.99$$

$$D(\mathbf{w}_4, \mathbf{x}_2) = \sqrt{(w_{14} - x_{21})^2 + (w_{24} - x_{22})^2} = \sqrt{(0.1 - 0.8)^2 + (0.2 - 0.1)^2} = 0.71$$

Подстройка:

$$w_{12}(t+1) = w_{12}(t) + 0.5 \times (x_{21} - w_{12}(t)) = 0.9 + 0.5 \times (0.8 - 0.9) = 0.85$$

$$w_{22}(t+1) = w_{22}(t) + 0.5 \times (x_{22} - w_{22}(t)) = 0.2 + 0.5 \times (0.1 - 0.1) = 0.15$$

Третий входной вектор $\mathbf{x}_3$ = (0.2; 0.9).

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_3) = \sqrt{(w_{11} - x_{31})^2 + (w_{21} - x_{32})^2} = \sqrt{(0.85 - 0.2)^2 + (0.8 - 0.9)^2} = 0.66$$

$$D(\mathbf{w}_2, \mathbf{x}_3) = \sqrt{(w_{12} - x_{31})^2 + (w_{22} - x_{32})^2} = \sqrt{(0.85 - 0.2)^2 + (0.15 - 0.9)^2} = 0.99$$

$$D(\mathbf{w}_3, \mathbf{x}_3) = \sqrt{(w_{13} - x_{31})^2 + (w_{23} - x_{32})^2} = \sqrt{(0.1 - 0.2)^2 + (0.8 - 0.9)^2} = 0.14 \text{ (min)}$$

$$D(\mathbf{w}_4, \mathbf{x}_3) = \sqrt{(w_{14} - x_{31})^2 + (w_{24} - x_{32})^2} = \sqrt{(0.1 - 0.2)^2 + (0.2 - 0.9)^2} = 0.71$$

Подстройка:

$$w_{13}(t+1) = w_{13}(t) + 0.5 \times (x_{31} - w_{13}(t)) = 0.1 + 0.5 \times (0.2 - 0.1) = 0.15$$
$$w_{23}(t+1) = w_{23}(t) + 0.5 \times (x_{32} - w_{23}(t)) = 0.8 + 0.5 \times (0.9 - 0.8) = 0.85$$

Четвёртый входной вектор $\mathbf{x}_4$ = (0.1; 0.1).

Конкуренция:

$$D(\mathbf{w}_1, \mathbf{x}_4) = \sqrt{(w_{11} - x_{41})^2 + (w_{21} - x_{42})^2} = \sqrt{(0.85 - 0.1)^2 + (0.8 - 0.1)^2} = 1.03$$
$$D(\mathbf{w}_2, \mathbf{x}_4) = \sqrt{(w_{12} - x_{41})^2 + (w_{22} - x_{42})^2} = \sqrt{(0.85 - 0.1)^2 + (0.15 - 0.1)^2} = 0.75$$
$$D(\mathbf{w}_3, \mathbf{x}_4) = \sqrt{(w_{13} - x_{41})^2 + (w_{23} - x_{42})^2} = \sqrt{(0.15 - 0.1)^2 + (0.85 - 0.1)^2} = 0.75$$
$$D(\mathbf{w}_4, \mathbf{x}_4) = \sqrt{(w_{14} - x_{41})^2 + (w_{24} - x_{42})^2} = \sqrt{(0.1 - 0.1)^2 + (0.2 - 0.1)^2} = 0.10 \text{ (min)}$$

Подстройка:

$$w_{14}(t+1) = w_{14}(t) + 0.5 \times (x_{41} - w_{14}(t)) = 0.1 + 0.5 \times (0.1 - 0.1) = 0.1$$
$$w_{24}(t) = w_{24}(t) + 0.5 \times (x_{42} - w_{24}(t)) = 0.2 + 0.5 \times (0.1 - 0.2) = 0.15$$

В таблице 6.1 представлены значения весов нейронов до и после первой подстройки. Конечно, движение нейронов к объектам достаточно очевидно. Однако, пример хорошо иллюстрирует основы работы алгоритма обучения сети Кохонена с использованием конкуренции между нейронами (рис. 6.2).

Таблица 6.1. Значения весовых коэффициентов до и после одной эпохи обучения

	Начальные значения		Значения после 1-й эпохи	
	w_1	w_2	w_1	w_2
Нейрон 1	0.9	0.8	0,85	0,80
Нейрон 2	0.9	0.2	0,85	0,15
Нейрон 3	0.1	0.8	0,15	0,85
Нейрон 4	0.1	0.2	0,10	0,15

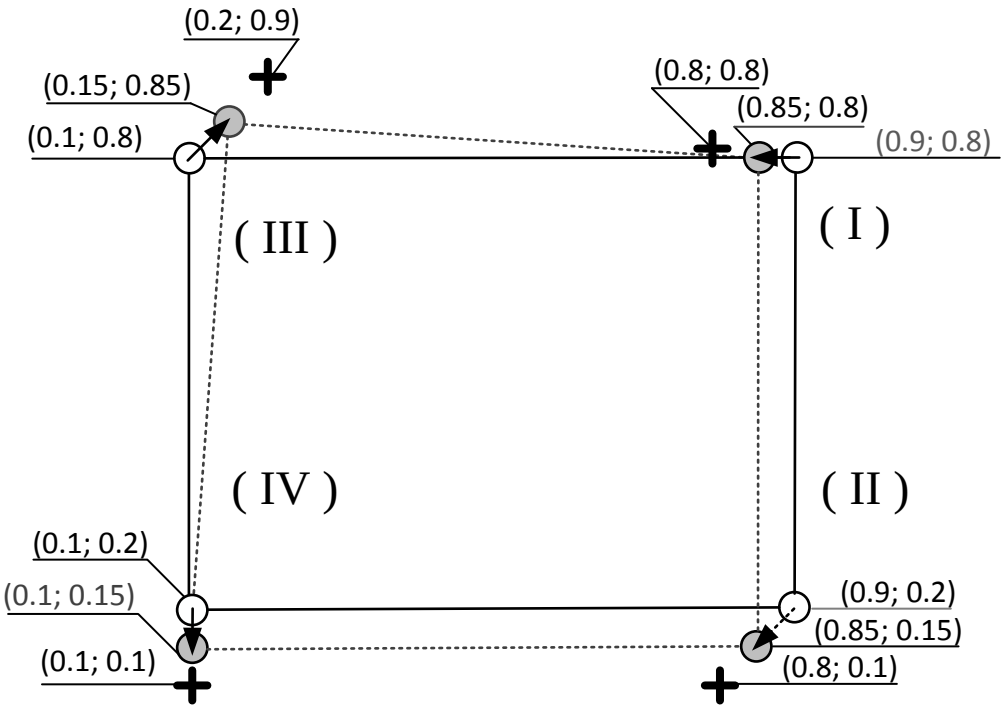


Рис. 6.2. Иллюстрация обучения сети Кохонена

Вопросы для самоконтроля

1. Каково принципиальное отличие в обучении сетей Кохонена от обучения сетей прямого распространения сигнала?
2. Для каких типов задач сети Кохонена являются наилучшим решением?
3. Какова архитектура сети Кохонена?
4. Что понимается под термином «нейрон-победитель»?
5. Опишите формально алгоритм обучения сети Кохонена.
6. Каковы параметры алгоритма? Каким образом они выбираются?
7. Какие фазы можно выделить в процессе обучения сети Кохонена?
8. С какой целью производится нормирование входных данных? Какие виды нормирования Вам известны?
9. Какие критерии останова могут использоваться для завершения алгоритма? Предложите свой критерий.

Тема 7. САМООРГАНИЗУЮЩИЕСЯ КАРТЫ КОХОНЕНА

Самоорганизующиеся нейронные карты (англ. s*elf organizing map, SOM*) или карты Кохонена – это разновидность нейронных сетей Кохонена. Они представляют собой не только эффективный инструмент кластеризации, но и не менее эффективный инструмент визуализации. В отличии от сети Кохонена, в которой число нейронов соответствует количеству формируемых кластеров, в *SOM* число нейронов заведомо больше предполагаемого числа кластеров.

Карты позволяют представлять результаты группировки объектов в виде двумерных карт, где расстояния между объектами соответствуют расстояниям между их векторами в многомерном пространстве, а сами значения признаков отображаются различными цветовыми оттенками.

Карта Кохонена состоит из сегментов прямоугольной или шестиугольной формы, называемых ячейками. Каждая ячейка связана с определенным нейроном и представляет собой «сферу влияния» или «область притяжения» данного нейрона. Шестиугольные ячейки позволяют более корректно отражать расстояния между объектами на карте, поскольку в этом случае расстояния между центрами смежных ячеек одинаковы, т.е. расстояние не зависит от направления (рис. 7.1). Чем больше число сегментов карты или её размер, тем более подробно можно представить распределение признаков объектов. Можно вернуться к обычной сети Кохонена, уменьшив число сегментов карты до нужного количества кластеров.

Подстройка весовых коэффициентов нейронов *SOM* получается так же, как и в обычной сети Кохонена, т.е. на основе конкурирующего обучения. Объекты, векторы признаков которых оказываются ближе к вектору весов данного нейрона, попадают в связанную с ним ячейку. Распределение объектов на карте в целом соответствует распределению векторов весов нейронов. И если объекты на карте расположены близко друг к другу, то и векторы признаков этих объектов близки и наоборот, если ячейки с объектами находятся далеко друг от друга, то и векторы их признаков различаются сильно.

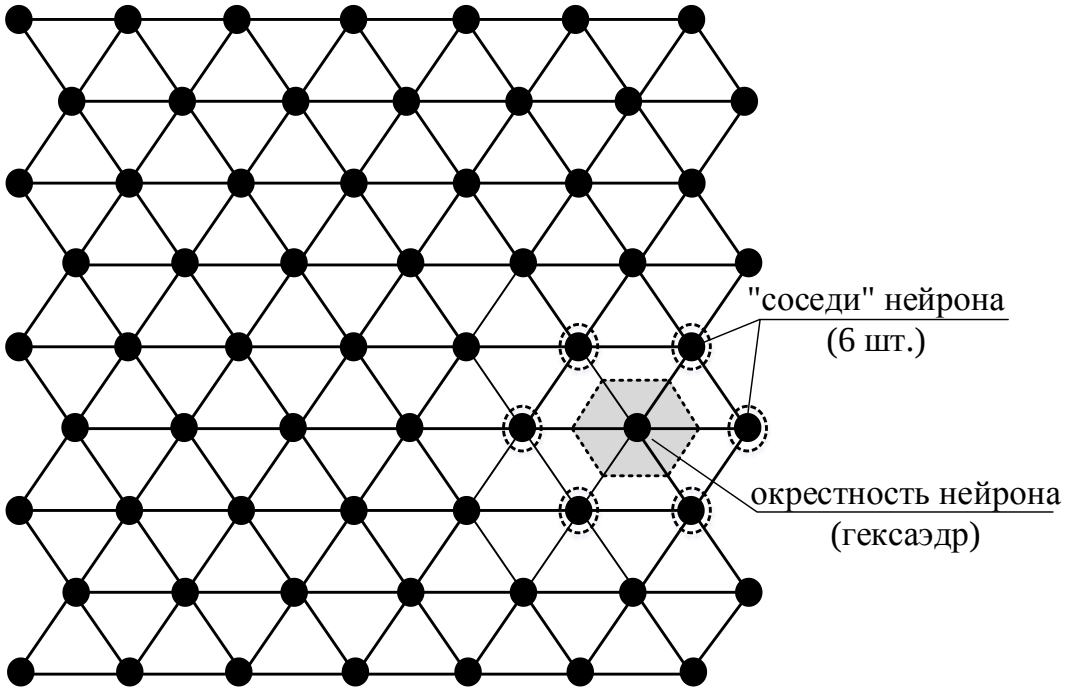


Рис. 7.1. Самоорганизующаяся карт размером 9×7

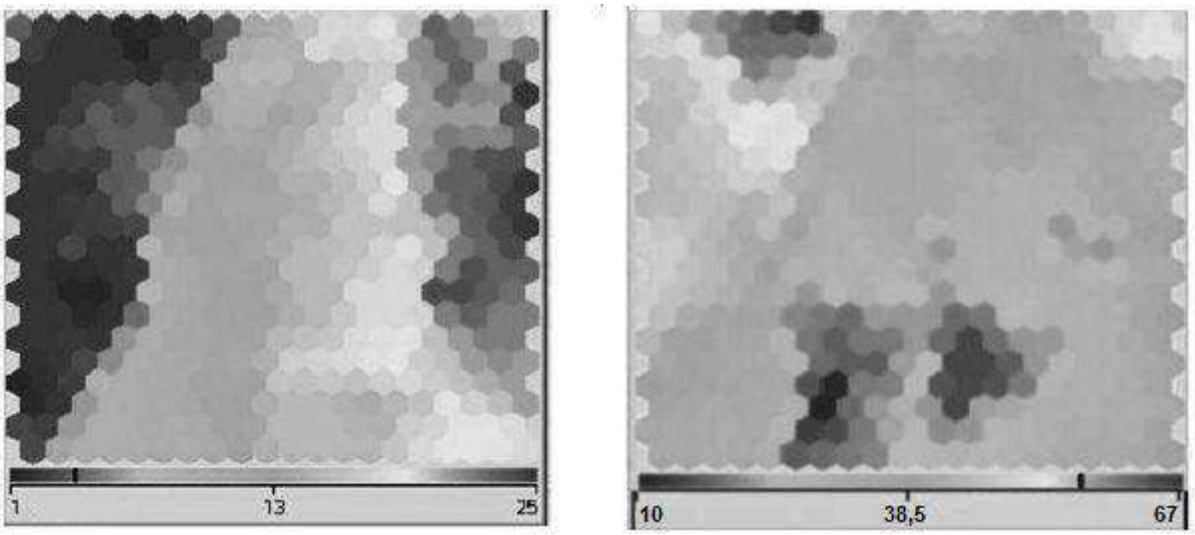
Отображение результатов обучения

Отобразить *SOM* целиком нельзя, поскольку её узлы-нейроны представляют собой точки в *n*-мерном признаковом пространстве. Но можно построить отдельные двумерные раскрашенные карты для каждого признака. Совокупность таких карт даёт интегральную картину, характеризующую набор исследуемых объектов. Специальная раскраска *SOM* выполняет функции третьего измерения и помогает понять, как распределены признаки объектов. Каждой ячейке (нейрону) на раскрашенной карте назначается цвет в соответствии со значениями признаков, попавших в неё объектов.

Раскраску карт можно осуществить разными способами. Например, если в ячейку попал только один объект, то её цвет будет определяться по одному значению признака. Если таких объектов несколько, цвет ячейки будет пропорционален среднему значению признака. Если центр ячейки – «мертвый» нейрон, для вектора весов которого не нашлось ни одного достаточно близкого вектора признаков объекта при обучении карты, то в ячейку не попадет ни одного объекта, и ее расцветка может формироваться на основе соответствующего веса нейрона, поскольку в процессе обучения нейрон наверняка «подтягивался» в ту или иную сторону и резкого

различия между цветом пустой ячейки и цветом соседних ячеек может и не быть. Либо у пустых ячеек выделяются границы, а сами ячейки заполняются какой-либо штриховкой (резко выделяются).

Раскраска карт позволяет оценивать и сами результаты кластеризации. Если ячейки с примерно одинаковой расцветкой образуют обособленные области, то результаты кластеризации хорошие, т.е. алгоритму удалось выделить группы объектов с похожими признаками. Если ячейки с разными цветами разбросаны вперемешку по всей карте, то результаты плохие. Хотя возможна ситуация, когда по одним признакам объекты различаются хорошо, а по другим хуже. Это может привести к тому, что карты, построенные по одним признакам, которые для разных групп объектов различаются хорошо, покажут хорошую кластеризацию, а по другим, которые различаются хуже, – плохую (рис. 7.2).



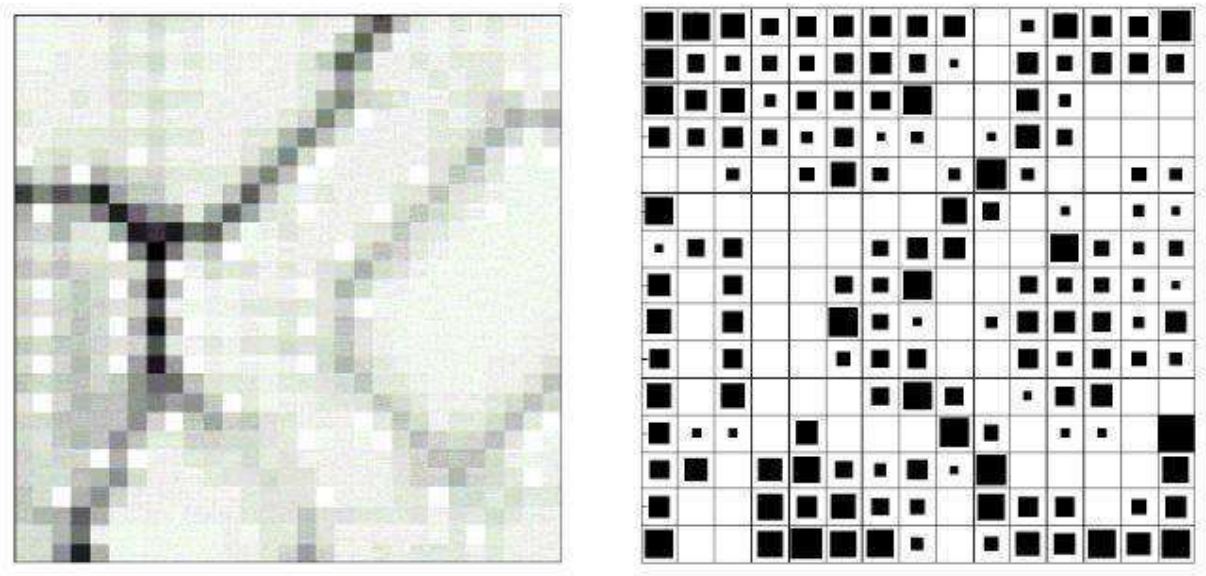
а *б*

Рис. 7.2. Примеры удачной (*а*) и неудачной (*б*) раскрасок

Автоматическое выделение кластеров на обученной карте может производиться на основе построения *унифицированной матрицы расстояний* (*U*-матрица), которая строится в оттенках серого и отображает расстояния между соседними узлами карты. В этом методе рассчитанное по евклидовой метрике расстояние между соседними нейронами представляется в виде различной раскраски между соседними вершинами. Темная раскраска соответствует большему расстоянию между нейронами, а светлая свидетельствует о близком соседстве (рис. 7.3, *а*). Светлые области могут считаться

кластерами, а темные – разделителями. Это может быть полезным представлением при попытке найти кластеры во входных данных без наличия априорной информации о кластерах.

Ещё одним популярным способом визуализации является *диаграмма Хинтона*. На каждом узле сетки изображается квадрат, размер которого пропорционален числу точек, ближайших к данному узлу, а оттенок соответствует значению соответствующего отображаемого признака (рис. 7.3 *б*). Чем больше точек попадает в узел, тем больше соответствующий размер квадрата. Оттенок квадрата соответствует значению цвета на матрице расстояний для данного нейрона. Таким образом диаграмма Хинтона соответствует матрице плотности попадания объектов в узлы карты.



а *б*

Рис. 7.3. Пример *U*-матрицы (*а*) и диаграммы Хинтона (*б*) для карт с прямоугольными ячейками.

Частота попадания объектов в область определённого нейрона карты может отображаться и на каждой эпохе обучения. Такие карты по своему виду схожи с диаграммами Хинтона, однако они отображают число реагирований нейронов-победителей на входные данные на каждой эпохе и в совокупности позволяют отслеживать динамику активности нейронов.

Выбор числа нейронов карты

Выбор числа ячеек в *SOM* зависит от особенностей решаемой задачи. Если нужна детальная информация по каждому объекту и его признакам, то количество нейронов карты должно приближаться к числу объектов выборки. Но увеличение числа нейронов карты приводит к росту вычислительных и временных затрат на ее обучение. В большинстве случаев для анализа представляют интерес обобщенные данные, когда сделанные на их основе выводы и заключения можно распространить на группы объектов. В этом случае нужно, чтобы число нейронов было в несколько раз меньше числа кластеризуемых объектов. Строгих правил для определения числа нейронов *SOM* не существует. Пользователь должен сам определить (эмпирически) оптимальное количество нейронов исходя из решаемой задачи и особенности анализируемых данных. Например, если разброс значений признаков в обучающей выборке сильный (т.е. векторы объектов в пространстве признаков разрежены), то, возможно, следует уменьшить число нейронов карты, чтобы избежать большого количества пустых ячеек. И наоборот, если векторы объектов расположены в пространстве признаков плотно, то для получения лучших результатов можно увеличить число нейронов.

Недостатки SOM

Одним основных недостатков является эвристический характер метода. В большинстве случаев начальная инициализация карты, т.е. задание начальных векторов весов нейронов, является произвольной и тогда возможна потеря однозначности результата. Если обучать карту несколько раз, то итоги будут различаться из-за того, что устанавливаемые случайным образом начальные веса нейронов будут различны и обучение пойдет несколько по-иному.

Второй недостаток – «мертвые» нейроны, веса которых были проинициализированы таким образом, что их векторы оказались в той части пространства признаков, где отсутствуют или почти отсутствуют векторы признаков объектов. Большое число мёртвых нейронов может привести к выполнению обработки входных данных меньшим числом нейронов, что ухудшит результаты кластеризации. В карте может не хватить активных нейронов для достоверной интерпретации исследуемых объектов.

Для устранения указанных недостатков, можно, во-первых, инициализировать карту не случайными весами, а векторами признаков объектов обучающей выборки, во-вторых, подсчитывать количество «побед» каждого нейрона с целью учета его активности. Если это число превышает некоторое значение (т.е. нейрон слишком активен), то чтобы дать возможность другим тоже стать победителями, он штрафуется искусственным завышением расстояния между вектором его весов и вектором весов объектов. При этом величина «штрафа» пропорциональна числу побед нейрона. После того как в процесс обучения вовлекается достаточное количество нейронов, можно продолжить конкуренцию без регуляризации, т.е. без штрафов.

Обучение SOM

В отличие от сети Кохонена, при настройке *SOM* активизируется не только нейрон-победитель, но и его соседи. Победивший нейрон становится центром некоторого множества «нейронов-соседей». Все нейроны такого соседства тоже обладают правом подстройки весов в сторону текущего объекта (рис. 7.4).

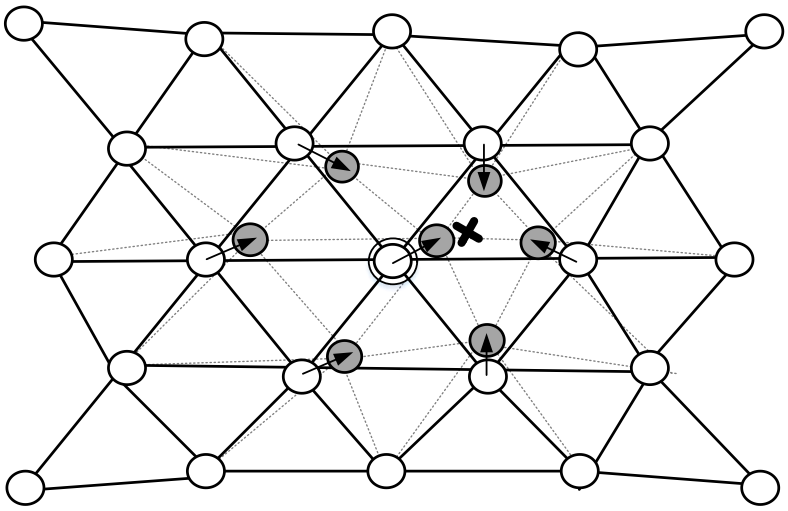


Рис. 7.4. Иллюстрация процесса подстройки сети Кохонена с учётом топологической окрестности

Их веса подстраиваются таким образом, чтобы в дальнейшем нейроны смогли увеличить свой шанс стать победителем для похожих наборов входных объектов. Иногда нейроны-соседи подстраиваются на меньшие величины по сравнению с нейроном-

победителем. В ряде случаев («совершенная» конкуренция) ситуация может быть прямо противоположной: нейрон-победитель получает минимальные или даже отрицательные значения для подстройки, а его соседи – существенно приближаются к входному объекту.

Наиболее употребимой функцией для определения радиуса топологической окрестности (или гиперсферы), в которую попадут нейроны-соседи, является функция Гаусса:

$$h(u,c,t) = \exp\left(-\frac{\rho(c,u)}{\sigma(t)}\right) \tag{7.1}$$

где u – номер нейрона в двумерной решетке слоя нейронов, для которого вычисляется значение h; c – номер нейрона-победителя; t – параметр времени.

Величина топологической окрестности σ нейрона-победителя может быть задана константой или функцией, которая линейно или экспоненциально стремится к нулю с увеличением параметра времени:

$$\sigma(t) = \frac{1}{\exp(t^{-2})} \tag{7.2}$$

Оптимальные значения параметров алгоритма обучения сети Кохонена – начальная скорость обучения η, радиус окрестности σ или закон его изменения также определяются исследователем для каждой предметной области на основе серии экспериментов.

Программная реализация

Ещё одна библиотека языка *python* – *tensorflow* – предназначена для реализации различных алгоритмов машинного обучения. Основной её целью является предоставление исследователю гибких, простых в использовании, но в то же время мощных инструментов для реализации задач из области построения и обучения нейронных сетей. Использование возможностей *tensorflow* позволяет решать задачи автоматического нахождения и классификации

образов, достигая качества человеческого восприятия. Функции библиотеки объединяют вычислительную алгебру и методы оптимизации для легкого вычисления многих математических выражений. Вычисления *tensorflow* выражаются в виде потоков данных через граф состояний. Название *tensorflow* происходит от операций с многомерными массивами данных, которые также называются «тензорами». Это объекты линейной алгебры, линейно преобразующие элементы одного линейного пространства в элементы другого. Частными случаями тензоров являются скаляры и векторы. Ранги таких тензоров d равны соответственно нулю или единице. В большинстве случаев тензор представляют собой многомерную таблицу размерности $d \times d \times ... \times d$, заполненную числами – компонентами тензора [10-12].

В приложении представлена программная система, состоящая из трёх модулей, для реализации карты Кохонена:

1. Модуль *model.py* содержит разработанный класс «*SOM*», в котором реализованы методы инициализации и обучения нейросети на основе технологии тензорных вычислений.
2. Модуль *grid.py* предназначен для создания и отображения раскрашенных карт.
3. Модуль *main.py* предназначен для обработки входных данных, определения оптимального значения параметра топологической окрестности нейронов-соседей и последующего старта процесса обучения карты.

Программная система обеспечивает выполнение следующих функций: подготовку данных к виду, пригодному для обработки нейронной сетью; инициализацию нейронной сети на основе технологии тензорных вычислений; вычисление значения параметра топологической окрестности для минимизации ошибки обобщения; последующее обучение сети с использованием оптимального σ. В результате проведённой серии экспериментов производится группировка входных объектов по совокупности признаков; наглядное представление результатов группировки; тестирование обученной нейронной сети для последующего соотнесения неизвестных ей объектов к сформированным кластерам.

В качестве данных для обучения используются реальные объекты – *SQL*-запросы, описываемые набором из 32-х параметров

производительности. При подготовке данных к виду, пригодному для обучения сети, и с целью сокращения размерности входного пространства используется метод главных компонент. Это позволяет перейти от 32-х мерного к 4-х мерному пространству.

Помимо библиотеки *tensorflow* для работы программы требуется установка библиотечных модулей *numpy* и *matplotlib*.

Вопросы для самоконтроля

1. Что понимается под картой Кохонена и чем она отличается от нейронной сети Кохонена?
2. Какие возможности реализуют нейронные карты?
3. Что понимается под раскраской карт? Как выполняется раскраска карт признаков кластеризуемых объектов?
4. Какие типы раскрасок отображают топологические особенности множества входных объектов? Как они создаются?
5. Какими факторами следует руководствоваться при определении числа узлов карты?
6. Что понимается под окрестностью нейрона? Какова её роль в обучении?
7. Формализуйте алгоритм обучения *SOM*.
8. Каковы параметры алгоритма обучения и каким образом они определяются?
9. Опишите достоинства и недостатки карты Кохонена.
10. Изучите программный код приложения для реализации самоорганизующейся карты и поясните каким образом тензорные вычисления оптимизируют процесс создания и обучения *SOM*?
11. Как в программной системе осуществляется подготовка данных для создания и обучения карты?
12. Опишите процесс определения оптимального значения топологической окрестности.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Варианты заданий по теме 1

1. Битовые образы – буквы русского алфавита размером 3×4 в количестве 5 первых букв фамилии студента.
2. Битовые образы – четные арабские цифры размером 4×4: 0,2,4,6,8.
3. Битовые образы – нечетные арабские цифры размером 4×4: 01,3,5,7,9.
4. Битовые образы – буквы латинского алфавита размером 3×4 в количестве 5 первых букв имени студента.
5. Битовые образы – графические изображения символов верхнего ряда клавиатуры размером 3×4 в количестве (!, #, *, и т.п.).
6. Битовые образы – буквы греческого алфавита размером 4×5 в количестве 5 шт.

Варианты заданий по теме 2

Рассчитать ошибку нейронной сети (рис. 2.1) на обучающей выборке, представляющей собой таблицу истинности логической функции двух аргументов. Определить минимальную и максимальную ошибку за эпоху. Весовые коэффициенты выбрать самостоятельно, основываясь на личных данных студента. Например, w_1=0,10 w_2=–0,30 w_3= 0,20 w_4=–0, 01 w_5=0,23 w_5=0,49 (дата рождения студента число, месяц, год – 10.03.2001, последние цифры зачетки - 2349).

1. Ошибка MSE, логическая функция – XOR.
2. Ошибка RMSE, логическая функция – XOR.
3. Ошибка Arctan, логическая функция – XOR.
4. Ошибка MSE, логическая функция – OR.
5. Ошибка RMSE, логическая функция – OR.
6. Ошибка Arctan, логическая функция – OR.
7. Ошибка MSE, логическая функция – штрих Шефера.
8. Ошибка RMSE, логическая функция - штрих Шефера.
9. Ошибка Arctan, логическая функция - штрих Шефера.
10. Ошибка MSE, логическая функция – стрелка Пирса.
11. Ошибка RMSE, логическая функция – стрелка Пирса.
12. Ошибка Arctan, логическая функция – стрелка Пирса.

Варианты заданий по теме 3

1. Рассчитать величину дельта на основе посчитанной ранее ошибки для одного (любого) элемента обучающей выборки.
2. Подстроить весовые коэффициенты сети.
3. Провести расчет ошибки для этого же элемента обучающей выборки на сети с новыми весовыми коэффициентами.
4. Выполнить пункты 1-3 для всей эпохи.
5. Обосновать полученный результат.

Сеть, её начальные значения весовых коэффициентов и обучающая выборка те же, что и в предыдущем задании. Значение скорости обучения задать самостоятельно.

Варианты заданий по теме 4

Провести ручной расчет первоначальной ошибки, подстройку коэффициентов сети и новое значение ошибки с применением матричных операций на собственном наборе входных данных. Архитектура сети: 3 входа, 3 выхода, 3 нейрона в скрытом слое.

Варианты заданий по теме 5

Провести программный расчет ошибки и коррекцию весовых коэффициентов нейронной сети для собственного набора входных данных. Вычислить ошибку новой сети на тех же входных данных.

Архитектура сети:

1. 2 входа, 1 выход, 1 скрытый слой с 2-мя нейронами.
2. 2 входа, 1 выход, 1 скрытый слой с 3-мя нейронами.
3. 2 входа, 1 выход, 1 скрытый слой с 4-мя нейронами.
4. 2 входа, 2 выхода, 1 скрытый слой с 2-мя нейронами.
5. 2 входа, 2 выхода, 1 скрытый слой с 3-мя нейронами.
6. 3 входа, 1 выход, 1 скрытый слой с 2-мя нейронами.
7. 3 входа, 1 выхода, 1 скрытый слой с 3-мя нейронами.
8. 3 входа, 2 выхода, 1 скрытый слой с 2-мя нейронами.
9. 3 входа, 2 выхода, 1 скрытый слой с 3-мя нейронами.
10. 3 входа, 3 выхода, 1 скрытый слой с 2-мя нейронами.
11. 3 входа, 3 выхода, 1 скрытый слой с 2-мя нейронами.
12. 3 входа, 3 выхода, 1 скрытый слой с 3-мя нейронами.

Начальные значения весовых коэффициентов сети задать случайным образом. Векторы входных/выходных значений, скорость

обучения и момент сформировать на основе личных данных студента.

Варианты заданий по теме 6

Произвести расчет изменения весовых коэффициентов нейронной сети Кохонена на самостоятельно созданном обучающем наборе. Значения признаков пронормировать. Для ускорения расчётов следует воспользоваться возможностями табличного процессора *Excel*. Параметры для построения сети Кохонена – размер сети; количество признаков объекта и предполагаемое количество кластеров для каждого варианта заданы в таблице.

Таблица Варианты заданий

№	Параметры задачи	№	Параметры задачи
1.	2×2, 2 признака; 4 кластера	7.	3×2; 2 признака; 6 кластеров
2.	2×2; 2 признака; 3 кластера	8.	3×2; 2 признака; 4 кластера
3.	2×2; 2 признака; 2 кластера	9.	3×2; 2 признака; 2 кластера
4.	2×2; 3 признака; 4 кластера	10.	3×2; 3 признака; 6 кластера
5.	2×2; 3 признака; 3 кластера	11.	3×2; 3 признака; 4 кластера
6.	2×2; 3 признака; 2 кластера	12.	3×2; 2 признака; 2 кластера

Примеры предметных областей для создания обучающих наборов: абоненты ресурсоснабжающей организации; объекты растительного и животного мира; учащиеся учебных заведений; объекты купли-продажи; клиенты финансовых организаций; пользователи социальных сетей; объекты социально-политической деятельности и др.

Варианты заданий по теме 7

1. Подготовьте данные для обучения карты Кохонена на основе примера, приведённого в программном коде. В качестве входных данных используйте объекты из предыдущего задания.

2. Запустите программу для получения оптимального значения ширины топологической окрестности.

3. Обучите новую карту с использованием оптимального параметра. Проанализируйте сформированные программой раскрашенные карты. Сделайте выводы о распределении объектов.

ЛИТЕРАТУРА

1. Мак-Каллок У.С., Питтс У. Логическое исчисление идей, относящихся к нервной активности // «Автоматы» под ред. К.Э.Шеннона и Дж. М.Маккарти: 1956. - С. 362 - 384. [Электронный ресурс]: http://www.raai.org/library/books/mcculloch/mcculloch.pdf

2. Кузнецова А.В., Мохов В.А. Нечеткая логика и нейронные сети в наукоемких производствах: учеб. Пособие: Новочеркасск: ЮРГТУ (НПИ), 2011. -142 с.

3. Нейронные сети. Statistica Neural Networks: Методология и технологии современного анализа данных / под редакцией В.П. Боровикова. - 2-е изд., перераб. и доп. – М.: Горячая линия-Телеком, 2008. - 392 с.

4. Яхъяева Г.Э. Нечеткие множества и нейронные сети : учеб. пособие для вузов. – М.: БИНОМ. Лаборатория знаний : Интернет-Ун-т Информ. технологий, 2006. - 316 с.

5. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы: пер. с польск. / Д. Рутковская, М. Пилиньский, Л. Рутковский. – М.: Горячая линия-Телеком, 2006. - 452 с.

6. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика / 2-е изд. – М.: Горячая линия-Телеком, 2002. - 382 с.

7. Нейронные сети: STATISTICA Neural Networks : пер. с английского. – М.: Горячая линия-Телеком, 2000. - 182 с.

8. Сузи Р. А. Язык программирования PYTHON: учебное пособие – М.: БИНОМ. Лаборатория знаний: Интернет-Ун-т Информ. технологий, 2006. - 326 с

9. Уроки по программированию, DevOps и другим IT-технологиям: Учебники/Машинное обучение. [Электронный ресурс]: https://coderlessons.com/tutorials

10. Официальный сайт TensorFlow. [Электронный ресурс]: https://www.tensorflow.org/

11. Справочник Википедии по TensorFlow. [Электронный ресурс]: https://en.m.wikipedia.org/wiki/TensorFlow

12. Ramsundar B., Zadeh R.B. TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning // «O'Reilly Media», 1 edition, March 23, 2018. – 256 p.

ПРИЛОЖЕНИЕ

МОДУЛЬ model.py

```
#!/usr/bin/python
import tensorflow as tf
import numpy as np
import grid

class SOM:
  def __init__(self, Rows, Cols, Inputs, seed=None):
    # значение для инициалиации генератора псевдослучайных чисел
    self.Seed = seed

    # вычисление топологии SOM заданного размера
    self.grid = grid.Grid(Rows, Cols)

    # вычисление топологических расстояний между каждыми двумя элементами карты
    topodists = np.linalg.norm(np.expand_dims(self.grid.Centers, axis=2) -
                               np.expand_dims(self.grid.Centers, axis=1),axis = 0)

    # задание константы топологических расстояний в вычислительном графе
    self._TopoDists = tf.constant(topodists, name='Distances')

    # матрица ближайших соседей для каждого элемента
    # (0 - сам элемент или любой не соседний, 1 - ближайший соседний)
    self._NNN  = tf.constant((np.absolute(topodists - 1) < 0.001).astype(np.float32))
```

```
# задание константы координат элементов карты в вычислительном графе
self._Coords = tf.constant(self.grid.Centers.T,  dtype=tf.float32, name='Grid')

# входные узлы вычислительного графа:
#   Alpha  - скорость обучения,
#   Sigma  - параметр ширины функции соседства,
#   Input  - поэлементный вход самоорганизующейся карты
#   BatchedInput - пакетный вход самоорганизующейся карты
self._Alpha= tf.placeholder(shape=(None), dtype=tf.float32, name='Alpha')
self._Sigma= tf.placeholder(shape=(None), dtype=tf.float32, name='Sigma')
self._Input= tf.placeholder(shape=(Inputs), dtype=tf.float32, name='Input')
self._BatchedInput = tf.placeholder(shape=(None, Inputs), dtype=tf.float32,
                                    name='BatchedInput')

# переменная для хранения весовых коэффициентов(координат)
# элементов карты в пространстве входного множества
self._Weights= tf.Variable(tf.random_normal([Cols*Rows, Inputs], seed=self.Seed),
                           dtype=tf.float32, name='Weights')

# вычисление расстояние между текущим входным элементом
# и всеми элементами карты пространстве входных данных
self._DeltaW = self._Input - self._Weights
self._Dist = tf.norm(self._DeltaW, axis=1)
```

```
# получение индекса ближайшего элемента карты
self._BMU  = tf.argmin(self._Dist)

# то же, но для пакетного входа
self._BatchedDeltas = (tf.expand_dims(self._BatchedInput, axis = 1) -
                       tf.expand_dims(self._Weights,      axis = 0))

# индекс кластера для каждого входного элемента, предоставленного на пакетный вход
self._BatchedBMU = tf.argmin(tf.norm(self._BatchedDeltas, axis = 2), axis = 1)

# значение индекса кластера к которому принадлежит
# элемент входного мнодества в унитарном коде
self._OneHotBBMU = tf.one_hot(self._BatchedBMU,    axis=1, depth=Cols*Rows)

# количество элементов входного множества в каждом кластере
self._ClustHits = tf.reduce_sum(self._OneHotBBMU, axis=0, keepdims=True)

# соотнесения элементов входного множества с сответствующими им
# элементами самоорганизующейся карты
self._ClustBatchInp = tf.multiply(
                        tf.expand_dims(self._BatchedInput, axis = 1),
                        tf.expand_dims(self._OneHotBBMU,   axis = 2))

# вычислене ошибки квантования для каждого кластера
self._Qmap = tf.reshape(
```

```
            tf.divide(
              tf.reduce_sum(
                tf.norm(
                  tf.multiply(
                    self._ClustBatchInp -
                      tf.expand_dims(self._Weights,     axis = 0),
                        tf.expand_dims(self._OneHotBBMU, axis = 2)),
                    axis = 2),
                  axis = 0),
                self._ClustHits),
              shape=[-1])

# вычисление U-map для карты
self._Umap = tf.reduce_sum(
                    tf.multiply(
                      tf.norm(
                        tf.expand_dims(self._Weights, axis=1) -
                        tf.expand_dims(self._Weights, axis=0),
                          axis=2),
                        self._NNN),
                      axis=1) / tf.reduce_sum(self._NNN, axis=1)

# вычисление маски интерполяции
self._InterpolationMask = tf.multiply(
                            tf.cast(
```

```
                                    tf.abs(
                                      self._TopoDists -
                                      tf.reduce_min(
                                        tf.boolean_mask(
                                          self._TopoDists,
                                          tf.reshape(self._ClustHits, shape=[-1]) > 0),
                                        axis = 0,
                                        keepdims = True )) < 0.001,
                                    dtype=tf.float32),
                                  tf.cast(
                                    tf.transpose(self._ClustHits) > 0,
                                    dtype=tf.float32))

    # вычисление среднего для каждого кластера
    self._ClusterMean   = tf.divide(
                                  tf.reduce_sum(self._ClustBatchInp, axis = 0),
                                  tf.transpose(self._ClustHits))

    # вычисление дисперсии для каждого кластера
    self._ClusterVariance   = tf.divide(
                                  tf.reduce_sum(
                                    tf.square(
                                      tf.multiply(
                                        self._ClustBatchInp - self._ClusterMean,
                                        tf.expand_dims(self._OneHotBBMU, axis=2))),
```

```
                                axis = 0), tf.transpose(self._ClustHits))

        # интерполяция средних кластеров с ненулевы количеством элементов
        # входного множества на кластеры с нулевым количеством элементов
        self._ClustMeanInerpol  = tf.divide(
                                    tf.reduce_sum(
                                      tf.multiply(
                                        tf.expand_dims(self._InterpolationMask, axis=2),
                                        tf.expand_dims(
                                          tf.where(
                                            tf.is_nan(self._ClusterMean),
                                            tf.zeros_like(self._ClusterMean),
                                            self._ClusterMean),
                                          axis=1)), axis=0),
                                    tf.transpose(
                                      tf.reduce_sum(
                                        self._InterpolationMask, axis=0, keepdims=True)))

        # интерполяция дисперсий кластеров с ненулевы количеством элементов
        # входного множества на кластеры с нулевым количеством элементов
        self._ClustVariInerpol  = tf.divide(
                                    tf.reduce_sum(
                                      tf.multiply(
                                        tf.expand_dims(self._InterpolationMask, axis=2),
```

```
                                    tf.expand_dims(
                                      tf.where(
                                        tf.is_nan(self._ClusterVariance),
                                        tf.zeros_like(self._ClusterVariance),
                                        self._ClusterVariance),
                                      axis=1 )), axis=0),
                                  tf.transpose(
                                    tf.reduce_sum(
                                      self._InterpolationMask, axis=0, keepdims=True)))

    # вычиление значений функции соседства для всех элементов карты
    self._Neighbourhood = tf.multiply(
                              self._Alpha,
                              tf.exp(
                                tf.negative(
                                  tf.square(
                                    tf.norm( tf.subtract(self._Coords,
                                                         self._Coords[self._BMU, :]),
                                      axis=1, ord='euclidean', keepdims=True) ) /
                                    (2 * (self._Sigma ** 2)) )),
                                name='Neighbourhood')

    # вычисление значений для обновления весов карты
    self._WUpdDelta = tf.multiply(self._DeltaW, self._Neighbourhood)
```

```
  # обновление весов карты
  self._Train = tf.assign_add(self._Weights,self._WUpdDelta, name='Train')

  # создание сесси для исполнения вычислительного графа и инициализация
  # переменных (весов карты)
  self._Session = tf.Session()
  tf.global_variables_initializer().run(session=self._Session)

# метод для обучения карты
def train( self, EpochsNum, Alpha, train_ds, validation_ds=None,
           sigma_func  = lambda x, x_max, a, b: a-(a-b)*(x/x_max),
           sigma_start = 50, sigma_end   = 0 ):

# обнуление векторов ошибок
  GenErr = []
  Qerr = []

  # цикл обучения карты
  for epoch in range(0, EpochsNum):
    for i in train_ds:
      Ws = self._Session.run( self._Train,
             feed_dict={self._Input: i, self._Alpha: Alpha,
                        self._Sigma: sigma_func(epoch, EpochsNum,
                                                sigma_start, sigma_end) })
```

```
    # вычисление ошибки обобщения для карты на текущей эпохе обучения
    if validation_ds is not None:
      TrMeans, TrVars, Qmap = self._Session.run(
                          [self._ClustMeanInerpol, self._ClustVariInerpol, self._Qmap],
                            feed_dict={self._BatchedInput: train_ds} )

      VlMeans, VlVars = self._Session.run([self._ClustMeanInerpol,
                self._ClustVariInerpol], feed_dict={self._BatchedInput: validation_ds} )

      GenErrByWeight     np.square(TrMeans - VlMeans) + (TrVars + VlVars)/2
      GenErr.append(GenErrByWeight.mean())
      Qerr.append(np.nanmean(Qmap))
  return Ws, np.array(GenErr), np.array(Qerr)

# метод для получения значений индексов ближайших элементов карты
# для каждого элемента входного множества
def classify(self, DataSet):
  return self._Session.run(
           self._BatchedBMU, feed_dict = {self._BatchedInput: np.array(DataSet)})

# метод для получения значений индексов ближайших элементов карты
# для каждого элемента входного множества в унитарном коде
def classify_onehot(self, DataSet):
  return self._Session.run(
           self._OneHotBBMU, feed_dict = {self._BatchedInput: np.array(DataSet)})
```

```
  # метод вычисления ошибки квантования для каждого элемента карты с учётом входных данных
  def calculate_qmap(self, DataSet):
    return self._Session.run(
             self._Qmap,       feed_dict = {self._BatchedInput: np.array(DataSet)})
```

МОДУЛЬ GRID.PY

```
import numpy as np
class Grid():
  def __init__(self, Rows, Cols, Stride=1, Type='hex'):
    seq   np.array(range(0, Rows*Cols))     # порядковый номер каждого элемента решётки

    # вычисление координат для каждого элемента решётки
    self.Centers = np.stack((Stride / 2 * (seq // Cols % 2) + seq % Cols,
                             Stride * (seq // Cols) * np.sin(np.pi/3)), axis=0 )

    # вычисление координат вершин шестиугольника для визуализации карты
    radius    = Stride / (2 * np.cos(np.pi/6))
    self.Vertices = np.stack((np.cos(np.pi*2*np.array(range(0,6))/6+np.pi/6),
                    np.sin(np.pi*2*np.array(range(0,6))/6+np.pi/6)),axis=0 )*radius

    # координаты вершин шестиугольника для каждого элемента решётки
    self.Cells = np.expand_dims(self.Vertices.T, axis=0) + np.expand_dims(self.Centers.T,
axis=1)
```

МОДУЛЬ main.py

```
from matplotlib.collections import PatchCollection, PolyCollection
from matplotlib.patches import Polygon
import grid
import matplotlib.pyplot as plt
import model
import numpy as np
import tensorflow as tf

# имена компонент входных данных – признаки кластеризуемых объектов
var_names = ['executions', 'sharable_mem', 'loaded_versions', 'version_count',
  'io_interconnect_bytes', 'io_offload_elig_bytes', 'io_offload_return_bytes',
  'physical_read_bytes', 'cell_uncompressed_bytes', 'physical_write_bytes',
  'parse_calls', 'px_servers_execs', 'end_of_fetch_count', 'fetches', 'buffer_gets',
  'invalidations', 'loads', disk_reads', 'optimized_physical_reads', '
   physical_read_requests', 'rows_processed', 'sorts', 'physical_write_requests',
  'direct_writes', 'cpu_time', 'elapsed_time', 'javexec_time', 'plsexec_time',
  'iowait', 'apwait', 'ccwait', 'clwait']

# загрузка данных
data   = np.loadtxt('data4.txt').tolist()
labels = np.loadtxt('labels4.txt', dtype=np.str).tolist()
       = np.matrix(data, dtype=np.float)
```

```
# вычисление основных моментов для входных данных
mx_d = np.nanmean(d, axis=0)
std_d = np.nanstd(d, axis=0)

# центрирование и стандартизация входных данных
d0     = (d - mx_d) / std_d

# вычисление ковариационной\корреляционной матрицы входных данных
cov_d0 = np.cov(d0.T)

# вычисление собственных векторов и чисел корреляционной матрицы
d0_eigval, d0_eigvec = np.linalg.eig(cov_d0)   # первичные компоненты

# сортировка компонент по убыванию объясняемой каждым компонентом дисперсии
index     = np.argsort(d0_eigval)[::-1]
d0_eigvec = d0_eigvec[:, index]
d0_eigval = d0_eigval[index]

# проекция данных на первичные компоненты
InputData      = np.matmul(d0, d0_eigvec[:, 0:11])

# разделение входных данных на обучающую и проверочную выборки для SOM
TrainData      = InputData[:InputData.shape[0]//2, :]
ValidationData = InputData[InputData.shape[0]//2:, :]
```

```
# размеры карты
Rows = 70
Cols = 70

# создание объекта карты
MySOM1 = model.SOM(Rows, Cols, InputData.shape[1], seed=12345)

# обучениие карты
weights, GenErr, Qerr = MySOM1.train(25, 0.05, train_ds      = np.array(TrainData),
                                     validation_ds = np.array(ValidationData) )

# вычисление границ для визуализации
borders = np.stack((MySOM1.grid.Cells.min((0,1)),MySOM1.grid.Cells.max((0,1))), axis=0).T

# получение результатов кластеризации входных данных
BMUs = MySOM1.classify(InputData)

# вычисление количества элементов входных данных в каждом кластере
HIST = MySOM1.classify_onehot(InputData).sum( axis=0 )

# вычисление U-map для самоорганизующейся карты
NNN = (np.abs(np.linalg.norm(
           np.expand_dims(MySOM1.grid.Centers, axis=2) -
           np.expand_dims(MySOM1.grid.Centers, axis=1),
           axis=0)) - 1 < 0.01).astype(int)
```

```
Umap = np.multiply(
         np.linalg.norm(np.expand_dims(weights.T, axis=2) -
           np.expand_dims(weights.T, axis=1),
           axis=0), NNN ).sum(axis=1) / NNN.sum(axis=1)

# получение карты ошибки квантования для каждого кластера
Qmap = MySOM1.calculate_qmap(InputData)

# перевод значений весов карты к компонентам входного сигнала
# из пространства первичных компонент для визуализации
restored_weights = np.matmul(weights, d0_eigvec.T[0:11])

# вычисление карты удалённостей центроидов кластеров от
# точки отсчёта координат (мат. ожидания входного множества)
magnitude = np.linalg.norm(weights, axis=1)

# отрисовка визуализации карты и сохранение визуализации в файлы
for i, j in zip( np.concatenate(
                   ( np.stack(
                     ( np.array(HIST).reshape(-1,),
                       Qmap,
                       np.nan_to_num(Qmap),
                       Umap,
                       magnitude ), axis=1),
                     weights,
```

```
                      restored_weights ),
                    axis=1).T,
                    ['HIST', 'QMAP_W_NAN', 'QMAP_WO_NAN', 'UMAP', 'MAGNITUDE'] +
                    ['COMP_%02d' % num for num in range(0, InputData.shape[1])] +
                    ['%02d_%s' % (index, value) for index, value in enumerate(var_names) ]
):

  fig, ax = plt.subplots()
  fig.set_size_inches( fig.get_size_inches() * 2 )
  plt.tight_layout()
  plt.axis('scaled')
  ax.set_title(j)
  ax.set_facecolor('grey')

  p = PolyCollection(MySOM1.grid.Cells, array=i, cmap='jet', antialiased=False,
snap=False, linewidths = 0)
  p.autoscale()
  ax.add_collection(p)
  plt.xlim(borders[0])
  plt.ylim(borders[1])
  plt.colorbar( mappable = ax.collections[0] )
  plt.savefig(j + '.png')
  plt.close()
  del p
```

Учебное издание

Кузнецова Алла Витальевна
Мохов Василий Александрович
Алгазали Салах Махди Мадлол

АЛГОРИТМЫ ОБУЧЕНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Учебно-методическое пособие
по изучению дисциплины
«Искусственный интеллект и мягкие вычисления»

Редактор Я.В. Максименко.

Подписано в печать 30.06.2020
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 4,18. Уч.-изд.л. 4,25. Тираж 50 экз. Заказ 46-0242.

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru