ОСНОВНОЙ ГОСУДАРСТВЕННЫЙ ЭКЗАМЕН

Д.М. Ушаков

ИНФОРМАТИКА

НОВЫЙ ПОЛНЫЙ

СПРАВОЧНИК

ДЛЯ ПОДГОТОВКИ

К ОГЭ

100 БАЛЛОВ

Д. М. Ушаков

ИНФОРМАТИКА

НОВЫЙ ПОЛНЫЙ

СПРАВОЧНИК

ДЛЯ ПОДГОТОВКИ

К ОГЭ

АСТ
Москва

УДК 373:002
ББК 32.81я721
У93

Ушаков, Денис Михайлович.

У93 Информатика : Новый полный справочник для подготовки к ОГЭ / Д.М. Ушаков. — Москва: Издательство АСТ, 2017. — 286, [2] с.: ил.

ISBN 978-5-17-096738-4

(Новый полный справочник для подготовки к ОГЭ)

ISBN 978-5-17-096734-6

(Самый популярный справочник для подготовки к ОГЭ)

В справочнике представлен материал курса информатики в объёме, проверяемом на государственной итоговой аттестации. Структура справочника соответствует современному кодификатору элементов содержания по предмету, на основе которого составлены контрольные измерительные материалы (КИМы) основного государственного экзамена (ОГЭ).

Книга будет незаменимым помощником при подготовке к экзамену в формате ОГЭ, при изучении и закреплении нового материала, повторении пройденных тем.

УДК 373:002
ББК 32.81я721

ISBN 978-5-17-096738-4
(Новый полный справочник для подготовки к ОГЭ)
ISBN 978-5-17-096734-6
(Самый популярный справочник для подготовки к ОГЭ)

Содержание

Предисловие

Данное учебное пособие предназначено для подготовки учащихся к сдаче основного государственного экзамена по информатике.

Справочник будет полезен как для учащихся при самостоятельной подготовке к экзамену, так и для преподавателей, желающих подготовить учащихся к сдаче экзамена.

Пособие написано на основе большого педагогического опыта подготовки автором учащихся к подобного рода экзаменам по информатике (ОГЭ и ЕГЭ).

Структура справочника соответствует современному кодификатору элементов содержания по предмету, на основе которого составлены контрольные измерительные материалы (КИМы) основного государственного экзамена (ОГЭ).

Материал сгруппирован по главам, в каждой из которых изучается определённая тема курса информатики и ИКТ, проверяемая на экзамене.

Главы включают в себя:

- теоретический материал, который необходим для понимания изучаемой темы,
- примеры задач с подробным разбором метода решения с обсуждением нескольких вариантов решения и рекомендациями по выбору нужного метода,

Автор надеется, что это пособие окажется полезным Вам, дорогой читатель.

Удачи на экзамене!

ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ

1 Представление информации

Количество информации. Единицы измерения количества информации. Формулы для вычисления количества информации

Конспект

Информация — базовое понятие, которому нельзя дать точного определения. Можно только определить его через один из синонимов. Например, «Информация — сведения об окружающем нас мире».

Для измерения количества информации придуманы специальные единицы **измерения информации.**

Основная единица измерения информации — **один бит.**

Один бит — это количество информации, уменьшающее неопределённость в два раза.

Что такое неопределённость? Проще всего это понимать как выбор из нескольких вариантов.

Например, Вася должен угадать, в какой из восьми одинаковых коробок, стоящих в ряд, лежит конфета.

Ему говорят, что конфета лежит в одной из левых 4-х коробок.

У Васи был выбор из 8-ми коробок, а остался выбор только из 4-х коробок. То есть, количество вариантов уменьшилось в 2 раза. Неопределённость уменьшилась в 2 раза. Значит, Васе сообщили 1 бит информации. Чтобы прийти к определённости (в какой конкретно коробке лежит конфета), нужно, чтобы остался только один вариант, т.е. оставшееся количество вариантов следует поделить на 2 (останется 2 вариан-

та, Вася получил ещё один бит информации), а затем оставшееся количество вариантов поделить ещё раз на 2 (останется 1 коробка, Вася получил ещё один бит информации).

В итоге, если изначально у нас был выбор из 8-ми коробок, и мы узнали, что конфета лежит в некоторой конкретной коробке, то мы должны были эту неопределённость (8 коробок) трижды поделить на 2, чтобы осталась только одна коробка, т.е. получить 3 бита информации.

Из этих соображений выведем основную формулу для вычисления количества информации.

Пусть у нас есть выбор из N одинаковых объектов. Необходимо выбрать один.

Будем делить количество объектов на 2 (уменьшать неопределённость в 2 раза) столько раз, сколько нужно для получения определённости (чтобы остался только один объект). Получаем формулу:

$$N / 2^i = 1.$$

Заметим, что ровно 1 можно получить только в том случае, если число N является степенью числа 2.

В противном случае, необходимо понимать, что бит — минимальная единица измерения информации и она не бывает не целой. То есть, если в процессе деления на 2 будут получаться не целые числа, нужно округлять до ближайшего целого (вверх).

Например, если бы изначально у нас было бы 5 объектов, нужно было бы делить так же 3 раза, как если бы их было бы 8 ($5 / 2 = 2{,}5$, округляем до 3; $3 / 2 = 1{,}5$, округляем до 2; $2 / 2 = 1$. Всего делили 3 раза).

Данное замечание можно понимать и по-другому — считать, что определённость, это когда возможных вариантов остаётся не больше одного, и нашу формулу правильнее было бы записать так:

$$N / 2^i \leqslant 1.$$

Перемножив обе части неравенства на 2^i, получим **формулу Хартли:**

$$2^i \geqslant N,$$

где N — количество равновероятных событий, i — количество информации (бит) в сообщении об одном таком событии, где этом i — минимальное целое число.

Если у нас есть выбор только из одного варианта, то количество информации в сообщении о таком событии рано нулю (сообщение о событии, которое происходит всегда, не несёт в себе информации).

Ещё одно определение *бита*: это количество информации в сообщении, которое может принимать только два возможных значения. Например, «да» или «нет».

Для обозначения возможных значений одного бита обычно используют цифры 0 и 1.

Если сообщение состоит из нескольких символов, и при этом эти символы равновероятны (нельзя заранее сказать, что какой-то символ сообщения может появиться чаще какого-нибудь другого), то количество информации в таком сообщении может быть вычислено по формуле:

$$I = k \cdot i,$$

где k — количество равновероятных символов в сообщении, i — количество информации (бит) в одном таком символе, I — количество информации (бит) во всем сообщении.

1 бит — достаточно маленькая единица измерения информации. Для большего удобства люди придумали более крупные единицы.

1 байт = 8 бит.
1 Кбайт = 1024 байт (= 2^{10} байт).
1 Мбайт = 1024 Кбайт (= 2^{20} байт).
1 Кбит = 1024 бит.
1 Мбит = 1024 Кбит.

Заметим, что приставки К и М, которые используются применительно к терминам бит и байт традиционно считаются как 1024 (2^{10}) и 10242 (2^{20}), а не как 1000 (10^3) и 1000000 (10^6), как это принято для других единиц измерения. Однако, их принято читать как «кило» и «мега». Так сложилось исторически.

При этом возникала путаница, потому что приставки «кило» и «мега» обозначают 10^3 и 10^6 соответственно. Чтобы её избежать, было принято решение для обозначения множителей 2^{10} и 2^{20} использовать термины «киби» и «миби»:

1 килобайт = 1000 байт, а 1 кибибайт = 1014 байт.

Однако, для приставок К и М такого правила формально введено не было. Поэтому в учебниках по информатике, на экзаменах по информатике и вообще в ИТ-среде принято считать, что К — это 2^{10}, а М — это 2^{20}.

В использованных определениях часто используется термин «равновероятный» (равновероятное событие, равновероятный символ). Это можно понимать так, что ни про одно событие (ни про один символ) нельзя заранее сказать, что частота его появления больше, чем у какого-нибудь другого события (символа).

 В случае, когда это свойство не выполняется, применяют другие способы вычисления количества информации. Они, как правило, связаны с понятием энтропии и знанием термина «логарифм» в математике. В рассматриваемом нами курсе информатики до 9-го класса включительно эти термины не считаются изученными и поэтому не используются. То есть, на ОГЭ вам не встретятся задачи, в которых события (символы) не будут равновероятными.

Наиболее известный пример такого (неравновероятного) события — встретить на улице динозавра. Так как это событие очень и очень маловероятно, то неверным будет считать,

что количество информации в сообщении «Я встретил сегодня на улице динозавра» равно одному биту! Действительно, можно предполагать, что ответ на вопрос «Встретил ли ты сегодня на улице динозавра?» имеет только два возможных варианта ответа — «да» и «нет» и поэтому несёт в себе только один бит информации. Но это верно только для равновероятных вариантов ответа. В данном же случае ответ «нет» встречается гораздо чаще ответа «да» и поэтому к нему не может применяться это определение бита.

Разбор типовых задач

Задача 1. В магазине продаётся 30 одинаковых упаковок шоколадных шариков. Известно, что в одной из этих упаковок находится приз. Вася покупает одну упаковку.

Какое количество информации содержится в сообщении о том, что приз находится именно в упаковке, купленной Васей?

Решение

Анализируем исходные данные. Так как все 30 упаковок одинаковые и приз находится только в одной из них, то сообщение, что приз находится именно в упаковке, купленной Васей — это одно из 30 равновероятных событий. То есть, мы применяем формулу Хартли: $2^i \geqslant N$.

Здесь N — количество равновероятных событий (30). Нужно подобрать наименьшее целое i такое, что $2^i \geqslant 30$. Если Вы не знаете наизусть степени числа 2 (что весьма полезно для сдачи ОГЭ по информатике), предлагаем подбирать эти степени последовательно, начиная с первой:

$2^1 = 2$. $2 \geqslant 30$? Нет. Берём следующую степень (домножаем на 2).

$2^2 = 2 \cdot 2 = 4$. $4 \geqslant 30$? Нет. Берём следующую степень (домножаем на 2).

$2^3 = 4 \cdot 2 = 8$. $8 \geqslant 30$? Нет. Берём следующую степень (домножаем на 2).

$2^4 = 8 \cdot 2 = 16$. $16 \geqslant 30$? Нет. Берём следующую степень (домножаем на 2).

$2^5 = 16 \cdot 2 = 32$. $32 \geqslant 30$? Да.

Получилось, что наименьшее i, при котором $2^i \geqslant 30$, — это число 5. Вспоминаем, что в формуле Хартли количество информации измеряется в битах.

Ответ: 5 бит.

Другой вариант нахождения нужного нам числа 5 — делить исходное число 30 на 2 до тех пор, пока не получится число, меньшее или равное 1:

$30 / 2 = 15$. $15 \leqslant 1$? Нет. Продолжаем.

$15 / 2 = 7{,}5$. Округляем до 8. $8 \leqslant 1$? Нет. Продолжаем.

$8 / 2 = 4$. $4 \leqslant 1$? Нет. Продолжаем.

$4 / 2 = 2$. $2 \leqslant 1$? Нет. Продолжаем.

$2 / 2 = 1$. $1 \leqslant 1$? Да.

Подсчитываем количество раз, которое мы делили на 2. Получаем 5.

Ответ: 5 бит.

Быстрый вариант решения

Если мы наизусть знаем первые степени числа 2 (рекомендуется знать первые 10 степеней числа 2), то мы можем быстро определить, что $16 \geqslant 30$? Нет, но $32 \geqslant 30$? Да. Значит, наименьшая степень числа 2, которая больше или равна исходному числу 30, это число 32, а это 2 в степени 5.

Значит, *ответ*: 5 бит.

Задача 2. В племени Мума-Тума в языке используется всего 64 различных слова. Один из членов племени говорит другому фразу, состоящую из 100 слов. Какое количество информации он сообщил?

Решение

В условии задачи требуется найти количество информации в сообщении. Будет использовать формулу $I = k \cdot i$. В ней нужно знать количество информации в одном символе и количество символов в сообщении.

Количество информации в одном символе нам не дано, но его можно постараться найти при помощи формулы Хартли: $2^i \geqslant N$.

То есть, остаётся определить, какое из чисел — 64 и 100 — является числом равновероятных событий N, а какое — количеством символов в сообщении k.

Анализируем условие и понимаем, что сообщение — это фраза, которую один член племени говорит другому. По условию фраза состоит из 100 слов. Но ведь в формуле $I = k \cdot i$ число k — это количество символов, а нам дано количество слов.

В данном случае нужно иметь в виду, что если племя общается между собой при помощи всего 64 слов, то они не разделяют слова на буквы, а используют каждое слово как отдельный, неделимый элемент общения. То есть, их слова — это и есть то, что мы при анализе сообщений называем символами. Значит, количество символов в сообщении $k = 100$.

Методом исключения получаем, что 64 — это количество равновероятных событий N. Действительно, если люди племени используют в разговоре всего 64 различных слова, то каждое из этих слов и есть одно из равновероятных событий, которые мы подсчитываем в формуле Хартли.

Подставляем подобранные величины в формулы.

Сначала по формуле Хартли найдём количество информации в одном слове (символе): $2^i \geqslant 64$.

Минимальное i, при котором это выполняется, равно 6 ($2^6 = 64$) и, следовательно, в одном слове племени содержится 6 бит информации.

Подставляем это в формулу $I = k \cdot i$. Получаем $I = 100 \cdot 6 = 600$ бит.

Ответ: 600 бит.

 Для измерения количества информации вовсе не обязательно, чтобы сообщение было хоть каким-нибудь образом осмысленным.

Задача 3. Какое количество байт в одном Кбите?

Решение

У нас имеется 1 Кбит. Нужно перевести это в байты.

Составим дробь. В числителе запишем исходное количество информации. В знаменателе — количество информации, которое необходимо получить:

1 Кбит / 1 байт.

Чтобы можно было делать действия с этими величинами, следует привести их к единой размерности. Проще всего сводить к самой маленькой величине — к битам.

Достаточно выучить, сколько бит в каждой из 5-ти единиц измерения информации, используемых в учебных задачах:

1 Кбит	= 1024 бит	$= 2^{10}$ бит
1 Мбит	= 1024 Кбит	$= 2^{20}$ бит
1 байт	= 8 бит	$= 2^3$ бит
1 Кбайт	= 1024 байт	$= 1024 \cdot 8$ бит $= 2^{13}$ бит
1 Мбайт	= 1024 Кбайт	$= 2^{23}$ бит

Подставим эти величины в нашу дробь и сократим степени двойки:

$2^{10} / 2^3 = 2^7 = 128$.

Ответ: 128 байт.

Системы счисления

Конспект

Общие сведения о системах счисления

Для записи чисел люди с давних времён используют специально придуманные для этого значки — цифры. Правила записи чисел и выполнения операция над ними называется **системой счисления**. Набор символов, использующийся для записи чисел, называется **алфавитом системы счисления**. В разные времена в разных странах использовались разные алфавиты и разные принципы составления чисел. Пример: римская система счисления (IV — четыре, VI — шесть).

В настоящее время мы пользуемся позиционной системой счисления. Для понимания этого термина введём ещё несколько понятий.

Назовём **весом цифры** тот вклад, который эта цифра добавляет в значение числа. Под значением числа будем представлять себе, например, количество палочек, которое число описывает. Так, уже упомянутое римское число VI — это шесть палочек — ||||||, число IV — это четыре палочки — ||||. Определим вес цифр в этих числах.

Цифра V в обоих случаях имеет вес пять — |||||, цифра I в числе VI имеет вес один — ещё одна палоч-

ка |, а в числе IV — имеет вес минус один — минус одна палочка. В римском числе II обе цифры I имеют одинаковый вес — один — одна палочка |.

Рассмотрим знакомые мы с детства числа той системы счисления, которой мы пользуемся в повседневной жизни. В числе 11 цифры 1 имеют разный вес. Правая цифра 1 имеет вес десять — десять палочек — ||||||||||. В то же время левая цифра 1 имеет вес один — одна палочка — |.

Система счисления, в которой вес цифры зависит от её положения в числе, называется **позиционной системой счисления.**

Обратите внимание, в чём ключевое отличие позиционной системы счисления от непозиционной — в позиционной системе счисления вес каждой цифры зависит от её положения в числе.

На примере чисел VI и IV римская система счисления имеет признаки позиционной (вес цифры I в этих числах зависит от положения цифры I относительно цифры V). Но наличие в системе счисления числа II, в котором вес обеих цифр одинаковый, сразу переводит римскую систему счисления в категорию непозиционных.

Количество цифр в алфавите позиционной системы счисления называется **основанием системы счисления.** Именно во столько раз в позиционной системе счисления вес каждого разряда больше предыдущего. Таким образом, уже знакомая нам система счисления называется **десятичной позиционной.** В ней десять цифр (от 0 до 9) и вес каждого разряда в десять раз больше предыдущего. Основание системы счисления принято писать справа внизу маленькими цифрами возле числа. Например, упомянутое нами число 11 правильнее записать как 11_{10}.

Для десятичной системы счисления основание обычно опускают.

Как уже было сказано ранее, в позиционной системе счисления используется столько различных цифр, чему равно основание этой системы счисления, начиная с нуля.

Например, в пятеричной системе счисления — пять цифр. Это 0, 1, 2, 3 и 4.

Обратите внимание!

Цифры 5 в пятеричной системе счисления нет. И бóльших цифр тоже нет.

Если в процессе осуществления действий в некоей системе счисления у вас неожиданно получилась цифра, больше или равная основанию системы счисления, значит, вы где-то ошиблись.

В двоичной системе счисления — всего две цифры. Это 0 и 1.

В восьмеричной системе счисления — 8 цифр. Это 0, 1, 2, 3, 4, 5, 6 и 7.

Самая большая цифра, которая используется в позиционной системе счисления, на 1 меньше, чем основание системы счисления. В десятичной системе счисления самая большая цифра — 9. В двоичной — 1, в пятеричной — 4.

Теперь отдельно рассмотрим 16-ричную систему счисления. В ней, соответственно, должно быть 16 цифр. Но нас в детства научили писать только 10 различных цифр — от 0 до 9! Как же быть? Для оставшихся цифр от десяти до пятнадцати принято использовать последовательные буквы латинского алфавита. То есть, цифра десять обозначается как A, цифра одиннадцать — B, а цифра пятнадцать — F.

Ещё раз хотелось бы обратить внимание на то, что буквам A, B, C, D, E, F соответствуют цифры от десяти до пятнадцати! Одна из частных ошибок — учащиеся считают, что A — это одиннадцать.

Перевод из любой системы счисления в десятичную систему счисления

Пусть имеется число 345_{10}. Посмотрим, сколько это с точки зрения системы счисления.

Все мы знаем, что 345 содержит 3 сотни, 4 десятка и 5 единиц. Попробуем представить это более формально. Пронумеруем разряды числа справа налево, начиная с нуля:

$$\begin{matrix} 210 \\ 345 \end{matrix}$$

Теперь вспомним, что сотня — это 10^2, а десяток — это 10^1, а единицу представим как 10^0. Получится, что наше число 345_{10} представимо как:

$$3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 = 3 \cdot 100 + 4 \cdot 10 + 5 = 345.$$

Получили **правило, по которому число из любой системы счисления можно перевести в десятичную систему:**

1) пронумеровать разряды числа справа налево, начиная с нуля,

2) выписать цифры числа, помноженные на основание системы счисления, возведённое в степень (номер разряда), написанную над цифрой,

3) сложить все полученные величины.

Переведём число 4312_5 в десятичную систему счисления.

Пронумеруем разряды справа налево, начиная с нуля:

$$\begin{matrix} 3210 \\ 4312_5. \end{matrix}$$

Теперь выписываем цифры числа, умноженные на основание системы счисления (5) в степени номера разряда: $4 \cdot 5^3 + 3 \cdot 5^2 + 1 \cdot 5^1 + 2 \cdot 5^0 = 4 \cdot 125 + 3 \cdot 25 + 1 \cdot 5 + 2 \cdot 1 = 500 + 75 + 5 + 2 = 582$.

Аналогично переведём число $2AB_{16}$. Нумеруем разряды:

$$2AB_{16} = 2 \cdot 16^2 + 10 \cdot 16^1 + 11 \cdot 16^0 = 512 + 160 + 11 = 683.$$

Точно так же можно перевести число из любой системы счисления, в том числе, из двоичной. Однако, при переводе из двоичной системы счисления процедуру возможно несколько упростить. Это связано с тем, что все цифры в двоичной системе счисления — нули и единицы. Умножать на нули не имеет смысла, потому что получится всё равно ноль. А умножать на единицу не имеет смысла, потому что получится то же, что и было. Поэтому **процедура перевода из двоичной системы счисления** сводится к следующему: нумеруем разряды справа налево, начиная с нуля; складываем двойки, возведённые в степени номеров разрядов, при которых стоят единицы.

Пример:

$1011010_2 = 1011010_2 = 2^6 + 2^4 + 2^3 + 2^1 = 64 + 16 + 8 + 2 = 90$.

Перевод из десятичной системы счисления в любую другую систему счисления

Для перевода чисел из десятичной системы счисления в любую другую используется другой алгоритм. Исходное десятичное число нужно делить нацело на основание той системы счисления, в которой мы хотим получить число, пока при делении не получится ноль, и выписать полученные остатки в обратном порядке.

Рассмотрим перевод числа 13 в двоичную систему счисления. Так как хотим получить двоичное число, будем делить уголком нацело на 2.

$$\begin{array}{r|l} 13 & 2 \\ \hline 12 & 6 \\ \hline 1 & \end{array}$$

После первого деления получаем частное 6 и в остатке 1.

Продолжаем делить частное на 2 с остатком, пока не получим частное, равное нулю:

$$\begin{array}{r|l} 13 & 2 \\ \hline 12 & 6 \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 6 & 2 \\ \hline 6 & 3 \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 2 \\ \hline 2 & 1 \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 0 & 2 \\ \hline 0 & 0 \\ \hline 1 & \end{array}$$

Обратите внимание, последнее действие — деление числа 1 на число 2. При этом получается вовсе не 0,5, как мы привыкаем к 9-му классу, а 0 целых и 1 в остатке. То есть, если делимое меньше делителя, то частное получается 0 и в остатке остаётся целиком делимое. Например, при делении 13 нацело на 16, получается 0 и в остатке 13.

Получившиеся в процессе четырёх делений остатки (1, 0, 1, 1) выписываем в обратном порядке:

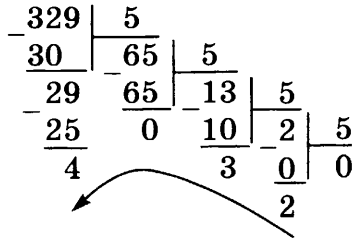
$$\begin{array}{r|l} 13 & 2 \\ \hline 12 & 6 \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 6 & 2 \\ \hline 6 & 3 \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 2 \\ \hline 2 & 1 \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 0 & 2 \\ \hline 0 & 0 \\ \hline 1 & \end{array}$$

Получаем двоичное число: 1101_2.

Аналогичным образом осуществляется перевод в любую систему счисления из десятичной системы счисления.

Например, переведём число 329 в пятеричную систему счисления.

Для этого будем делить уголком на 5 и собирать остатки в обратном порядке:



Получаем число 2304_5.

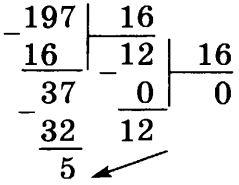
Как проверить, что мы правильно перевели в другую систему счисления?

Можно просто перевести полученное число снова в десятичную систему счисления:

Проверка:

$$2304_5 = \overset{3210}{2304_5} = 2 \cdot 5^3 + 3 \cdot 5^2 + 0 \cdot 5^1 + 4 \cdot 5^0 =$$
$$= 2 \cdot 125 + 3 \cdot 25 + 4 = 250 + 75 + 4 = 329.$$

Рассмотрим ещё один пример — перевод в 16-ричную систему счисления:



В результате деления получилось два остатка — 12 и 5.

Типичная ошибка — записать ответ как 125! Ведь остатков получилось два, значит, и в ответе должны быть записаны две цифры (а не три, как в числе 125). В этот момент следует вспомнить, что 12 нужно записать как 16-ричную цифру, т.е., как цифру С.

Получаем ответ: $C5_{16}$.

Другой способ перевода в двоичную систему счисления

Существует ещё один способ перевода в двоичную систему счисления. Он более быстр, чем перевод делением уголком, но и подвержен большему количеству потенциальных ошибок. Для его применения нужно знать наизусть все степени числа 2, не меньшие переводимого числа. Или иметь под рукой таблицу этих степеней.

Идея такая. Если при переводе числа из двоичной системы счисления в десятичную мы на втором шаге представляем наше число в виде суммы степеней числа 2 (при номерах разрядов которых стоят единицы), то и в обратную сторону можно сделать так же.

То есть:

1) представляем исходное число в виде суммы степеней числа 2;

2) выписываем двоичное число с единицами в тех позициях, которые имеют степени числа 2.

Рассмотрим эту операцию подробно на примере числа 53. Переведём его в двоичную систему счисления. Сначала выделим из числа 53 наибольшую возможную степень числа 2, которая в ней «помещается». Это $2^5 = 32$. Действительно, следующая степень числа 2 — $2^6 = 64$. Это больше 53. Вычитаем из исходного числа 53 выделенную степень числа 2: $53 = 32 + (53 - 32) = 2^5 + (53 - 32) = 2^5 + 21$.

Все эти действия нужно делать быстро и в уме. Если это вызывает затруднения, рекомендуем использовать более надёжный способ деления уголком.

Затем нужно выделить из оставшегося числа (теперь это 21) снова наибольшую степень числа 2. $53 = 2^5 + 21 = 2^5 + 16 + (21 - 16) = 2^5 + 2^4 + 5$. И так далее до тех пор, пока все число не будет разложено на

сумму степеней числа 2. В данном случае: $53 = 2^5 + 2^4 + 4 + (5 - 4) = 2^5 + 2^4 + 2^2 + 1 = 2^5 + 2^4 + 2^2 + 2^0$.

Когда все исходное число разложено на сумму степеней числа 2, нужно записать его в двоичном виде.

Мы предлагаем такой способ: начиная с самой большой степени числа 2, которая присутствует в числе, называть до нуля целые номера разрядов и записывать цифру 1, если такая степень числа 2 присутствует в сумме-разложении. Либо записывать цифру 0, если такая степень не присутствует.

В данном примере будем называть цифры 5, 4, 3, 2, 1, 0. Для номеров разрядов 5, 4, 2, 0 сумма содержит степени числа 2. Для оставшихся номеров разрядов (3 и 1) — не содержит.

Получим итоговое число 110101_2.

Это способ хорош для относительно небольшого размера чисел и в том случае, если подобные арифметические действия вы быстро и без ошибок делаете в уме.

Родственные системы счисления

Важно понимать, что системы счисления с основанием 2, 8 и 16 имеют особое, весьма важное, значение в информатике. Это связано с тем, что вся информация в компьютере хранится, обрабатывается и передаётся в двоичной системе счисления. Но запись чисел в двоичной системе счисления зачастую оказывается достаточно неудобной из-за того, что числа получаются весьма длинными. Чтобы сократить длину чисел и при этом сохранить запись в виде, из которого очень легко можно получить двоичную систему счисления, используется 8-ричная и 16-ричная системы счисления. О том, как быстро переводить числа из двоичной системы счисления в 8-ричную и 16-ричную и обратно, вы узнаете далее.

Возьмём некоторое двоичное число, например, 11111_2. В нём вес каждого разряда в два раза больше

веса предыдущего разряда. Рассмотрим, разряд номер 1. Его вес — $2^1 = 2$. Если рассмотреть разряд, на 2 позиции отстоящий от текущего (в данном случае, номер 3), его вес равен $2^3 = 8$. При этом он в 4 раза больше разряда номер 1 ($4 = 2 \cdot 2 = 2^2$). А если рассмотреть разряд, на 3 позиции отстоящий от разряда номер 1 (разряд номер 4, его вес равен $2^4 = 16$), его вес будет в 8 раз больше ($8 = 2 \cdot 2 \cdot 2 = 2^3$). Но ведь именно таким свойством — когда вес каждого разряда в 8 раз больше предыдущего — обладают числа, записанные в 8-ричной системе счисления. То есть, вес трёх подряд идущих двоичных разрядов как раз в 8 раз больше веса предыдущих трёх подряд идущих двоичных разрядов. Ровно так же, как вес соседних 8-ричных разрядов.

Из этих соображений получается **правило для перевода числа из двоичной в восьмеричную систему счисления:**

1) группируем разряды справа налево по три; если общее количество разрядов не кратно трём (самая левая группа оказывается неполной) — оставляем её такой (или дописываем слева нулями, это неважно);

2) каждую группу записываем как восьмеричную цифру;

3) полученные цифры записываем в том же порядке, что и группы, из которых они были получены.

Перевод чисел из двоичной в 16-ричную систему счисления происходит по такой же схеме. Несколько отличий:

1) так как число 16 — это четвёртая (а не третья, как число 8) степень числа 2, то и группировать двоичные разряды нужно по 4 (а не по 3, как при переводе в 8-ричную системы счисления);

2) важно понимать, что в процессе перевода должно получиться ровно столько цифр, сколько было получено групп разрядов.

Типичная ошибка — переведя 4 двоичных разряда в десятичную систему счисления, учащиеся забывают, что нужно вместо полученного десятичного числа, бóльшего 9, записать не две десятичные цифры, а одну — соответствующую 16-ричную цифру.

В остальном процедура перевода двоичного числа в 16-ричную систему счисления полностью аналогична переводу двоичного числа в 8-ричную систему счисления.

Таблица соответствия первых 16-ти десятичных, двоичных, 8-ричных и 16-ричных чисел:

Система счисления			
10-я	**2-я**	**8-ричная**	**16-ричная**
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Перевод из 16-ричной системы счисления в двоичную систему счисления осуществляется по таким же правилам, что и перевод из 8-ричной системы счисления в двоичную. Единственное исключение — нужно каждую 16-ричную цифру заменять на 4 двоичные цифры (а не на 3 двоичные цифры, как при переводе из 8-ричной системы счисления).

Разбор типовых задач

Задача 1. Переведите число 10101110_2 в восьмеричную систему счисления.

Решение

Для начала группируем разряды справа налево, по 3 цифры: $10\ 101\ 110_2$.

Заметим, самая левая группа оказалась неполной. Это нормально.

Теперь вместо каждой группы из трёх двоичных цифр (для левой группы — двух цифр) запишем соответствующую ей восьмеричную цифру.

Это можно сделать несколькими способами. Самый простой способ — знать наизусть все двоичные коды первых 16-ти чисел. Второй способ — воспользоваться таблицей, в которой приведено соответствие двоичных и восьмеричных чисел. Третий способ (если нет возможности воспользоваться первым и вторым способом) — перевести каждую группу двоичных цифр как отдельное двоичное число в десятичную систему счисления. Для цифр от 0 до 7 (которые могут в этом случае встретиться) десятичная и 8-ричная системы счисления совпадают.

Воспользуемся третьим способом. Переведём число 10_2. Нумеруем разряды: 10_2. Теперь складываем степени числа 2, при которых стоят цифры 1. В данном случае это $2^1 = 2$.

Переведём число 101_2. Нумеруем разряды: $^{210}101_2$. Складываем степени числа 2: $2^2 + 2^0 = 4 + 1 = 5$.

Переведём число 110_2. Нумеруем разряды: $^{210}110_2$. Складываем степени числа 2: $2^2 + 2^1 = 4 + 2 = 6$.

Выписываем полученные цифры в том же порядке. Получаем 256_8.

Ответ: 256_8.

Задача 2. Перевести число 1110101110_2 в шестнадцатеричную систему счисления.

Решение

Для начала группируем разряды справа налево по 4 цифры: 11 1010 1110_2. Заметим, самая левая группа оказалась неполной. Это нормально.

Теперь вместо каждой группы из трёх двоичных цифр (для левой группы — двух цифр) запишем соответствующую ей 16-ричную цифру.

Переведём число 11_2. Нумеруем разряды: $^{10}11_2$. Теперь складываем степени числа 2, при которых стоят цифры 1. В данном случае это $2^1 + 2^0 = 3$.

Переведём число 1010_2. Нумеруем разряды: $^{3210}1010_2$. Складываем степени числа 2: $2^3 + 2^1 = 8 + 2 = 10$. Записываем это в 16-ричном виде: $10_{10} = A_{16}$.

Переведём число 1110_2. Нумеруем разряды: $^{3210}1110_2$. Складываем степени числа 2: $2^3 + 2^2 + 2^1 = 8 + 4 + 2 = 14$. Записываем это в 16-ричном виде: $14_{10} = E_{16}$.

Выписываем полученные цифры в том же порядке. Получаем $3AE_{16}$.

Ответ: $3AE_{16}$.

Рассмотрим теперь обратную задачу — перевод из восьмеричной или шестнадцатеричной системы счисления в двоичную систему счисления. В задаче 1 мы рассматривали перевод из двоичной системы счисле-

ния в восьмеричную систему счисления. При этом мы заменяли каждые три двоичных разряда на один восьмеричный. Соответственно, теперь мы будем делать наоборот — заменять каждый восьмеричный разряд на группу из трёх двоичных разрядов.

Задача 3. Перевести число 625_8 в двоичную систему счисления.

Решение

Каждую цифру исходного числа будем по отдельности переводить в двоичную систему счисления.

Это возможно сделать несколькими способами. Самый простой — знать, какому восьмеричному числу соответствует какое двоичное число. Другой способ — посмотреть в таблицу соответствия восьмеричных и двоичных чисел (см. стр. 25). Если же первые два способа вам недоступны (не знаете наизусть соответствия восьмеричных и двоичных чисел и не имеете под рукой таблицы соответствия), всегда остаётся третий способ (универсальный, но более медленный). Можно просто перевести каждое двоичное число уголком (или разложением на сумму степеней числа 2) в двоичную систему счисления.

В нашем случае, сначала переведём число 6. Разложим его на сумму степеней числа 2:

$6 = 4 + 2 = 2^2 + 2^1 = 110_2$.

Теперь так же переведём цифру 2: $2 = 2^1 = 10_2$.

Переведём оставшуюся цифру 5:

$5 = 4 + 1 = 2^2 + 2^0 = 101_2$.

Осталось выписать все полученные двоичные цифры в ряд, друг за другом.

Внимание! Не допустите типичную ошибку. В частности, если делать «не задумываясь», получится ответ 11010101_2. Этот ответ — неверный. Причина ошибки в том, что мы должны были вместо каждой

восьмеричной цифры выписать 3 двоичные цифры! А мы вместо цифры 2 выписали только 2 цифры! Очень важно, чтобы каждая восьмеричная цифра была заменена именно на три двоичные цифры. Если цифр не хватает, обязательно дополните группу двоичных цифр слева нужным количеством нулей до трёх.

В данном случае, нужно было в середине записать не 10, а 010. Получаем ответ: 110010101_2.

Ответ: 110010101_2.

Задача 4. Перевести число $73E_{16}$ в двоичную систему счисления.

Решение

Каждую цифру исходного числа будем по отдельности переводить в двоичную систему счисления. Как и в случае с переводом из 8-ричной в двоичную систему счисления, это можно сделать несколькими способами.

Самый простой — знать, какому 16-ричному числу соответствует какое двоичное число. Другой способ — посмотреть в таблицу соответствия 16-ричных и двоичных чисел (см. стр. 25). Если же первые два способа едоступны (вы не знаете наизусть соответствия 16-ричных и двоичных чисел и не имеете под рукой таблицы соответствия), всегда остаётся третий способ (универсальный, но более медленный). Можно просто перевести каждое двоичное число уголком (или разложением на сумму степеней числа 2) в двоичную систему счисления.

В нашем случае, сначала переведём число 7.

Разложим его на сумму степеней числа 2:

$6 = 4 + 2 + 1 = 2^2 + 2^1 + 2^0 = 111_2$.

Теперь переведём число 3.

Разложим его на сумму степеней числа 2:

$3 = 2 + 1 = 2^1 + 2^0 = 11_2$.

Переведём число E. Вспомним, что E — это число 14. Разложим его на сумму степеней числа 2:

$14 = 8 + 6 = 2^3 + 4 + 2 = 2^3 + 2^2 + 2^1 = 1110_2$.

Заметим, что просто записать теперь ответ, составив рядом полученные двоичные числа — неправильно. Каждую из 16-ричных чисел нужно записать как 4 двоичные цифры, а для наших цифр 7 и 3 это получилось не так. Соответственно, нужно добавить к обоим числам слева столько нулей, чтобы получилось 4 цифры.

Запишем полученные 4-значные двоичные цифры друг за другом в том порядке, в котором стоят соответствующие им 16-ричные цифры исходного числа. Получаем 0111 0011 1110. Первый незначащий ноль можно, не записывать.

Получаем 11100111110.

Ответ: 11100111110.

Так как цифра 7 исходного числа — первая цифра числа, то для него не обязательно выполнять правило, что каждая 16-ричная цифра исходного числа должна быть заменена ровно на 4 двоичные цифры.

Однако, именно эта ошибка — забыть добавить ведущие нули для не первых цифр числа — является основной ошибкой в подобных задачах. Поэтому мы рекомендуем помнить, что каждая 16-ричная цифра должна быть заманена на ровно 4 двоичные цифры, а уже потом не трудно убрать ведущие незначащие нули в ответе.

2 Передача информации

Скорость передачи информации. Единицы измерения скорости передачи информации. Формулы для вычисления скорости передачи информации

Конспект

Чтобы передать информацию от одного объекта к другому, необходимо наличие **канала связи**. Этим каналом может быть, например, воздух (именно так люди обмениваются информацией при разговоре), электрические провода (например, телефонный разговор), просто любое пространство (если это беспроводная передача данных посредством радиоволны, например). По каналу связи информация передаётся в виде **сигналов**. Ими могут быть, например, звуки (при обычном разговоре), электрические импульсы (по телефонной линии), электромагнитная волна (при сотовой связи, например) и многое другое. Соответственно, возникает вопрос о скорости передачи информации.

Скорость передачи информации — это количество информации, переданное за единицу времени.

Принято измерять количество информации в битах в секунду. То есть, чтобы посчитать скорость передачи информации, нужно поделить количество переданной информации (в битах) на время, которое эта информация передавалась (в секундах). Получаем формулу:

$$v = I / t,$$

где v — скорость передачи информации, I — количество информации в сообщении (в битах), t — время передачи сообщения (в секундах).

Разбор типовых задач

Задача 1. Вася передаёт Пете сообщение, состоящее из 200 символов в течение 10 секунд. При этом каждый символ несёт в себе 5 бит. Определите скорость передачи информации.

Решение

Для вычисления скорости передачи информации нам нужно знать количество передаваемой информации и время передачи информации. Время передачи мы знаем — 10 секунд. Посчитаем объём передаваемой информации. В условии дано количество передаваемых символов и количество информации в каждом символе. Воспользуемся формулой $I = k \cdot i$. Количество символов равно 200. Это k. Количество информации в одном символе равно 5 бит. Это i. Подставим это в формулу: $I = 200 \cdot 5 = 1000$ бит. Теперь посчитаем скорость передачи: $v = 1000 / 10 = 100$ бит в секунду.

Ответ: 100 бит/с.

Задача 2. Маша передаёт Даше сообщение, состоящее из 60 вывешенных в ряд флажков в течение полминуты. Известно, что в сообщении используется только 16 различных флажков. Какова скорость передачи информации?

Решение

Для вычисления скорости передачи информации необходимо знать количество информации в сообщении и время передачи сообщения ($v = I / t$). Время передачи нам дано. Это полминуты, т.е. 30 секунд. Будем искать количество информации в сообщении.

Для этого нужно количество символов в сообщении и количество информации в одном символе ($I = k \cdot i$). Попытаемся понять, что в данном случае является символами сообщения. Это то, что последовательно передаётся в качестве сообщения. В данном случае это флажки. То есть один флажок — один символ. Про них нам известно количество 60 и количество 16. Что из этого k — количество символов в сообщении? Это то, сколько символов передаётся в ряд. Так как в ряд висит 60 флажков, то $k = 60$. Ещё нужно знать количество информации в одном символе. Это можно найти по формуле Хартли ($2^i \geqslant N$). Так как N — количество равновероятных событий, то это и есть 16 — количество различных флажков.

Подставляем: $2^i \geqslant 16 \rightarrow i = 4$ (количество информации в одном символе-флажке).

Подставляем в формулу $I = k \cdot i = 60 \cdot 4 = 240$ бит (количество информации в сообщении).

Подставляем в формулу $v = I / t = 240 / 30 = 8$.

Ответ: 8 бит/с.

Задача 3. Файл размером 2000 Кбайт передаётся через некоторое соединение в течение 30 секунд. Определите размер файла (в Кбайт), который можно передать через это соединение за 12 секунд.

В ответе укажите одно число — размер файла в Кбайт. Единицы измерения писать не нужно.

Решение

Формула для вычисления скорости передачи информации: $v = I / t$. Определим скорость передачи через указанное соединение: v = 2000 Кбайт / 30 секунд = 200/3 Кбайт/сек.

В задаче спрашивается, какого размера файл можно передать через это соединение (то есть, с той же скоростью) за 12 секунд.

Из формулы $v = I / t$ выразим I:

$$I = v \cdot t.$$

Подставим в формулу имеющиеся у нас величины: $I = 200/3 \cdot 12 = 800$ Кбайт.

Ответ: 800 Кбайт.

Другое решение задачи

Из формулы скорости передачи информации $v = I / t$ видно, что величины v и I прямо пропорциональны друг другу (увеличение в n раз одной величины приведёт к увеличению в такое же количество раз другой величины).

Составим пропорцию:

2000 Кбайт	—	30 секунд
X Кбайт	—	12 секунд

Отсюда выразим X:

$X = 2000$ Кбайт $\cdot$ 12 секунд / 30 секунд = 800 Кбайт.

Ответ: 800 Кбайт.

Кодирование и декодирование при передаче информации

Конспект

Равномерный и неравномерный коды

При передаче информации часто возникает необходимость в особой форме кодирования информации. Как правило, это делается для ускорения передачи информации или для исправления ошибок, возникающих в процессе передачи.

Основной способ кодирования при передаче информации — такой же, как и при представлении информации в компьютерном виде. То есть, каждому передаваемому символу ставится в соответствие определённый код. Этот код известен передатчику (его принято называть **источник**) и **приёмнику**. Источник преобразует передаваемые символы в соответствующие коды (кодирует информацию), коды передаются по каналу связи в виде сигналов, после чего приёмник преобразует полученные коды в символы (декодирует информацию). Последовательность сигналов, которая соответствует передаваемому символу, называется **кодовым словом**. То есть, код, используемый при передаче для кодирования и декодирования информации, состоит из кодовых слов.

Коды, которые присваиваются символам для передачи, могут быть равномерными или неравномерными. **Равномерным** называется код, у которого длина каждого кодового слова одинаковая. **Неравномерным** называется код, у которого хотя бы у двух кодовых слов длины различаются.

В обоих случаях процесс кодирования передаваемого сообщения одинаков — каждому передаваемому символу по очереди из кодовой таблицы выбирается его кодовое слово, и это кодовое слово передаётся по каналу связи.

Пример равномерного кода:

Символ	А	Б	В	Г	Д
Код символа	000	001	010	011	100

Пример неравномерного кода:

Символ	А	Б	В	Г	Д
Код символа	000	001	00	1	10

При декодировании неравномерного кода основная проблема — как разбить входящую последователь-

ность на кодовые слова. Ведь не известно, на последовательности какой длины нужно разбить последовательность (длины кодовых слов у разных символов разные).

Префиксный код и дерево декодирования

Самый простой способ, при котором задача определения длины кодовых слов решается просто — использовать для передачи **префиксный код**.

Для префиксного кода должно выполняться следующее **правило: никакое кодовое слово не должно быть началом никакого другого кодового слова**. Если это правило выполняется, то для такого кода возможно построить очень удобную для декодирования конструкцию — **дерево декодирования**. Как правило, количество передаваемых через канал связи сигналов равно двум, поэтому дерево декодирования является **двоичным деревом**. Если передают большее количество сигналов (например, K), то и дерево используется K-ичное. На рёбрах этого дерева записываются передаваемые сигналы, а листьями являются декодируемые символы.

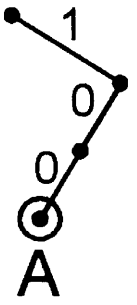
Пример префиксного кода:

Символ	А	Б	В	Г	Д
Код символа	100	11	01	101	00

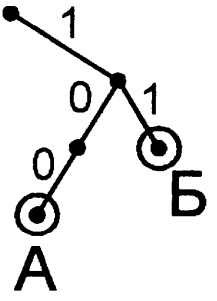
Приведём **алгоритм построения дерева декодирования**. Рассмотрим его на примере префиксного кода, только что приведённого в таблице в качестве примера.

1. Нарисуем корень дерева — точку, от которой будет «расти» наше дерево и от которой впоследствии будет начинаться декодирование каждого следующего символа.	●

2. Возьмём кодовое слово первого символа (в нашем примере — это кодовое слово 100 символа А). Будем рисовать от корня ветки дерева, добавляя для каждого сигнала кодового слова (в нашем случае это 0 и 1) ветку налево, если этот сигнал 0, и ветку направо, если это сигнал 1 (можно и наоборот, но желательно делать это по общему правилу, чтобы не запутаться). В данном случае сначала рисуем ветку вправо (первый символ — кодового слова 100). Затем два раза рисуем ветку влево (второй и третий сигнал кодового слова 100). На каждой рисуемой ветке пишем соответствующий ей сигнал — 0 для веток налево и 1 для веток направо. Лист дерева мы специально обвели в кружок, чтобы в дальнейшем не перепутать его с узлом дерева (и не нарисовать случайно из этой вершины новую ветку). Рядом с листом пишем символ, который соответствует этому листу дерева (символ А).

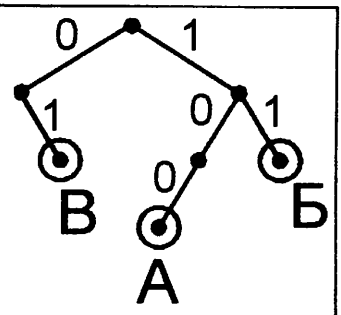


3. Проделаем такую же процедуру для следующего символа (это символ Б, его кодовое слово — 11). Первая ветка (вправо, соответствующая сигналу 1) от корня дерева уже нарисована. Просто «движемся» вдоль ветки право. Вторую ветку рисуем от этой точки вправо (второй сигнал кодового слова 11). Не забываем подписать эту ветку (цифрой 1). Так же обводим получившийся лист в кружок и подписываем его кодируемой буквой Б.



4. Ту же операцию повторим для следующего символа — В. Его кодовое слово 01. Нарисуем от корня одну ветку налево (для первого сигнала (0) кодового слова 01). Затем от её конца добавим одну ветку направо (для второго сигнала (1) кодового слова 01).

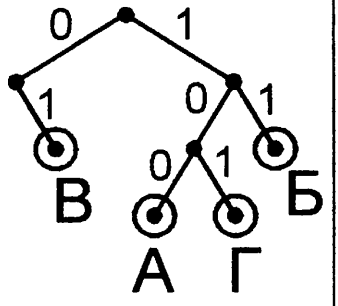
Не забываем подписывать ветки соответствующими цифрами и обвести в кружок получившийся лист, подписав его кодируемой буквой В.



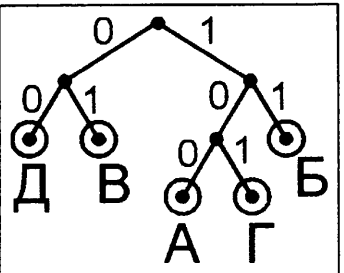
5. Для следующего символа — Г (кодовое слово 101) повторяем операцию. Для первых двух сигналов его кодового слова (1 и 0) от корня дерева уже нарисованы соответствующие ветки. Просто «движемся» вдоль них от корня.

Потом рисуем от узла, до которого мы «доехали» таким образом (по пути от корня дерева сначала по ветке 0, затем по ветке 1) ветку, соответствующую оставшемуся сигналу (остался сигнал 1, ветка вправо). Подписываем ветку. Обводим в кружок лист.

Подписываем его буквой Г.



6. Для оставшегося символа — Д, кодовое слово 00 — повторяем процедуру.
От корня дерева сначала «движемся» влево по ветке 0, потому что нужная нам ветка влево (для первого сигнала 0) уже нарисована. От этого узла добавляем ветку влево. Подписываем ветку. Обводим в кружок лист. Подписываем его буквой Д.



Дерево декодирования готово.

Алгоритм использования дерева декодирования при декодировании следующий:

1. Из канала связи (входного потока приходящих сигналов) приёмник извлекает получаемые сигналы по одному.

2. Получив первый сигнал, приёмник «движется» по дереву декодирования по ветке, подписанной полученным сигналом.

3-а. Если при этом приёмник «доезжает» до листа дерева, приёмник принимает решение, что получен символ, которым подписан этот лист. После этого приёмник считает, что нужно принимать кодовое слово нового символа. То есть, начинает извлекать новые сигналы по одному, получая кодовое слово следующего символа и путешествуя по дереву декодирования от корня.

3-б. Если приёмник «доезжает» до узла дерева (вершины, не являющейся листом), приёмник считает, что очередной символ ещё не получен, извлекает из входного потока новый сигнал и «движется» по дереву декодирования до очередной вершины дерева. После

этого снова анализирует, до какого вида вершины он «доехал» (переходит к шагу 3 алгоритма).

Декодирование префиксных кодов очень удобно тем, что это можно делать «на лету», получая из входного канала сигналы по одному и сразу же производя декодирование, не дожидаясь окончания всего сообщения.

Непрефиксные неравномерные коды

Ещё один способ использования кодов, однозначно декодирующихся — **постфиксные коды**. Это почти то же самое, что и префиксные коды, только для них должно выполняться **правило — никакое кодовое слово не является концом никакого другого кодового слова**.

Такие коды обрабатываются точно так же, как и префиксные коды, только они декодируются с конца. Это не очень удобно, так как для декодирования принятого сообщения приходится ждать, пока все сообщение будет принято, а потом уже начинать процесс его декодирования.

Использование при передаче неравномерных префиксных и постфиксных кодов является удобным, но не является обязательным. Коды, которые не обладают свойством превиксности или постфиксности, часто тоже можно однозначно декодировать. Для этого нужно лишь, чтобы не существовала последовательность сигналов, которая может быть двумя разными способами разбита на отдельные кодовые слова.

Задача построения таких кодов не очень проста, а при наличии такой технологии, как префиксные коды, их использование и вовсе не имеет смысла. Однако, эти коды встречаются и их тоже можно декодировать.

Бывают даже коды, которые не допускают однозначного декодирования, но при этом некоторые последовательности, закодированные при помощи подобных кодов, все же возможно декодировать.

Разбор типовых задач

Задача 1. Закодируем равномерным кодом последовательность символов АВГАБ.

Решение

Будем последовательно записывать коды символов в том порядке, в котором они даны в кодируемой последовательности: 000 010 011 000 001. Пробелы мы поставили только для понятности того, что мы делаем. В действительности в результате кодирования получается слитная последовательность. То есть, 000010011000001.

Ответ: 000010011000001.

Если используется равномерный код, процесс декодирования происходит так же просто — приёмник знает длины кодовых слов, он отсчитывает из канала связи количество сигналов, равное длине кодового слова, «заглядывает» в таблицу соответствия передаваемых символов и кодовых слов и извлекает оттуда соответствующий символ. Т.е., получив на вход последовательность 000010011000001, приёмник однозначно разбивает её на кодовые слова одинаковой длины (3) 000 010 011 000 001 и записывает вместо каждого кодового слова соответствующий этому кодовому слову символ: АВГАБ.

Задача 2. Закодируем неравномерным кодом последовательность символов АВГАБ.

Решение

Принцип кодирования точно такой же — последовательно для каждого кодируемого символа из таблицы извлекается кодовое слово и записывается в том же порядке.

В данном случае, получается: 000 001 000 001. Как и в первой задаче, мы поставили пробелы для лучшего понимания, как происходил процесс. В действительности получается сплошная последовательность: 000001000001.

Ответ: 000001000001.

Задача 3. Разведчик передал в штаб радиограмму

• – – • • • – • • • – – • • – • – –

В этой радиограмме содержится последовательность букв, в которой встречаются только буквы А, Д, Ж, Л, Т. Каждая буква закодирована с помощью азбуки Морзе. Разделителей между кодами букв нет. Запишите в ответе переданную последовательность букв.

Нужный фрагмент азбуки Морзе приведён ниже.

А	Д	Ж	Л	Т
• –	– • •	• – • •	–	• • • –

Решение

Сначала исследуем приведённый код на предмет префиксности. Проверим все кодовые слова маленькой длины, не являются ли они началом какого-нибудь другого кодового слова. Обнаружим, что кодовое слово – (буквы Л) является началом кодового слова – • • (буквы Д). Значит, этот код не префиксный.

Тогда исследуем приведённый код на предмет постфиксности. Проверим все кодовые слова маленькой длины, не являются ли они концами каких-нибудь

других кодовых слов. Обнаружим, что кодовое слово – (буквы Л) является концом кодового слова • • • – (буквы Т).

Значит, быстро декодировать последовательность не удастся. Придётся анализировать и рассматривать различные варианты декодирования.

Будем пытаться разбивать радиограмму на отдельные кодовые слова.

Радиограмма начинается на точку. Это значит, что первая буква — А, Ж или Т.

Далее идёт тире. На точку-тире начинается кодовое слово буквы Ж, а также это целиком кодовое слово буквы А. Однако, так как следующим (третьим) полученным символом является тире, мы принимаем решение, что первые два символа (• –) — это буква А, потому что в коде буквы Ж третьим символом должна быть точка, а других кодовых слов, начинающихся на • – –, в коде нет.

Теперь пытаемся разбить на кодовые слова последовательность – • • • – • • – – • • – • – –.

Пусть первый символ последовательности — это кодовое слово буквы Л (–). Тогда последующие символы (• • • –) — это кодовое слово буквы Т (нет другого кодового слова, начинающегося на • •). Следовательно, при попытке выделить в оставшейся последовательности • • – – • • – • – – следующее кодовое слова, мы приходим в тупик. В коде нет кодового слова, начинающегося на • • –. Саму последовательность • • – также нельзя разбить на кодовые слова, так как кодового слова, состоящего из одной точки, в коде нет. Кодовое слово, начинающееся на • •, это только кодовое слово буквы Т (• • • –). В нём после двух точек должна быть снова точка, а не тире, как в начале оставшейся последовательности • • – – • • – • – –.

Значит, наше предположение (что первым символом последовательности – • • • – • • – – • • – • – – является символ Л — неверно).

Из этого делаем вывод, что в начале этой последовательности стоит другое кодовое слово, начинающееся на тире — буква Д (– • •).

Пытаемся выделить следующее кодовое слово в оставшейся последовательности • – • • – – • • – • – –. Пусть это буква А (• –). Тогда у нас остаётся последовательность • • – – • • – • – –, которая, как мы ранее уже выясняли, не декодируется. Следовательно, первая буква — не А, а Ж (• – • •).

Оставшаяся последовательность: – – • • – • – –. В ней первая буква может быть только Л (нет ни одного кодового слова, начинающегося на – –).

Оставшаяся последовательность: – • • – • – –. Если предположить, что в ней первая буква — снова Л, то остальная последовательность не декодируется (мы уже проверили, что последовательность, начинающаяся с символов • • –, в этом коде быть не может). Значит, первая буква в этой оставшейся последовательности (– • • – • – –) — это буква Д (– • •).

Оставшаяся последовательность: – • – –. Первой буквой может быть только Л (нет кодового слова, начинающегося с – • –). Остаётся • – –, которая может быть разбита на кодовые слова только как • –, – (А, Л).

То есть, общее разбиение исходной последовательности на кодовые слова получилось таким:

• –, – • •, • – • •, –, – • •, –, • –, –

Заменим полученные кодовые слова на соответствующие им символы и получим АДЖЛДЛАЛ.

Ответ: АДЖЛДЛАЛ.

Заметим, что приведённый код в общем случае не допускает однозначного декодирования, потому что в нем последовательность – • • • – может быть разбита на кодовые слова как – • , • • – (то есть декодирована как Д, А), а может быть разбита на кодовые слова как – , • • • – (то есть, декодирована как Л, Т). В данном случае приведённая радиограмма однозначно декодируется, потому что попытка трактовать последовательность – • • • – как два законченных кодовых слова приводит к невозможности декодировать оставшееся сообщение. Приходится считать, что последовательность – • • • –, — это кодовое слово – • •, а оставшиеся символы • –, являются началом другого кодового слова. Поэтому двойственности декодирования не возникает.

3 Обработка информации

Логические значения, операции, выражения

Конспект

Введение

В окружающем нас мире мы постоянно сталкиваемся с утверждениями. Например, «Маша — молодец», «Дважды два — пять», «Идёт дождь», «Завтра будет хорошая погода». Если про утверждение можно сказать, истинно оно или ложно, такое утверждение называется **логическим утверждением.**

Например, среди приведённых утверждений:

«Маша — молодец»	Не является логическим утверждением, потому что подобная оценка (молодец) субъективна.
«Дважды два — пять»	Является логическим утверждением. Про него можно сказать, что это — ложь
«Идёт дождь»	Является логическим утверждением, если в этот момент видно, какая в настоящий момент погода (идёт дождь или нет). Соответственно, если дождь идёт, то это истинное логическое утверждение, а если не идёт — ложное логическое утверждение.

«Завтра будет хорошая погода»	Не является логическим утверждением, потому что проверить его в настоящий момент нельзя.

Результаты логических утверждений (истину и ложь) обычно обозначают:

Истина	Да	True	1
Ложь	Нет	False	0

В информатике, как правило, используются обозначения 1 и 0. Это удобно, коротко, наглядно. Позволяет пользоваться математическим аппаратом двоичной системы счисления. Однозначно определяет, как эти значения хранятся в памяти компьютера.

Логические операции

При обработке логических утверждений часто возникает необходимость объединять несколько логических выражений или преобразовывать их. Для этого придуманы специальные логические операции. Мы будем рассматривать только три основные логические операции.

- **Логическое И**

Также называется **логическое умножение** или **конъюнкция.**

Выполняется над двумя логическими операндами.

Операндом называется то, над чем выполняется операция. Например, всем известная операция сложения выполняется над двумя слагаемыми. Слагаемые — это операнды. А операция умножения производится над сомножителями. Сомножители — тоже операнды.

Обозначения: **И, AND, &, ·**

То есть, если операция выполняется над двумя логическими операндами A и B, то возможные обозначения:

A И B
A AND B
A & B
A · B

Результат операции истина, если значения обоих логических операнда — истина.

Как вы понимаете, диапазон значений, которые могут принимать логические операнды, ограничен (только истина и ложь, 0 и 1). Поэтому имеется возможность просто перечислить, какое значение будет у логической операции при всех возможных значениях операндов.

Все эти варианты логических операндов и результат для каждого случая логической операции принято записывать в таблицу, которая называется таблица истинности. Так как у операции логическое **И** всего два операнда и каждый принимает два возможных логических значения, то таблица истинности для этой операции содержит всего $2 \cdot 2 = 4$ строки. Для всех комбинаций возможных значений операндов.

Приведём таблицу истинности для операции логического **И**:

A	B	A И B
0	0	0
0	1	0
1	0	0
1	1	1

Как можно видеть, результат — истина получается только в том случае, если оба операнда — истина.

• **Логическое ИЛИ**

Также называется **логическое сложение** или **дизъюнкция.**

Выполняется над двумя логическими операндами.

Обозначения: **ИЛИ, OR,** $\vee$, +

То есть, если операция выполняется над двумя логическими операндами A и B, то возможные обозначения:

A ИЛИ B

A OR B

A $\vee$ B

A + B

Результат операции истина, если у хотя бы одного логического операнда значение — истина.

Приведём таблицу истинности для операции логическое **ИЛИ:**

A	B	**A ИЛИ** B
0	0	0
0	1	1
1	0	1
1	1	1

Как можно видеть, результат — ложь получается только в том случае, если оба операнда — ложь.

• **Логическое НЕ**

Также называется **отрицание** или **инверсия.**

Выполняется над одним логическим операндом.

Обозначения: **НЕ, NOT,** $\neg$, $\overline{\ }$ (последнее обозначение — линия сверху над операндом)

То есть, если операция выполняется над логическим операндом A, то возможные обозначения:

НЕ A $\neg$ A

NOT A $\overline{A}$

Результат операции меняется на противоположный (из истины — ложь, из лжи — истина).

Приведём таблицу истинности для операции логическое НЕ:

А	НЕ А
0	1
1	0

Так как у операции логическое НЕ всего один операнд, то таблица истинности этой операции содержит всего две строки (для каждого возможного значения операнда).

Как можно видеть, результат — ложь получается только в том случае, если оба операнда — ложь.

Приоритеты логических операций

Часто бывает, что нужно выполнить сразу несколько логических операций и объединить их результаты другими логическими операциями. То есть, выполнить целое **логическое выражение**. Например, А **ИЛИ** В **И** С.

Для того, чтобы понять, какая из этих операций выполняется первой, а какая — второй, придуманы **приоритеты логических операций**. То есть, порядок, в котором разные операции выполняются в сложном логическом выражении. Вы, конечно, понимаете, что это такое! Ведь ещё с начальной школы все мы знаем, например, что умножение выполняется раньше сложения. Так выражение $2 + 3 \cdot 4$ равно выражению $2 + (3 \cdot 4)$. То есть, сначала выполнится умножение, а потом — сложение. Если же в выражении $2 + 3 \cdot 4$ нужно выполнять сначала сложение, а только потом умножение, нужно добавить скобки: $(2 + 3) \cdot 4$.

Так же и здесь. Логическое умножение (логическое **И**) выполняется раньше логического сложения

(логического **ИЛИ**). Полностью приоритеты изученных нами логических операций следующие:

0. Выражение в скобках
1. Логическое **НЕ**
2. Логическое **И**
3. Логическое **ИЛИ**

Операции одинакового приоритета выполняются слева направо.

Вычисление логического выражения по таблице истинности

В связи с тем, что количество возможных значений аргументов в логических выражениях ограничено, появляется возможность посчитать значение логического выражения для всех возможных значений аргументов. Мы уже использовали эту технологию для вычисления всех возможных значений логических операций. Точно так же можно сделать для более сложных логических выражений. Это называется **таблица истинности**. Количество строк в таблице истинности равно числу 2 в степени количества аргументов (переменных). То есть, если в выражении, например, три различных переменных, то количество строк в таблице истинности: $2 \cdot 2 \cdot 2 = 2^3 = 8$.

Разбор типовых задач

Задача 1. Расставьте, в каком порядке будут выполняться действия в логическом выражении

A **ИЛИ НЕ** B **И НЕ** C

Решение

Выражение не содержит скобок, поэтому порядок действий будет строго по правилу **НЕ-И-ИЛИ**.

Сначала выполнятся оба отрицания (логическое **НЕ**). Потом выполнится логическое умножение (логи-

ческое **И**). Последним выполнится логическое сложение (логическое **ИЛИ**). То есть, порядок выполнения действий будет таким:

	4		1			3		2	
А	**ИЛИ**	**НЕ**		**В**		**И**		**НЕ**	**С**

Задача 2. Расставьте, в каком порядке будут выполняться действия в логическом выражении:

(А И НЕ В) ИЛИ С И НЕ (В ИЛИ НЕ С)

Решение

Выражение содержит скобки, поэтому сначала нужно выполнить все операции в скобках, и только потом — операции вне скобок. Так как в выражении две пары скобок, не вложенных друг в друга, выполним по порядку: в левой скобке, потом — в правой.

В скобках по отдельности и потом — вне скобок, порядок действий будет по тому же правилу **НЕ-И-ИЛИ**. Сначала выполняется отрицание (логическое **НЕ**) в левой скобке (**НЕ** В). Потом — логическое умножение (логическое **И**) там же, в левой скобке (А **И** (**НЕ** В)). После этого левая скобка будет вычислена и можно переходить к вычислению следующей скобки.

В правой скобке выполняется отрицание (логическое **НЕ** — **НЕ** С). Потом выполнится логическое сложение (логическое **ИЛИ** — В **ИЛИ** (**НЕ** С)).

Теперь вычисляются операции вне скобок. Как и ранее, сначала выполним отрицание (логическое **НЕ**) над правой скобкой. Затем — логическое умножение (логическое **И**) для С **И** правой скобки. Последним выполним логическое сложение (логическое **ИЛИ**) для левой скобки и результатом только что полученного логического **И**. То есть, порядок выполнения действий будет таким:

	2	1		7		6	5		4	3
(А	**И**	**НЕ**	**В)**	**ИЛИ**	**С**	**И**	**НЕ**	**(В**	**ИЛИ**	**НЕ**	**С)**

Задача 3. Построить таблицу истинности выражения:

А И НЕ (В ИЛИ НЕ С)

Решение

Подсчитываем количество различных переменных в выражении. В выражении используются переменные: А, В, С. То есть, три переменных. Значит, в таблице истинности будет $2^3 = 8$ строк.

Вспоминаем приоритеты и расставляем порядок действий в выражении:

4 3 2 1

А И НЕ (В ИЛИ НЕ С)

Запишем таблицу истинности. Первые три столбца в ней будут соответствовать переменным А, В, и С. В этих столбцах перебирём все возможные значения этих переменных. Далее нарисуем ещё 4 столбца по количеству логических операций в выражении:

			1	2	3	4
А	**В**	**С**	**НЕ С**	**В ИЛИ (1)**	**НЕ (2)**	**А И (3)**
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

В столбцах А, В, С перебрали все возможные значения логических переменных А, В, С. Чтобы не сбиться и не забыть ни одной комбинации мы использовали следующую технологию. В первом столбце сначала записали половину нулей, затем вторую половину —

единиц. То есть, в данном случае, получилось 4 нуля и 4 единицы. Во втором столбцы мы сначала написали в два раза меньше нулей (в нашем случае, получилось два нуля), затем столько же единиц, затем снова столько же нулей, затем снова столько же единиц. В третьем столбце мы снова писали вдвое меньшее количество то нулей, то единиц. Это был последний столбец переменных, поэтому получилось по одному нулю и одной единице, чередуясь.

Данная технология хороша несколькими моментами. Во-первых, она позволяет быстро и почти не задумываясь построить таблицу истинности. Во-вторых, полученные таким образом последовательности нулей и единиц в строках являются последовательными двоичными числами. То есть, таким образом можно быстро породить также и необходимое количество кодов первых двоичных чисел (обычно 8 или 16).

Над первой строкой таблицы мы поставили номера логических операций, чтобы в таблице было удобнее вычислять их значения. Так, например, в столбце со второй операцией в действительности вычисляется значение выражения «B **ИЛИ НЕ** C». Однако, при такой записи этого выражения не очень наглядно, какая именно операция сейчас вычисляется и над какими аргументами она производится. Вместо этого ввели обозначения (пронумеровали) операции данного выражения. В результате вместо «B **ИЛИ НЕ** C» мы используем обозначение «B **ИЛИ** (1)». Где под обозначением (1) имеется ввиду результат операции номер 1 нашей таблицы. Это наглядно демонстрирует, какая именно операция и над какими аргументами (операция **ИЛИ** над столбцом B и столбцом, подписанным номером 1) сейчас выполняется.

Теперь будем последовательно, столбец за столбцом, заполнять результаты логических операций. В нашем примере первая операция — **НЕ** C. Её вычисление примитивно — для каждой цифры столбца C пи-

шем в столбец под номером 1 число 1, если в столбце С стоит 0, и число 0, если в столбце С стоит 1.

Теперь вычисляем вторую операцию — В **ИЛИ** (1). Смотрим последовательно в каждую строчку столбцов В и (1) и, если обнаруживаем в них хотя бы одну единицу, пишем в результирующий столбец (2) единицу. А если оба нуля — пишем 0.

			1	2	3	4
А	В	С	**НЕ** С	В **ИЛИ** (1)	**НЕ** (2)	А **И** (3)
0	0	0	1	1		
0	0	1	0	0		
0	1	0	1	1		
0	1	1	0	1		
1	0	0	1	1		
1	0	1	0	0		
1	1	0	1	1		
1	1	1	0	1		

Столбец 3 — **НЕ** (2) — вычисляем аналогично столбцу 1. То есть, пишем 1, если в столбце 2 стоит 0. И пишем 0, если в столбце 2 стоит 1.

			1	2	3	4
А	В	С	**НЕ** С	В **ИЛИ** (1)	**НЕ** (2)	А **И** (3)
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Столбец 4 вычисляем, просто перемножая числа, стоящие в столбцах А и (3).

Задача 4. Для какого из приведённых чисел ложно высказывание:

НЕ (число > 50) **ИЛИ** (число чётное)?

1) 123 2) 56 3) 9 4) 8

Решение

Самый простой («в лоб») способ решения этой задачи — подставить в логическое выражение каждое из предлагаемых чисел и посмотреть, в каком случае результат выражения будет ложным.

Удобнее всего (чтобы не ошибиться в уме и ничего не потерять) составить для этого аналог таблицы истинности. Количество строк будет равно количеству вариантов чисел. Для каждого числа мы будем вычислять логические утверждения. В данном случае это «число > 50» и «число чётное».

После подставим результаты этих логических утверждений в логические операции и вычислим их результат. Проанализируем порядок, в котором будут вычисляться операции данного выражения. Сначала вычислим выражения в скобках, потом — **НЕ**, последним — **ИЛИ**.

Получаем следующий порядок:

3 1 4 2

НЕ (число > 50) **ИЛИ** (число чётное)

Составляем «таблицу истинности»:

	1	2	3	4
число	(число > 50)	(число чётное)	**НЕ** (1)	(3) **ИЛИ** (2)
123				
56				
9				
8				

Заполняем последовательно, столбец за столбцом и строчка за строчкой, эту таблицу.

В частности:

число 123 > 50 ? Да. Пишем в соответствующую ячейку 1.

число 56 > 50 ? Да. Пишем ниже тоже 1.

число 9 > 50 ? Нет. Пишем ниже число 0.

И так далее.

Получаем:

	1	2	3	4
число	(число > 50)	(число чётное)	**НЕ** (1)	(3) **ИЛИ** (2)
123	1	0	0	0
56	1	1	0	1
9	0	0	1	1
8	0	1	1	1

Результат 0 (ложь) получился только в одной строке таблицы — в строке 1 для числа 123. Указываем его (номер 1) в качестве ответа.

Заметим, что если вы решили вычислять исходное выражение для каждого числа по отдельности, и получили для какого-нибудь из чисел сразу нужный результат (в данном случае, ложь), не спешите сразу считать это правильным ответом. Мы рекомендуем на всякий случай, вычислить значение выражения для всех возможных вариантов ответа, чтобы проверить себя. Потому что, если обнаружится нужный ответ ещё для одного варианта, это будет повод задуматься над тем, что вы, возможно, неверно вычисляете выражение.

Приведённый способ (построение таблицы) хотя и является достаточно медленным, но позволяет свести к минимуму риск ошибки.

Другой способ решения

Можно решать эту задачу аналитически. По условию, выражение **НЕ** (число > 50) **ИЛИ** (число чётное) должно быть ложным. Последняя вычисляемая операция этого выражения — **ИЛИ**. Операция **ИЛИ** — ложна тогда и только тогда, когда оба её операнда — ложны. То есть, должно быть ложно выражение **НЕ** (число > 50) и должно быть ложно утверждение (число чётное).

Чтобы выражение **НЕ** (число > 50) было ложно, нужно, чтобы утверждение (число > 50) было истинно.

А чтобы утверждение (число чётное) было ложно, нужно, чтобы число было нечётное.

Получаем, что все исходное выражение ложно в том случае, если число больше 50 и нечётно.

Ищем среди вариантов ответа нечётное число, большее 50 и находим его (123) под вариантом номер 1.

Ответ: 1.

Алгоритмы. Простые исполнители

Конспект

В данной главе мы собираемся обсудить принципы анализа и построения алгоритмов. **Алгоритм** — это последовательность команд, направленных на выполнение какого-либо действия. За время учебы в школе вы часто пользовались тем или другим алгоритмом. Обычно именно в форме алгоритма даются в математике правила, например, умножения или деления чисел. При обсуждении правил перевода чисел из одной

системы счисления в другую мы тоже используем алгоритм.

Для того, чтобы исполнение алгоритма приводило к известному, желаемому и предсказуемому результату, желательно наличие следующих условий.

Первое условие — наличие **исполнителя**, того, кто будет исполнять алгоритм. Им может быть робот, компьютер, человек, машина (например, стиральная или посудомойная).

Второе — **система команд исполнителя** (сокращенно — **СКИ**). Это набор тех команд/инструкций, которые исполнитель может выполнить. Крайне желательно, чтобы эти команды/инструкции были чётко определены и понятны для исполнителя. Особенно это важно, когда исполнитель — человек. Как правило, человек способен вольно трактовать данные ему команды/инструкции. Поэтому именно человек зачастую выполняет алгоритм с непредсказуемым результатом. В некоторых заданиях экзамена требуется именно исполнить предложенный алгоритм. Важно научиться выполнять алгоритм формально, не придумывая к нему при этом ничего лишнего.

Третье — должна быть известна **среда исполнения**, пространство, в котором работает/живёт/выполняет алгоритм исполнитель. Для робота, перемещающегося по клеткам лабиринта — это лабиринт. Для выдуманного исполнителя Кузнечика, прыгающего по числам на числовой прямой — это числовая прямая.

Четвёртое условие — **система отказов**. Это описание тех ситуаций, когда исполнитель не может выполнить указанную команду в том месте (в тот момент), в котором исполнитель сейчас находится. Например, если исполнитель должен уметь выполнять операцию деления, то отказ возникнет в тот момент, когда его попросят выполнить деление на ноль.

Рассмотрим несколько типичных учебных исполнителей.

1. У исполнителя **Удвоитель** две команды, которым присвоены номера:

1. Умножь на 2,

2. Прибавь 3.

Первая из них удваивает число на экране, вторая — увеличивает его на 3.

2. Исполнитель **Чертёжник** перемещается на координатной плоскости, оставляя след в виде линии. Чертёжник может выполнять команду **Сместиться на (*a*, *b*)** (где a, b — целые числа), перемещающую Чертёжника из точки с координатами (x, y) в точку с координатами $(x + a, y + b)$. Если числа a, b положительные, значение соответствующей координаты увеличивается; если отрицательные — уменьшается.

Например, Чертёжник находится в точке с координатами (9, 5). Команда **Сместиться на (1, −2)** переместит Чертёжника в точку (10, 3).

3. Исполнитель **Робот** живёт в прямоугольном лабиринте на клетчатой плоскости. Он умеет передвигаться, выполняя следующие команды:

вверх **вниз** **влево** **вправо**

При выполнении этих команд Робот перемещается на одну клетку соответственно:

вверх ↑, вниз ↓, влево ←, вправо →.

Если Робот начнёт движение в сторону стены, то он разрушится и программа прервётся.

Четыре команды проверяют истинность условия отсутствия стены у той клетки, где находится РОБОТ:

сверху свободно **снизу свободно** **слева свободно** **справа свободно**

Определим у приведённых исполнителей, чем являются для них перечисленные выше условия.

Исполнитель	Система команд исполнителя	Среда исполнения	Система отказов
Удвоитель	Умножь на 2 Прибавь 3	Шкала целых чисел. Текущее число отображается на экране. Можно сказать, числовая прямая	Отсутствует (нет каких-нибудь сведений или предположений, в каком случае Удвоителю может не удастся выполнить какую-либо из его команд)
Чертёжник	Сместиться на (a, b)	Бесконечное поле — координатная плоскость.	Отсутствует (нет сведений о том, что плоскость чем-то ограничена или что у Чертёжника что-то может сломаться)
Робот	вверх вниз влево вправо сверху свободно снизу свободно слева свободно справа свободно	Прямоугольный лабиринт на клетчатой плоскости	Движение в стену (если по границе клетки, в которой находится Робот, стоит стена и Робот получает команду движения в сторону этой стены, Робот разрушится)

Не всегда среда исполнения для исполнителя такая идеальная и бесконечная, как у Удвоителя или

Чертёжника. Часто среда исполнения ограничена размерами или препятствиями и у исполнителя могут возникнуть разные ситуации, которые иногда даже не может предположить разработчик. В этом случае, как Вы, вероятно, понимаете, алгоритм приводит к непредсказуемому результату.

В учебных задачах система команд исполнителя обычно проста и непредвиденных ситуаций не возникает. Выполнение ряда заданий при этом даже не требует особенных навыков, а выполняется в пределах здравого смысла и понимания основного принципа формального алгоритма. Ваша задача не придумывать, как мог бы выполниться алгоритм, потому что вам так хочется, а выполнять ровно то, что написано.

Разбор типовых задач

Задача 1. Некоторый алгоритм из одной цепочки символов получает новую цепочку следующим образом. Сначала вычисляется длина исходной цепочки символов; если она чётна, то в начало цепочки символов добавляется символ Г, а если нечётна, то дублируется средний символ цепочки.

В полученной цепочке символов каждая буква заменяется буквой, следующей за ней в русском алфавите (А — на Б, Б — на В и т. д., а Я — на А).

Получившаяся таким образом цепочка является результатом работы алгоритма.

Например, если исходной была цепочка **УРА**, то результатом работы алгоритма будет цепочка **ФССБ**, а если исходной была цепочка **ПУСК**, то результатом работы алгоритма будет цепочка **ДРФТЛ**.

Дана цепочка символов **РЕКА**. Какая цепочка символов получится, если к данной цепочке применить описанный алгоритм дважды (т.е. применить алго-

ритм к данной цепочке, а затем к результату вновь применить алгоритм)?

Русский алфавит:
А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я

Решение

Выполним формально, по действиям, предложенный алгоритм. Посчитаем в исходной цепочке символов **РЕКА** количество символов. Получаем 4. Это чётное число. Согласно алгоритму, нужно добавить в начало цепочки символ Г. Получаем цепочку символов **ГРЕКА**. Заменяем в цепочке каждый символ на следующий по алфавиту:

Г → Д
Р → С
Е → Ё
К → Л
А → Б

Получаем цепочку символов **ДСЁЛБ**.

Для полученной цепочки **ДСЁЛБ** ещё раз применим алгоритм.

Посчитаем количество символов в цепочке. Получаем 5. Это нечётное число.

Согласно алгоритму, дублируем средний символ цепочки. Это символ Ё. Получаем цепочку **ДСЁЁЛБ**. Заменяем в цепочке каждый символ на следующий по алфавиту:

Д→ Е
С → Т
Ё → Ж
Ё → Ж
Л → М
Б → В

Получаем цепочку символов **ЕТЖЖМВ**.

Ответ: **ЕТЖЖМВ**.

Задача 2. Автомат получает на вход трёхзначное десятичное число. По полученному числу строится новое десятичное число по следующим правилам.

1. Вычисляются два числа — сумма старшего и среднего разрядов, а также сумма среднего и младшего разрядов заданного числа.

2. Полученные два числа записываются друг за другом в порядке невозрастания (без разделителей).

Пример. Исходное число: 277.

Поразрядные суммы: 9, 14.

Результат: 149.

Определите, сколько из приведённых ниже чисел могут получиться в результате работы автомата.

1616 169 163 1916 1619 316 916 116

В ответе запишите только количество чисел.

Решение

Постараемся понять, что является средой исполнения для данного автомата. По условию автомат получает трёхзначное десятичное число и обрабатывает его цифры-разряды. Из математики известно, что трёхзначное десятичное число — это целое число от 100 до 999. Цифры-разряды этого числа принимают значения от 0 до 9 (за исключением первой цифры, которая не равно нулю, потому что иначе число не будет трёхзначным). Эти цифры в парах складываются. Сумма двух цифр может быть от 0 до 18. Ноль получается при сложении двух нулей, 18 — при сложении двух девяток. Больше 18 сумма получиться не может. То есть, полученные суммы — одно- или двухзначные числа.

Результирующее число алгоритма получается записыванием полученных сумм друг за другом. Из-за этого действия теряется информация о том, что является первой суммой, а что — второй, и где граница между ними.

Хорошо бы ещё чётко понимать, что означает приведённый здесь термин «невозрастающая последова-

тельность», а именно, что второе число не возрастает над первым, т. е. первое число больше или равно второму.

Кроме приведённых условий следует не забыть проверить ещё одно. По условию при получении двух сумм цифр-разрядов в обоих суммах используется одна и та же — средняя — цифра. Это значит, что нужно не забыть проанализировать полученные два числа-суммы на то, могут ли они быть получены по приведённому алгоритму, т.е. совпадет ли у них одна из цифр-слагаемых. Так, например, число 152 разбивается на части: 15 и 2. Каждое из этих чисел лежит в диапазоне 0..18 и первое больше или равно второму. Однако, оно не будет получено нашим автоматом, потому что наименьшая цифра, которая может участвовать как одно из двух слагаемых для получения числа 15, — это 6 (15 = 6 + 9). А наибольшая цифра, которая может участвовать как одно из двух слагаемых для получения числа 2 — это 2 (2 = 2 + 0). Поэтому приведённое число 152 не является результатом работы автомата.

Теперь анализируем данные по условию числа. Пытаемся разбить их на возможные пары сумм.

1616. Может быть разбито на два числа как 1 616, либо 16 16, либо 161 6. Первый и третий варианты не подходят, так как трёхзначной суммы получиться не должно. Вариант 16 16 не противоречит условию — число 16 лежит в нужном диапазоне (от 0 до 18) и последовательность 16, 16 является невозрастающей ($16 \geqslant 16$).

Попробуем получить пример трёхзначного исходного числа: 888. Обе суммы равны 16. Подходит.

169. Может быть разбито на два числа как 1 69, либо 16 9. Первый вариант не подходит, потому что 69 выходит за допустимый диапазон (от 0 до 18). Второй вариант подходит под диапазон и подходит под неубывание ($16 \geqslant 9$). Пример исходного числа: 790 (суммы $7 + 9 = 16$ и $9 + 0 = 9$). Подходит.

Для удобства составим таблицу анализа оставшихся чисел.

Число	Варианты разбиения на два числа	Попадают ли оба числа в диапазон 0..18	
163	1 63 16 3	0⩽1⩽18–Да, 0⩽63⩽18–**Нет** 0⩽16⩽18–Да, 0⩽3⩽18–Да	
1916	1 916 19 16 191 6	0⩽916⩽18–**Нет** 0⩽19⩽18–**Нет** 0⩽191⩽18–**Нет**	
1619	1 619 16 19 161 9	0⩽619⩽18–**Нет** 0⩽16⩽18–Да, 0⩽19⩽18–**Нет** 0⩽161⩽18–**Нет**	
316	3 16 31 6	0⩽3⩽18–Да, 0⩽16⩽18–Да 0⩽31⩽18–**Нет**	
916	9 16 91 6	0⩽9⩽18–Да, 0⩽16⩽18–Да 0⩽91⩽18–**Нет**	
116	1 16 11 6	0⩽1⩽18–Да, 0⩽16⩽18–Да 0⩽11⩽18–Да, 0⩽6⩽18–Да	

Среди всех предложенных чисел подошли только первые два (1616 и 169) и последнее (116), всего 3.

Ответ: 3.

Условие того, возможно ли разбить полученные два числа-суммы на три исходные цифры трёхзначного числа, можно в данном случае формализовать. Действительно, чтобы одна и та же цифра могла быть включена в обе суммы, эти суммы должны между собой отличаться не более чем на 9 (максимально возможную цифру).

	Выполняется ли условие неубывания	Подобрать пример исходного числа	Подходит ли хотя бы один вариант разбиения
	16⩾3–Да	Невозможно	Нет
			Нет
			Нет
	3⩾16–Нет		Нет
	9⩾16–Нет		Нет
	1⩾16–Нет 11⩾6–Да	560 (5 + 6 = 11, 6 + 0 = 6)	Да

Задача 3. У исполнителя **Делитель** две команды, которым присвоены номера:

1. раздели на 2

2. вычти 1

Первая из них уменьшает число на экране в 2 раза, вторая уменьшает его на 1.

Исполнитель работает только с натуральными числами.

Составьте алгоритм получения **из числа 65 числа 4**, содержащий не более 5 команд. В ответе запишите только номера команд.

(*Например*, 12112 — это алгоритм:
раздели на 2
вычти 1

раздели на 2
раздели на 2
вычти 1,
который преобразует число 42 в число 4).

Если таких алгоритмов более одного, то запишите любой из них.

Решение

Будем исходить из того, что в условии нас просят найти самый быстрый алгоритм. Хотя это и не указано в условии явно, фраза «алгоритм, ..., содержащий не более 5 команд» наводит именно на такие размышления. Из этих соображений анализируем возможные команды Делителя и понимаем, что команда номер 1 (раздели на 2) гораздо быстрее уменьшает текущее число, чем вторая команда (вычти 1). Поэтому будем пытаться использовать команду «раздели на 2» в каждом возможном случае. Нетрудно понять, что это произойдёт только в том случае, если текущее число — чётное. При делении нечётного числа на 2 получится нецелое число, а по условию Делитель работает только с натуральными числами. Попытаемся получить требуемое число 4 из числа 65 по принципу: при любой возможности, делить текущее число на 2 (т. е. если текущее число — чётное). Если это невозможно — вычитать 1.

Попробуем обработать по этому принципу исходное число 65.

№ шага алгоритма	Текущее число	Делится ли оно на 2 (чётное ли)?	Выполнение действия	№ команды Делителя
1	65	Нет	65 вычесть 1 = 64	2
2	64	Да	64 разделить на 2 = 32	1

№ шага алгоритма	Текущее число	Делится ли оно на 2 (чётное ли)?	Выполнение действия	№ команды Делителя
3	32	Да	32 разделить на 2 = 16	1
4	16	Да	16 разделить на 2 = 8	1
5	8	Да	8 разделить на 2 = 4	1

Составляем друг за другом последовательность номеров команд. Ответ: 21111.

Задача 4. У исполнителя **Удвоитель** две команды, которым присвоены номера:

1. умножь на 2

2. прибавь 1

Первая из них увеличивает число на экране в 2 раза, вторая увеличивает его на 1.

Исполнитель работает только с натуральными числами.

Составьте алгоритм получения **из числа 4 числа 42**, содержащий не более 5 команд. В ответе запишите только номера команд.

(*Например*, 12112 — это алгоритм:
умножь на 2
прибавь 1
умножь на 2
умножь на 2
прибавь 1,
который преобразует число 2 в число 21).

Если таких алгоритмов более одного, то запишите любой из них.

Решение

Как и в предыдущем примере, будем исходить из того, что в условии нас просят найти самый быстрый алгоритм.

Однако, в данном случае задача обратная — следует выбирать между операцией умножения и операцией сложения. Обе эти операции могут быть произведены над любым натуральным числом. Поэтому нам не удастся, решая задачу подобным же способом, принимать решение о том, какую из команд применять — умножение на 2 или прибавление 1 на основании какого-то свойства числа (чётности?).

Поэтому будем решать задачу в обратную сторону — искать, как можно получить из числа 42 число 4, выполняя обратные команды. Ведь, если число нечётное, то оно не может быть получено командой умножения на 2, следовательно, оно было получено другой командой (прибавь 1).

№ шага алгоритма	Текущее число	Могло ли оно быть получено умножением на 2 (чётное ли оно)?	Выполнение действия	№ команды Делителя
5	42	Да	42 / 2 = 21	1 (21 · 2 = 42)
4	21	Нет	21 – 1 = 20	2 (20 + 1 = 21)
3	20	Да	20 / 2 = 10	1 (10 · 2 = 20)
2	10	Да	10 / 2 = 5	1 (5 = 2 = 10)
1	5	Нет	5 – 1 = 4	2 (4 + 1 = 5)

В последнем столбце таблицы получили обратную последовательность необходимых действий.

Записав номера команд в порядке снизу вверх, получим ответ: 21121.

Задача 5. У исполнителя **Удвоитель** две команды, которым присвоены номера:

1. прибавь 1

2. умножь на 2

Первая из них увеличивает число на экране на 1, вторая увеличивает его в 2 раза.

Исполнитель работает только с натуральными числами.

Составьте алгоритм получения **из числа 5 числа 52**, содержащий не более 5 команд. В ответе запишите только номера команд.

(*Например*, 21221 — это алгоритм:
умножь на 2
прибавь 1
умножь на 2
умножь на 2
прибавь 1,
который преобразует число 2 в число 21).

Если таких алгоритмов более одного, то запишите любой из них.

Решение

Решим задачу тем же способом, что и в предыдущем примере.

№ шага алгоритма	Текущее число	Могло ли оно быть получено умножением на 2 (чётное ли оно)?	Выполнение действия	№ команды Делителя
5	52	Да	52 / 2 = 26	2 (26 · 2 = 52)
4	26	Да	26 / 2 = 13	2 (13 · 2 = 26)
3	13	Нет	13 − 1 = 12	1 (12 + 1 = 13)
2	12	Да	12 / 2 = 6	2 (6 · 2 = 12)
1	6	Да	6 / 2 = 3	?

Заметим, что предлагаемый алгоритм решения задачи на данном примере даёт сбой на последнем шаге. Если «бездумно» выполнить действие деления для текущего числа 6, то результатом будет число 3, которое никак не может получиться в требуемом алгоритме получения из числа 5 числа 52. Значит, на последнем шаге следует обратить внимание, что от нужного числа 5 нас отделяет только одно действие «вычесть 1» и вместо деления на 2 (обратного команде «2. умножь на 2») выполнить вычитание единицы (обратного команде «прибавь 1»).

Записываем получившиеся номера команд снизу вверх, поставив самой нижней командой 1: 12122.

Предлагаемый метод решения задачи, когда мы однозначно, за исключением последнего шага, принимаем решение, что при чётном числе было выполнено чётное действие (в информатике такой алгоритм называется жадным алгоритмом), не всегда возможно применять. Но попробовать решить задачу мы вам рекомендуем именно начиная с него. Если ответ не получается, попытайтесь «свернуть в другую сторону» — для чётного текущего числа вместо действия «умножь/подели на 2» выполните другое действие. Возможно, это даст нужный результат.

Задача 6. У исполнителя **Удвоитель** две команды, которым присвоены номера:

1. прибавь 3

2. умножь на 2

Первая из них увеличивает число на экране на 3, вторая увеличивает его в 2 раза.

Исполнитель работает только с натуральными числами.

Составьте алгоритм получения **из числа 14 числа 52**, содержащий не более 5 команд. В ответе запишите только номера команд.

(*Например*, 21221 — это алгоритм:

умножь на 2

прибавь 3
умножь на 2
умножь на 2
прибавь 3,
который преобразует число 2 в число 31).

Если таких алгоритмов более одного, то запишите любой из них.

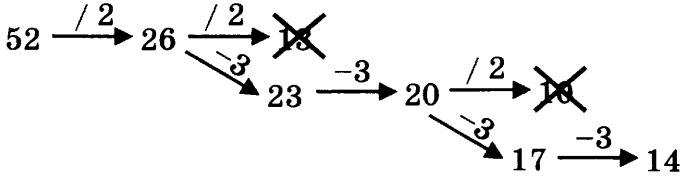
Решение

Будем решать эту задачу тем же способом, который предлагается в решении предыдущего примера.

№ шага алгоритма	Текущее число	Могло ли оно быть получено умножением на 2 (чётное ли оно)?	Выполнение действия	№ команды Делителя
5	52	Да	52 / 2 = 26	2 (26 · 2 = 52)
4	26	Да	26 / 2 = 13	?
Мы получили число 13, которое меньше стартового числа 14, которое мы хотели бы получить. Очевидно, этим способом ничего не получится. Вернёмся на шаг назад и выберем для него другую команду				
4	26	Да, но выбираем другое	26 – 3 = 23	1 (23 + 3 = 26)
3	23	Нет	23 – 3 = 20	1 (20 + 3 = 23)
2	20	Да	20 / 2 = 10	?
Снова неудача. Снова «откатываемся назад» и выбираем другую команду.				
2	20	Да, но выбираем другое	20 – 3 = 17	1 (17 + 3 = 20)
1	17	Нет	17 – 3 = 14	1 (14 + 3 = 17)

Записываем номера команд снизу вверх: 11112.

Подобный способ решения задачи называется **«перебор с возвратом»**. Можно его рассматривать в виде дерева решения, которое строим, последовательно изменяя текущее число и при каждой развилке (чётном текущем числе) двигаясь всегда в одну сторону (выбирая умножение/деление). Если на каком-нибудь шаге оказывается, что мы пришли по этому дереву в тупик (получили невозможное число), откатываемся назад к точке последней развилки и двигаемся далее по ней. Если и она приводит в тупик, возвращаемся к предыдущей развилке. В данном случае дерево решения выглядит следующим образом:



Можно было бы применять такой же метод и напрямую, «в лоб». То есть, не разворачивать задачу, рассматривая вместо сложения и умножения команды вычитания и деления, а сразу пытаться из исходного числа получить конечное. Такой способ решение будет сложнее и дольше, потому что в нём предположение о развилке придётся делать для каждого получаемого числа. Это серьезно увеличивает возможное дерево рассматриваемых решений.

Задачи, предлагаемые на экзамене, обычно решаются сразу, без единого возврата.

Задача 7. Исполнитель **Чертёжник** перемещается на координатной плоскости, оставляя след в виде линии. Чертёжник может выполнять команду **Сместиться на (*a*, *b*)** (где a, b — целые числа), перемещающую Чертёжника из точки с координатами (x, y) в точку с координатами $(x + a, y + b)$. Если числа a, b положи-

тельные, значение соответствующей координаты увеличивается; если отрицательные — уменьшается.

Например, если Чертёжник находится в точке с координатами (9, 5), то команда Сместиться на (1, −2) переместит Чертёжника в точку (10, 3).

Запись

Повтори *k* раз
Команда1 Команда2 Команда3
конец

означает, что последовательность команд **Команда1 Команда2 Команда3** повторится *k* раз.

Чертёжнику был дан для исполнения следующий алгоритм:

Повтори 3 раз
Сместиться на (−2, −3) Сместиться на (3, 2)
Сместиться на (−4, 0)
конец

На какую одну команду можно заменить этот алгоритм, чтобы Чертёжник оказался в той же точке, что и после выполнения алгоритма?

1. Сместиться на (−9,−3)
2. Сместиться на (−3,9)
3. Сместиться на (−3,−1)
4. Сместиться на (9,3)

Решение

Исследуем, на сколько сместится Чертёжник в результате выполнения приведённого алгоритма. Так как нас интересует только смещение Чертёжника относительно начальной позиции и не интересует, в каких именно координатах окажется Чертёжник, для простоты и удобства будем считать, что начальное положение Чертёжника — в точке (0, 0).

Первый метод решения. Простой, но долгий. Выполним друг за другом по порядку все команды алгоритма Чертёжника. В нём три команды повторяются 3 раза. Всего 9 команд. Не слишком долго.

Команда	Вычисление новых координат	Координаты Чертёжника после выполнения команды
Начальное положение	—	(0, 0)
Сместиться на (−2, −3)	0 − 2 = − 2, 0 − 3 = −3	(−2, −3)
Сместиться на (3, 2)	−2 + 3 = 1, −3 + 2 = −1	(1, −1)
Сместиться на (−4, 0)	1 − 4 = −3, −1 + 0 = −1	(−3, −1)
Сместиться на (−2, −3)	−3 − 2 = −5, −1 − 3 = −4	(−5, −4)
Сместиться на (3, 2)	−5 + 3 = −2, −4 + 2 = −2	(−2, −2)
Сместиться на (−4, 0)	−2 − 4 = −6, −2 + 0 = −2	(−6, −2)
Сместиться на (−2, −3)	−6 − 2 = −8, −2 − 3 = −5	(−8, −5)
Сместиться на (3, 2)	−8 + 3 = −5, −5 + 2 = −3	(−5, −3)
Сместиться на (−4, 0)	−5 − 4 = −9, −3 + 0 = −3	(−9, −3)

Чтобы Чертёжнику вернуться из точки (−9, −3) в исходную точку (0, 0), ему нужно выполнить команду: (0 − (−9), 0 − (−3)) = (9, 3). Ищем такой ответ в списке вариантов ответа.

Ответ: 4.

Второй метод решения

Понимаем, что перемещение Чертёжника по одной координате зависит только от соответствующего Смещения (a или b, в зависимости от координаты), и не зависит от значения другой координаты. Значит, можно

рассматривать суммарное смещение Чертёжника в результате выполнения алгоритма по каждой координате отдельно.

Рассматриваем смещение по координате X:

3 раза сместиться на: −2, 3 и на −4. Т. е.:

$3 \cdot (-2 + 3 - 4) = 3 \cdot (-3) = -9$.

Рассматриваем смещение по координате Y:

3 раза сместиться на: −3, 2 и на 0. Т. е.:

$3 \cdot (-3 + 2 + 0) = 3 \cdot (-1) = -3$.

Получаем, что в результате алгоритма Чертёжник суммарно сместится на (−9, −3). Чтобы вернуться обратно в исходную точку, Чертёжник должен компенсировать смещение по каждой оси. Т. е. сместиться по каждой оси на противоположное значение.

Получаем, что требуемое смещение: (9, 3).

Ответ: 4.

Задача 8. Исполнитель **Черепашка** перемещается на экране компьютера, оставляя след в виде линии. В каждый конкретный момент известно положение исполнителя и направление его движения. У исполнителя существуют две команды:

Вперёд n (где n — целое число), вызывающая передвижение Черепашки на n шагов в направлении движения.

Направо m (где m — целое число), вызывающая изменение направления движения на m градусов по часовой стрелке.

Запись

Повтори *k* раз

Команда1 Команда2 Команда3

конец

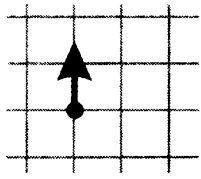
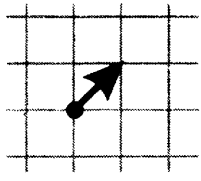
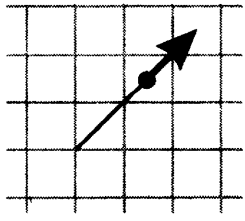
означает, что последовательность команд в скобках повторится k раз.

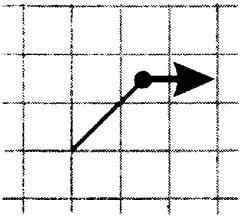
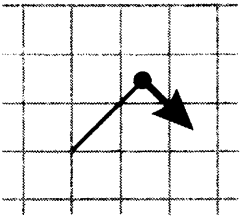
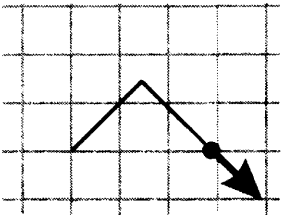
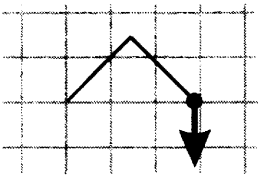
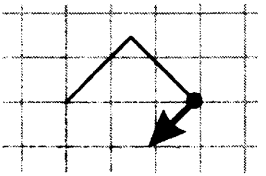
Черепашке был дан для исполнения следующий алгоритм:

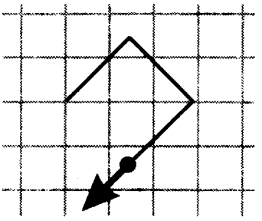
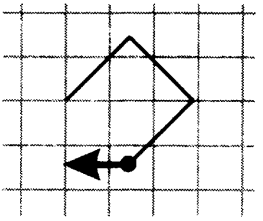
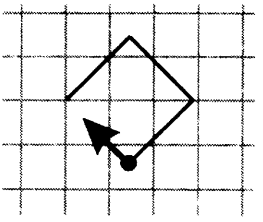
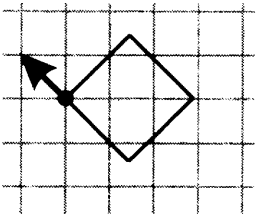
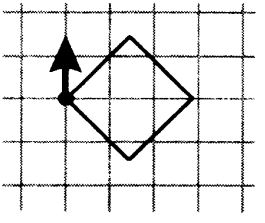
Повтори 12 раз
Направо 45 Вперёд 20 Направо 45
конец
Какая фигура появится на экране?
1. Квадрат
2. Правильный двенадцатиугольник
3. Правильный восьмиугольник
4. Незамкнутая ломаная линия

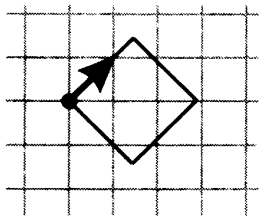
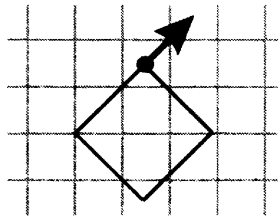
Решение

Способ первый. Выполним по действиям все команды Черепашки. Три команды повторяются 12 раз. Всего 36 команд. Довольно-таки много. Выполним эти команды по очереди. Будем рисовать линию, которую рисует Черепашка и отмечать её текущее положение и направление при помощи жирной точки и выходящей из неё стрелки.

Команда	**Положение Черепашки и след, который она за собой оставляет**
Исходное положение	
Направо 45	
Вперёд 20	

Команда	**Положение Черепашки и след, который она за собой оставляет**
Направо 45	
Направо 45	
Вперёд 20	
Направо 45	
Направо 45	

Команда	**Положение Черепашки и след, который она за собой оставляет**
Вперёд 20	
Направо 45	
Направо 45	
Вперёд 20	
Направо 45	

Команда	Положение Черепашки и след, который она за собой оставляет
Направо 45	
Вперёд 20	

К этому моменту становится ясно, что Черепашка дальше будет продолжать перемещаться по тому же самому пути, который она уже нарисовала и что дальнейшие действия Черепашки ничего не изменят в картинке. Делаем вывод, что фигура, которая появится на экране в результате этого алгоритма — квадрат.

Ответ: 1.

Второй способ решения задачи хорош в том случае, если вы изучали ранее в школе Черепашку (например, Лого Черепашку) и хорошо понимаете «Правило 360 градусов». То есть, чтобы Черепашка нарисовала правильный N-угольник со стороной X, Черепашка должна N раз повторить две команды: Вперёд X, Направо $360/N$. Анализируя алгоритм работы Черепашки в приведённой задаче, обнаруживаем, что Черепашка на каждом шаге делает одно движение Вперёд, и в начале и конце повторяющихся действий поворачивает-

ся на 45 градусов. То есть, в каждой точке поворота в сумме получается 45 + 45 градусов. Всего 90 градусов. Значит, Черепашка делает повторяющиеся действия практически эквивалентные командам «Вперёд X, Направо 90».

Если такие действия сделать хотя бы 4 раза, получится правильный четырёхугольник (360/90 = 4), то есть, квадрат.

Ответ: 1.

Программы. Оператор присваивания. Линейный алгоритм

Конспект

В этой главе рассматривается самая сложная тема курса информатики — **программирование**[1].

Основная задача процесса программирования — написать такую программу для компьютера, чтобы он выполнил требуемые действия. Проблема в том, что компьютер не умеет выполнять словесные алгоритмы, которыми пользуются люди. Компьютер умеет выполнять только ограниченный набор из нескольких сотен примитивных команд, каждая из которых имеет специальный номер (эти номера называются **машинными кодами**). Писать инструкции для компьютера при помощи машинных кодов — очень утомительное

[1] Данное пособие не может ставить своей целью полноценное обучение программированию. Мы стараемся дать только те основы, которые нужны для успешной сдачи ОГЭ по информатике. Поэтому, сознательно опускаются все типы данных, кроме целочисленного (integer) и алгоритмические конструкции, кроме базовых.

занятие, подверженное многочисленным ошибкам. Для облегчения этого процесса, люди придумали нечто среднее между формальными номерами — машинными кодами и языком человека. И называли это **языками программирования**. Язык программирования гораздо ближе к человеческому языку и на нём людям гораздо проще записывать свои алгоритмические мысли. В то же время он весьма формален и можно научить компьютер его понимать. Однако, за это удобство приходится платить двойным переводом. Сначала человек должен перевести свои алгоритмические мысли (на человеческом языке) на язык программирования (программу), а затем компьютер переводит текст программы в последовательность машинных кодов.

Мы рассматрим программирование на примере языка программирования Паскаль. Его придумал в середине 20-го века швейцарский профессор Николаус Вирт специально для обучения программированию. Это достаточно простой и удобный для начинающих язык. К тому же, он прекрасно поддерживается обоими экзаменами по информатике (ОГЭ и ЕГЭ).

Команды языка записываются при помощи некоторого набора английских слов. Даже если вы не знаете английского языка, слов используется так немного, что их легко запомнить.

Несмотря на теоретическую направленность данного пособия, изучать программирования крайне желательно на практике. То есть, программы, которые будут обсуждаться, желательно писать на компьютере, чтобы сразу запускать их и смотреть на результат. Такой процесс позволит вам использовать компьютер для поиска синтаксических ошибок и для того, чтобы сразу соотносить то, что выводит ваша программа с тем, что она должна выводить. Подобная практика решения задач позволит вам научиться самим находить ошибки в программе.

Мы рекомендуем использовать для обучения среду PascalABC. Это бесплатная среда, разрабатываемая в России специально для обучения программированию. Её можно легко скачать с сайта разработчиков. *Например*, pascalabc.net.

Программа. Линейный алгоритм

Выше уже говорилось, что язык программирования (и Паскаль, в частности) — это некое формальное описание, не допускающее толкований. Чтобы обеспечить эту формальность, язык программирования состоит из очень небольшого количества **команд**. Каждая из этих команд имеет название, которое должно быть написано именно так, буква в букву, как это придумано. Знаки препинания, которые используются при написании этих команд, также формализованы. Их нужно записывать именно по тем правилам, которые придуманы в языке.

Команды выполняются последовательно, одна за другой. Чтобы было легче читать программу и чтобы эту особенность — последовательно одна за другой — было легче воспринимать, команды принято записывать друг под другом, по одной команде в строке.

Любая программа на языке Паскаль должна иметь начало и конец. Это обозначается при помощи специальных слов Паскаля: `begin` (начало) и `end` (конец). Внимание! После слова `end`, заканчивающего программу, для указания, что это именно конец, должна стоять точка.

Самая простая программа на Паскале выглядит так:

```
begin
end.
```

Данная программа — пустая. Она ничего не делает и не содержит ни одной команды.

Рассмотрим команду языка Паскаль, которая делает что-то полезное и понятное. Это команда `write` — команда вывода на экран.

Что же выводит на экран команда `write`? То, что указано после команды `write` в скобках. Содержимое скобок называется **параметрами**.

Например, команда `write('Привет')` выводит на экран текст «Привет». Чтобы показать Паскалю, что «Привет» — это не команда, а просто текст, его нужно заключить в специальные одиночные кавычки, называемые **апострофами**. Символ апострофа располагается на клавиатуре непосредственно слева от клавиши [Enter] в английской раскладке. Программа, которая выводит на экран слово «Привет» выглядит так:

```
begin
  write('Привет')
end.
```

 Вы можете ввести эту программу в среде PascalABC в большом поле, где стоит курсор клавиатуры после запуска приложения и запустить её, нажав клавишу [F9]. В нижнем поле после этого появится результат работы программы — выведенное слово «Привет».

Рассмотрим программу, выводящую несколько фраз на экран. Например, «Привет! Как дела!?». Это можно, конечно, сделать, написав весь текст сразу в одной команде `write`, но мы специально запишем в двух разных командах, чтобы показать, как это делается. Программа будет такой:

```
begin
write('Привет!');
  write('Как дела!?')
end.
```

Обратите внимание — после первой команды `write(...)` стоит символ «`;`» (точка с запятой). Он нужен каждый раз, когда вы перечисляете несколько

команд для отделения одной команды от другой. Если забыть поставить точку с запятой, программа не запустится и компьютер скажет, что обнаружил синтаксическую ошибку.

Запустим эту программу и увидим, что обе фразы «Привет!Как дела!?» вывелись на экран слитно, в одной строке, друг за другом. Именно так работает команда `write`. После её выполнения воображаемый (обычно невидимый) **курсор вывода на экран** остаётся в конце того текста, который `write` выводит на экран.

Чтобы написать текст с новой строки, следует дать команду перевести этот курсор вывода на экран на следующую строку. Это делает другая команда языка Паскаль — `writeln`. Её название построено из двух слов — `write` (писать) и `line` (строка). Только второе слово сокращено до «`ln`». Таким образом, чтобы получить на экране текст в виде:

```
Привет!
Как дела!?
```

нужно написать такую программу:

```
begin
  write('Привет!');
  writeln;
  write('Как дела!?')
end.
```

Первый `write` выводит на экран «Привет!», вторая команда — `writeln` — переводит курсор вывода в начало следующей строки, третья команда выводит на экран «Как дела!?».

Приведённая программа является отличным примером **линейного алгоритма**. Команды записаны друг за другом и выполняются последовательно, одна за другой, от первой до последней.

Заметим, что необходимость вывести что-то на экран и затем перевести курсор вывода на следующую строку, возникает очень часто. Поэтому в языке Па-

скаль придумана специальная команда, которая и выводит на экран, и сразу после этого переводит курсор вывода на следующую строку. Эта команда — `writeln` с параметром.

А именно, вместо того, чтобы писать, как в нашем примере:

```
write('Привет');
writeln;
```

можно написать в одну команду:

```
writeln('Привет');
```

Целочисленный тип данных. Переменные

Программа, которая просто выводит на экран один и тот же тест и ничего не делает — не слишком часто нужная вещь. Обычно люди хотят от компьютера чего-нибудь бóльшего, например, чтобы он что-нибудь вычислил.

Когда требуется вычислить что-то конкретное один раз, можно просто попросить вывести результат в команде `write`. Например, вычислим и выведем на экран всем известный результат «два плюс три»:

```
begin
  write(2 + 3)
end.
```

После запуска программы на экране отобразится число 5. Как видите, команда `write` умеет выводить на экран не только текст, но и числа.

Такая программа тоже не слишком интересна. Она всегда вычисляет одно и то же. А нам хотелось бы, чтобы компьютер работал как калькулятор, и можно было бы дать программе (ввести с клавиатуры) какое-нибудь число (или несколько чисел). В свою очередь, программа пусть выполняет по этим числам некие вычисления и выдаёт на экран результат.

Для того, чтобы вводимое с клавиатуры число можно было обработать, программе это число необходимо,

хотя бы временно, сохранить. Хранение осуществляется в памяти компьютера, используя специальную технологию программирования, которая называется **переменная** (величина).

В данной главе мы рассмотрим **целочисленные переменные**. Это ячейки памяти компьютера, которые могут хранить в себе целые числа. Чтобы такое возможно было использовать в программе, требуется перед её началом (перед `begin`) сообщить компьютеру, что мы в программе собираемся использовать переменные, написав специальное служебное слово `var`. Это не команда, а слово языка Паскаль, которое означает, что с этого места начинается **раздел описания переменных** программы. После слова `var` нужно перечислить **имена переменных**, которые нужны, и указать их тип данных.

Например:

```
var a, b, vasya : integer;
begin


end.
```

Тип данных — это указание на то, какого рода информацию планируется хранить в переменных. В нашем случае мы собираемся хранить целые числа. Этот тип данных на Паскале называется `integer`.

Тип `integer` — основной тип данных языка Паскаль для хранения целочисленных данных. Если программа запускается на современном компьютере и в современной среде программирования, в ячейке типа `integer` можно хранить числа в диапазоне 2 147 483 648..+ 2 147 483 647 (чуть больше двух миллиардов по модулю). Этого диапазона начинающему вполне должно хватить. Иногда вы можете столкнуться с другими целочисленными типами данных (byte, word, longint). В учебных задачах просто считайте, что это `integer`.

Имена переменных — это то, как в программе мы хотим называть используемые ячейки памяти. В нашем примере требуется выделить для работы программы три ячейки памяти с названиями «a», «b» и «vasya». Имена переменных в Паскале придумывает тот, кто пишет программу (программист). Они должны начинаться с буквы латинского алфавита (или символа подчеркивания). Могут содержать в себе цифры, буквы латинского алфавита и знаки подчеркивания. Пробелы и знаки препинания использовать в именах переменных нельзя. Имена переменных не могут совпадать со служебными словами языка Паскаль.

Таким образом, текст программы «`var a, b, vasya : integer;`» сообщает компьютеру, что программа просит для своей работы выделить в памяти три ячейки, каждая из которых должна быть такого размера, чтобы в неё помещалось целое число типа `integer`, и к этим ячейкам программа планирует далее обращаться по именам «a», «b», «vasya».

Данные ячейки компьютер выделит в памяти в момент запуска программы и освободит, когда программа закончит свою работу.

Благодаря приведённому тексту программы: «`var a, b, vasya : integer;`», мы **описали переменные** «a», «b», «vasya». Теперь их можно использоваться в программе.

Возможно, вы заметили, что само по себе описание переменных ещё не означает, что в программе они будут использоваться. В приведённом примере программа выделяет три ячейки памяти, ничего больше не делает и заканчивает свою работу.

Рассмотрим теперь, что же можно делать с переменными типа `integer`.

Самое важное, что нужно понимать при работе с переменными — это выполнение инструкции «`:=`», называемой **оператор присваивания.**

Пример. `a := 15;`

Смысл этого оператора следующий. Необходимо вычислить выражение, которое записано в правой части оператора (после символов :=) и положить результат вычисления в переменную, имя которой записано в левой части оператора присваивания. В нашем случае выражение равно числу 15. И это число 15 будет помещено в ячейку-переменную «a».

Важно понимать, что значение, которое перед этим лежало в переменной «a», исчезнет и заменится на число 15.

Символы оператора присваивания «:=» должны быть обязательно написаны слитно, без пробела между ними.

Ещё один пример оператора присваивания:

```
b := a + 7;
```

В данном случае из ячейки-переменной «a» берётся число, которое в ней хранится в этот момент, прибавить к нему число 7 и результат кладётся в ячейку-переменную «b».

Возникает разумный вопрос: «Какое число хранится в этот момент в ячейке «a»?»

Если команда «`b := a + 7;`» выполняется после команды «`a := 15;`», то в ячейке «a» к этому моменту лежит число 15 и значение переменной «b» будет равно 22. Если же перед командой «`b := a + 7;`» не написать никакой команды, которая бы задала начальное значение переменной «a», то результат операции будет непредсказуем. Потому что, если не задать начальное значение переменной, в ячейке будет лежать неизвестное, произвольное число.

Скорее всего, конечно, там будет лежать число 0 (именно это можно увидеть, проверив значение ячейки, например, в среде PascalABC). Однако, в общем случае этого никто не гарантирует.

Не задать начальное значение переменной — грубая ошибка, за которую обычно снижают баллы на экзамене.

В команде «`b := a + 7;`» мы использовали одну из **арифметических операций**, которые можно осуществлять в Паскале с переменными типа `integer`.

Рассмотрим, какие ещё арифметические операции можно с ними производить.

```
var a, b : integer;
begin
  a := 15;
  b := a - 3;
  writeln(a,'-3=',b);
  b := a * 2;
  writeln(a,'*2=',b);
  b := a div 6;
  writeln(a,' div 6=',b);
  b := a mod 6;
  writeln(a,' mod 6=',b);
  a := a + 2;
  writeln(a)
end.
```

Результатом работы этой программы будет:

```
15 - 3 = 12
15*2 = 30
15 div 6 = 2
15 mod 6 = 3
17
```

Над целыми числами в Паскале можно осуществлять все четыре арифметические операции (сложение, вычитание, умножение и деление).

Со **сложением** и **вычитанием** всё просто — их обозначения совпадают с общепринятыми (+ и −). Операция **умножения** обозначается при помощи символа «*» (звёздочка). А вот операция **целочисленного деления** в Паскале не такая простая, как хотелось бы на

первый взгляд. Дело в том, что результат привычного нам деления не всегда бывает целый.

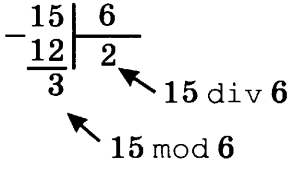
 В начальной школе учат как раз целочисленному делению, но к 9-му классу учащиеся обычно об этом забывают и считают, что 5 поделить на 2 — это всегда 2,5, а не 2 и 1 в остатке.

Так как мы рассматриваем сейчас только действия целочисленные и с целочисленными переменными, то результат обычного деления, например, «15/6» нельзя положить в целочисленную переменную, потому что он содержит целую и дробную части, а в целочисленную переменную можно положить только целое число.

Чтобы чётко описать, какой именно результат требуется получить в результате деления, в Паскале существует две операции целочисленного деления — `div` и `mod`.

Результат операции `div` — целая часть от деления. Например, `15 div 6 = 2`.

Результат операции `mod` — остаток от деления. Например, `15 mod 6 = 3`. Это легко понять, если поделить «уголком» в целых числах 15 на 6:



Обратите внимание — перед и после слов `div` и `mod` стоят пробелы, чтобы Паскаль отделял имена переменных от названий операций.

Ещё хотелось бы обратить ваше внимание на последнее присваивание в программе:

```
a := a + 2;
```

Если считать, что операция «:=» — это равенство, то приведённое выражение не имеет никакого

смысла. В математике «a» никогда не бывает равно «a + 2». В нашем случае, это ни в коем случае не равенство! Данную команду следует понимать именно как последовательность действий. Сначала нужно вычислить выражение, стоящее в правой части оператора присваивания, т. е. «a + 2». Для этого нужно взять текущее значение переменной «a» (в начале программы в переменную «a» было положено число 15) и прибавить к этому значению число 2. После, результат (15 + 2 = = 17) помещаем в ту переменную, имя которой записано в левой части оператора присваивания (переменная «a»), т. е. значение переменной «a» станет равно 17. Нетрудно видеть, что команду «a := a + 2;» следует понимать как «увеличить на 2 текущее значение переменной «a».

Кроме арифметических операций приведённый пример программы демонстрирует работу ещё одной технологии — использования нескольких параметров в команде writeln. В команде writeln (как и в команде write), можно перечислить через запятую несколько параметров. Если параметром является переменная, то на экран будет выведено её текущее значение. Так, например, команда write('a=',a); означает «вывести на экран сначала текст a=, а затем текущее значение переменной a».

При записи арифметических выражений на языке Паскаль используются те же правила, что и в математике, т. е. арифметические операции имеют приоритет (порядок выполнения операций, если в выражении встречается несколько операций). Приоритет у операций умножения и деления одинаковый между собой, но выше, чем у операций сложения и вычитания, у которых между собой приоритет тоже одинаковый. Операции одинакового приоритета выполняются слева направо.

Обратите внимание — операции div и mod — это операции деления! Их приоритет такой же, как у ум-

ножения (символ *). Они выполняются раньше сложения и вычитания и слева направо, если встречаются рядом. При необходимости изменить порядок выполнения операций используются скобки, как и в математике.

Рассмотрим ещё два действия, которые можно осуществлять на Паскале с целыми числами. Это функции `sqr` и `abs`. Слово **функция** в данном случае означает, что у каждой из них нужно указать параметр, который пишется в круглых скобках после имени функции. Эти функции возвращают целочисленный результат.

Имя функции	Возвращаемое значение	Обозначение в математике	Пример
`sqr`	Значение параметра в квадрате	x^2	`sqr(3) = 9`
`abs`	Значение модуля параметра (абсолютное значение параметра)	$\|x\|$	`abs(-7) = 7`

Пример программы:

```
var a, b : integer;
begin
  a := -6;
  b := sqr(a);
  writeln('sqr(',a,')=',b);
  b := abs(a);
  writeln('abs(',a,')=',b)
end.
```

Результатом работы этой программы будет:

```
sqr(-6) = 36
abs(-6) = 6
```

Все обсуждаемые нами до настоящего момента программы всегда делали одно и то же над одними и теми же числами. Пришла пора обсудить, как дать возможность пользователю, который будет запускать нашу программу, задать значения переменных, чтобы программа работала в зависимости от того, что указал пользователь. Для этого требуется научиться вводить значения переменных с клавиатуры.

Так же, как команда `write/writeln` позволяет программе обзаться с внешним миром и выводить значения переменных на экран, в Паскале есть команда `read/readln`, которая общается с внешним миром и спрашивает у пользователя значения переменных.

Напишем в программе команду «`read(a);`» и запустим программу. Произойдёт следующее. Когда компьютер дойдёт до выполнения строки «`read(a);`», программа приостановится, и компьютер станет ждать ввода пользователем целого числа с клавиатуры и нажатия им клавиши [Enter]. Введённое целое число будет записано в ячейку-переменную «a», компьютер продолжит выполнение программы со следующей команды.

Например, программа, которая вводит с клавиатуры целое число и выводит на экран число, которое на 7 больше введённого, будет такая:

```
var a : integer;
begin
  read(a);
  writeln(a + 7)
end.
```

Мы не стали использовать в этом решении дополнительную переменную и класть в неё результат вычисления «a + 7», чтобы потом уже вывести его на экран и показать вам приведённую технологию.

Разбор типовых задач

Задача 1. Вычислите значение выражения:
`7 + 34 mod 9 div 2*5 - 12`.

Решение

Расставим порядок вычисления операций в соответствии с их приоритетами.

В середине выражения видим две операции деления и одну — умножения. У них одинаковый приоритет, выполняются слева направо.

Важно! Из-за особенностей записи операций `div` и `mod` начинающие часто подсознательно считают, что эти операции нужно выполнять последними, Это приводит к ошибкам.

Итак, все три операции (`mod`, `div`, `*`) выполняются сначала слева направо. Затем выполняются сложение и вычитание и тоже слева направо. Получаем порядок действий:

```
  4      1      2    3  5
7 + 34 mod 9 div 2*5 - 12
```

Вычисляем по действиям. Числами в скобках здесь обозначены результаты соответствующих действий:

1. 34 `mod` 9 = 7.
2. (1) `div` 2 = 7 `div` 2 = 3.
3. (2) * 5 = 3 * 5 = 15.
4. 7 + (3) = 7 + 15 = 22.
5. (4) – 12 = 22 – 12 = 10.

Ответ: 10.

Задача 2. В программе знак «:=» обозначает оператор присваивания, знаки «+», «–», «*» и «/» — соответственно операции сложения, вычитания, умножения и деления. Правила выполнения операций и порядок действий соответствуют правилам арифметики.

Определите значение переменной a после выполнения алгоритма:

```
a := 6
b := 2
b := a/2*b
a := 2*a + 3*b
```

В ответе укажите одно целое число — значение переменной a.

Решение

Несмотря на то, что по условию задачи программа приведена на немного другом языке программирования — алгоритмическом, мы вполне уже можем справиться с её выполнением.

Небольшие отличия в данном случае состоят в том, что команды не разделяются здесь точками с запятой, как в Паскале, и вместо операции `div` используется операция «/». Для упрощения задачи разработчик специально подбирает такие числа, чтобы результат деления был всегда целым.

Чтобы свести ошибки при выполнении этой задачи (и подобных) к минимуму, рекомендуется выполнить подробную трассировку программы. Это значит, что мы составим таблицу, в которой будем записывать значения всех используемых в программе переменных и отслеживать их изменения после каждого шага программы. Саму программу будем выполнять по шагам.

Для этого составим таблицу трассировки. В ней будет столбец для записи текущей команды программы и по столбцу для каждой переменной, используемой в программе.

В данном случае в программе используется две переменных — a и b.

В столбцы переменных запишем значение только в тот момент, когда команда кладёт в переменную значение.

Команда	Переменные		Пояснения
	a	b	
a := 6	6		В результате этого простого присваивания переменная «a» получает свое начальное значение — 6. В столбец «b» мы ничего не записываем, потому что переменная «b» в этот момент не меняется. Её значение всё ещё не определено.
b := 2		2	Аналогично, переменная «b» получает начальное значение — 2. Переменная «a» в этот момент не меняется. Её значение продолжает оставаться прежним — 6.
b := a/2*b		6	Вычисляем выражение в правой части оператора присваивания. В выражении используются операции деления и умножения. У них одинаковый приоритет. Вычисляются слева направо, значит, сначала деление (a/2). Затем результат этой операции умножается на значение переменной «b». Значение переменной «a» в этот момент равно 6. «6/2» равно 3. Значение «b» в данный момент равно 2. Значение «a/2*b» = «3 · 2» = 6. Этот результат записывается в переменную «b». Переменная «a» не меняется и остаётся равной 6.

Команда	Переменные		Пояснения
	a	b	
a := 2*a + 3*b	30		Вычисляем выражение в правой части оператора присваивания. В выражении используются две операции умножения и одна — сложения. Сначала вычисляем результаты умножений, потом их складываем. Вычисляем «2*a». Значение переменной «a» в этот момент равно 6. (Если вы не помните значения переменной «a», можно «подняться взглядом» вверх в столбце переменной «a» от текущей строки до той самой нижней ячейки столбца «a», в которой написано какое-нибудь число. Это число и будет текущим значением переменной «a».) Значение «2*a» равно 12. Аналогично вычисляем «3*b». Значение переменной «b» в этот момент равно 6. (Не перепутайте — текущее значение переменной находится в самой нижней заполненной ячейки её столбца!) Значит, значение «3*b» равно 18. Складываем эти два результата. Получаем ответ 30. Его записываем в переменную «a».

По условию нас спрашивают значение переменной «a». Его мы находим как самое нижнее значение, записанное в столбце переменной «a». Это число 30.

Ответ: 30.

Программирование. Логические операции

Конспект

В данном разделе рассмотрим правила составления условий, которые умеет проверять программа на языке Паскаль[1].

Кроме вычисления арифметических выражений, расмотренных в предыдущем разделе, Паскаль умеет также вычислять логические выражения.

Логические выражения более подробно описываются на стр. 50. Здесь рассмотрим, как те же выражения можно записать на языке программирования Паскаль.

Операции сравнения

Операции сравнения, которые умеет проверять Паскаль — это привычные нам в математике операции больше, меньше, равно и прочие.

Операция сравнения	Обозначение в математике	Запись на языке Паскаль
больше	>	>
меньше	<	<

[1] Для более подробного и глубокого изучения программирования будет правильным сначала изучить логические переменные, в них научиться сохранять значения операций сравнения, и над ними выполнять логические операции.

Окончание таблицы

Операция сравнения	Обозначение в математике	Запись на языке Паскаль
равно	=	=
не равно	≠	<>
больше или равно	⩾	>=
меньше или равно	⩽	<=

Обратите внимание на необычную форму записи операции «не равно». Она записывается как «меньше или больше». Постарайтесь запомнить, например, что эта запись похожа на ромбик. Неправильная запись: «><».

Запись операций «больше или равно» и «меньше или равно» запомнить не трудно. Принцип «как слышится, так и пишется», т. е. мы говорим «больше или равно» и пишем это знак как сначала символ «больше», а затем символ «равно» (>=). Неправильная запись: «=>». Аналогично и для «меньше или равно».

При помощи операций сравнения в Паскале возможно сравнивать меду собой значения переменных, чисел и выражений. Результаты операций сравнения можно выводить на экран.

Логические операции

Для объединения в одном выражении результатов нескольких сравнений используют **логические операции**. В Паскале они работают так же, как в алгебре логики, которую мы обсуждали на стр. 47.

- **Логическое И.** Обозначение: **and**

Выполняется над двумя сравнениями.

Результат — истина, если результаты обоих сравнений — истина.

Таблица истинности:

Результат первого сравнения	Результат второго сравнения	(...) and (...)
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

• **Логическое ИЛИ.** Обозначение: `or`

Выполняется над двумя сравнениями.

Результат — истина, если результаты хотя бы одного сравнения — истина.

Таблица истинности:

Результат первого сравнения	Результат второго сравнения	(...) or (...)
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

• **Логическое НЕ.** Обозначение: `not`

Выполняется над одним сравнением.

Результат — истина, если результат сравнения — ложь, и ложь, если результат сравнения — истина.

Таблица истинности:

Результат сравнения	not (...)
FALSE	FALSE
TRUE	TRUE

Очень важно помнить, что приоритет любой логической операции в Паскале выше, чем у любой операции сравнения. Поэтому при использовании в выражении любой логической операции необходимо каждую операцию сравнения брать в скобки.

Разбор типовых задач

Задача 1. Напишите программу, которая вводит с клавиатуры целое число и выводит на экран, верно ли, что:

– это число положительное,

– это число ближе к нулю, чем к числу 100.

Пример ввода	Пример вывода
-8	FALSE TRUE
25	TRUE TRUE
80	TRUE FALSE
180	TRUE FALSE

В некоторых разновидностях сред программирования на Паскале вывод на экран `TRUE` и `FALSE` происходит как `True` и `False`. При выполнении этого и подобных заданий это не считается ошибкой.

Решение

В программе нужно ввести с клавиатуры целое число. Для этого используем целочисленную переменную:

```
var a : integer;
```

Это целое число вводим с клавиатуры:

```
begin
  read(a);
```

Проверяем, положительное ли полученное число. Это делает проверка «`a > 0`». Её результат выводим на экран:

```
writeln(a > 0);
```

Теперь нужно проверить, верно ли, что число ближе к нулю, чем к числу 100. Степень близости переменной к какому-либо значению определяется в ма-

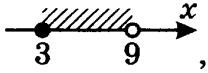
тематике как разность по модулю. Значит, вычислим разность по модулю числа с нулём и сравнить её с разностью по модулю числа с числом 100. Выводим результат на экран:

```
writeln(abs(a - 0) < abs(a - 100))
```

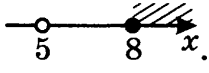
Все вместе:

```
var a : integer;
begin
  read(a);
  writeln(a > 0);
  writeln(abs(a - 0) < abs(a - 100))
end.
```

Задача 2. Напишите программу, которая вводит с клавиатуры целое число x — координату точки на числовой прямой и проверяет, принадлежит ли введённое число x заштрихованной области:

– 

,

– 

.

Пример ввода	Пример вывода
1	FALSE TRUE
4	TRUE TRUE
5	TRUE FALSE
9	FALSE TRUE
10	FALSE TRUE

Решение

Опишем целочисленную переменную x и введём её с клавиатуры:

```
var x : integer;
begin
  read(x);
```

Проверим, верно ли, что x принадлежит промежутку от 3 (включительно) до 9 (не включительно). Для этого x должен быть одновременно больше или равен 3 и строго меньше 9. Требуется, чтобы данные условия выполнялись одновременно, значит, оба эти сравнения следует объединить логической операцией `and`:

```
writeln((x >= 3)and(x < 9));
```

Ещё раз обратите внимание — каждое сравнение должно быть записано в своих скобках!

При проверке второго условия x должен оказаться либо в левой заштрихованной области, либо в правой. Чтобы x оказался в левой области, он должен быть строго меньше 5. Чтобы он оказался в правой заштрихованной области, x должен быть больше или равен 8. Нас устроит, чтобы из этих двух условий выполнилось хотя бы одно, поэтому объединим сравнения при помощи логической операции `or`:

```
writeln((x < 5)or(x >= 8))
```

Всё вместе (ответ):

```
var x : integer;
begin
  read(x);
  writeln((x >= 3)and(x < 9));
  writeln((x < 5)or(x >= 8))
end.
```

Задача 2. С клавиатуры вводится натуральное трёхзначное число. Программа должна проверить, верно ли, что у этого числа первая цифра равна последней цифре.

Пример ввода	Пример вывода
326	FALSE
282	TRUE

Решение

Сначала научимся выделять первую и последнюю цифры данного числа.

Чтобы выделить первую цифру числа, важно понимать, что нам дано трёхзначное число. Найти нужную цифру позволяет вычисление результата целой части от деления нашего числа на 100, так как первой цифрой трёхзначного числа является количество сотен в числе. На Паскале это делает операция `div 100`.

Для выделения последней цифры любого натурального числа достаточно найти остаток от деления числа на 10. На Паскале это делает операция `mod 10`.

Получаем программу (ответ):

```
var x : integer;
begin
  read(x);
  writeln(x div 100 = x mod 10)
end.
```

Программирование. Условный оператор

Конспект

До сих пор мы рассматривали только линейную программу. Команды выполнялись последовательно, друг за другом, от начала до конца, без исключений. Такой технологии недостаточно, чтобы решать множество компьютерных задач. Чаще программе требуется сделать, например, либо одно, либо другое действие, в зависимости от выполнения тех или иных условий. Для этого придуман **условный оператор `if`** — специальная команда, которая проверяет заданное условие, и, в зависимости от результата проверки, меняет течение программы.

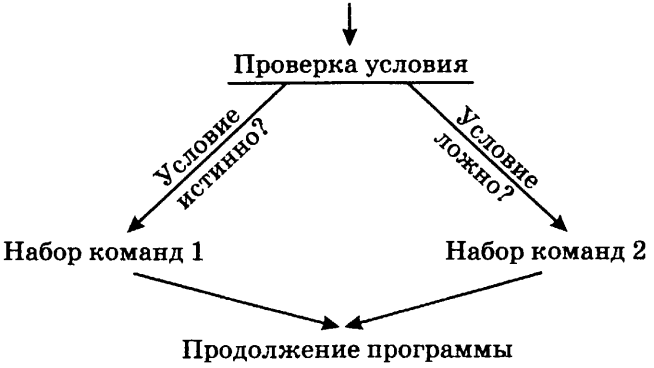
Принцип работы условного оператора `if` такой:

– выполняется проверка условия.

– если условие истинно, выполняется один набор команд.

– если условие не выполнилось, выполняется другой набор команд.

Схематично это можно представлять следующим образом:



В программе на языке Паскаль данная конструкция оформляется:

```
if <условие> then
   <оператор-да>
else
   <оператор-нет>
```

Обратите внимание, `if`, `then` и `else` — это служебные слова языка (если, то, иначе). Между `if` и `then` записывается логическое выражение, результатом которого является истина или ложь. После слова `then` должен стоять оператор, который выполняется в случае истинности условия. После слова `else` указывается оператор, который выполняется в случае, если проверяемое условие ложно.

Заметим, что независимо от того, будет проверяемое условие истинным или ложным, после окончания условного оператора компьютер выполнит команду, стоящую после этого оператора.

Вот, фрагмент программы, который проверяет, чётность некоторой переменной a и выводит на экран слово «Even». Если переменная ложна, то на экран выводится слово «Odd»:

```
if a mod 2 = 0 then
  writeln('Even')
else
  writeln('Odd')
```

Следует обратить внимание, что при написании условия (между if и then) возможно записывать не только одиночные операции сравнения, но и любые логические выражение, составленные, из нескольких логических операций.

Приведём пример фрагмента программы, который проверяет, верно ли, что целочисленная переменная a является трёхзначным натуральным числом («YES»/«NO»).

```
if (a >= 100) and (a <= 999) then
  writeln('YES')
else
  writeln('NO')
```

Возможно, вы заметили, что при рисовании схемы условного оператора мы использовали термин «набор команд», а при описании конструкции на языке Паскаль — уже термин «оператор». Это не одно и то же. Под термином «оператор» на данном этапе мы подразумеваем одну команду. А термин «набор команд» обещает, что в настоящий момент можно выполнить несколько команд. Термин «оператор» в Паскале — это не то же самое, что «команда». Хотя «команда», как правило, является оператором. В то же время, всю условную конструкцию называем «условным оператором». Примером послужит фрагмент программы:

```
if a mod 2 = 0 then
  writeln('Even')
else
  writeln('Odd')
```

Это тоже оператор, имеющий в своем составе, в частности, пару операторов, выводящих текст на экран.

Простым оператором служит одна команда. Например, оператор присваивания, вывод на экран или ввод с клавиатуры.

Ещё в Паскале есть **структурированные операторы**. Они обычно состоят из нескольких частей. Одним из таких примеров является условный оператор.

Так как же написать на Паскале, что в случае, например, истинности проверяемого условия, нужно выполнить не одну команду-оператор, а несколько!?

Для этого в Паскале есть специальный оператор, называемый **составным оператором**, или **операторной скобкой**. Это объединение нескольких операторов в единое целое, являющееся, в свою очередь одним оператором, потому что после слов `then` и `else` в операторе `if` можно написать только один оператор.

Операторной скобкой являются слово `begin`, после которого помещают любое количество разных операторов, и затем слово `end`.

Приведём пример части программы, которая проверяет, является ли переменная `a` нечётной. Если является, то переменная увеличивается на 1 и её новое значение выводится на экран. Если не является, то сначала на экран выводится её старое значение, потом из переменной вычитается 3.

```
if a mod 2 <> 0 then
begin
  a := a + 1;
  writeln(a)
end
else
begin
  writeln(a);
  a := a - 3
end
```

В приведённом примере в случаях then и в else необходимо выполнить более одной команды, поэтому в обоих случаях используется операторная скобка.

Заметим, что потеря операторной скобки является весьма частой ошибкой и на экзамене ведёт к потере балла.

Поэтому рекомендуется запомнить такое правило Паскаля: **если после do[1], then, else нужно выполнить несколько команд, необходимо использовать операторную скобку (begin и end).**

Ещё одно важное правило, касающееся синтаксиса языка Паскаль: **перед словом else не ставится символ «;» (точка с запятой)!**

Символ «;» точка с запятой на Паскале используется при перечислении операторов, которые должны быть выполнены друг за другом, чтобы отделить их друг от друга. В рассматриваемом нами условном операторе if между словом then и словом else можно написать только один оператор. В этом месте не следует ничего перечислять. При необходимости перечисления операторы объединяют операторной скобкой begin–end. Поэтому перед else точка с запятой не ставится!

Обратите внимание, операторная скобка очень похожа на пару «begin/end.», которая начинает и заканчивает любую программу на языке Паскаль. Но это не совсем одно и то же. Чтобы показать это, конец программы заканчивается точкой, а операторная скобка никогда не имеет после себя точку.

После then и else возможно не написать ни одного оператора. Например:

```
if a > 0 then
else
  writeln('<=0')
```

[1] О том, что такое «do» и когда он применяется, вы можете прочитать на стр. 113. Вкратце — это часть описания цикла.

Или:

```
if a > 0 then
  writeln('>0')
 else
    ;
```

Точка с запятой в данном случае явно показывает, что оператор после `else` отсутствует. Для этой же цели используют пустую операторную скобку:

```
if a > 0 then
  writeln('>0')
else
begin
end
```

В случае, когда не требуется выполнять никаких действий после `else`, это служебное слово опускается, т. е. в условном операторе `if` будет только раздел `then` и не будет раздела `else`. Такая форма записи называется **неполной формой оператора `if`**. В то время как при наличии обоих блоков — `then` и `else` — говорят о **полной форме оператора `if`**.

Вы уже обратили внимание, что после `then` и `else` надо писать ровно один оператор. При этом сама конструкция `if-then-else` (или `if-then`) тоже называется оператором. Это означает, что совершенно нормальной и удобной является конструкция **каскадных** или **вложенных** операторов `if`.

Например, при необходимости вывода на экран условия: является ли целочисленная переменная `a` числом положительным («positive»), отрицательным («negative») или нулём («zero»), — будет очень кстати такое решение:

```
if a > 0 then
  writeln('positive')
else
  if a < 0 then
    writeln('negative')
  else
    writeln('zero')
```

В данном примере в случае первого `else` записан один оператор — снова оператор `if`, который выбирает, какой вариант верен — меньше нуля или ноль. (`else` для условия `a > 0` означает, что $a \leqslant 0$.)

Программирование. Оператор цикла for

Конспект

При написании программы достаточно часто возникает потребность выполнить несколько раз какой-то фрагмент программы. Просто скопировать фрагмент несколько раз — не очень удобная технология, так как её можно использовать только тогда, когда количество повторений известно уже при написании программы и в процессе работы программы не меняется. Но даже в этом случае такой подход неудобен при изменении программы: когда повторяемый фрагмент нужно модернизировать, приходится это делать столько раз, сколько копий его находится в программе, либо удалять старое и копировать заново.

Гораздо более удобная технология, позволяющая повторить некоторое количество команд программы, — использовать **цикл**.

Цикл представляет собой некую повторяющуюся последовательность действий. Сами повторяющиеся при этом действия принято называть **телом цикла**.

Мы рассмотрим цикл с известным числом повторений, а именно: в процессе выполнения программы, к моменту, когда компьютер доходит до команды, начинающей цикл, точно известно, сколько раз тело цикла нужно выполнить.

На Паскале для выполнения такого вида цикла используется цикл `for`. Его ещё иногда называют **циклом со счётчиком**.

Особенность этого вида цикла состоит в том, что для своего выполнения в цикле `for` используется специальная переменная, называемая **счётчиком цикла**. Эта переменная должна быть счётного типа данных, обычно — целочисленная (`integer`).

Для работы цикла `for` указываются начальное и конечное значение счётчика.

В начале работы цикла `for` для счётчика цикла задаётся начальное значение, которое проверяется перед выполнением каждого очередного тела цикла, не изменилось ли это значение дальше конечного. Если значение счётчика не изменилось дальше конечного значения, то тело цикла выполняется один раз, затем изменяется на 1 значение счётчика и снова происходит его сравнение с конечным значением.

На Паскале это записывается следующим образом:

`for` переменная := начальное_значение `to` конечное_значение `do`
оператор-тело_цикла

Например:

```
for i := 4 to 6 do
  writeln(i)
```

Рассмотрим, как выполняется данный цикл.

Значение переменной `i` устанавливается равным 4.

Проверяется, не превысило ли значение переменной `i` конечного значения 6.

Не превысило, поэтому выполняется тело цикла — на экран выводится текущее значение переменной `i` (4).

Тело цикла (состоящее из одной команды `writeln(i)`) выполнилось, значит, переменная-счётчик цикла `i` увеличивается на 1 и становится равна 5.

Проверяется, не превысило ли значение переменной `i` конечного значения 6.

Не превысило, поэтому выполняется тело цикла — на экран выводится текущее значение переменной `i` (5).

Тело цикла (состоящее из одной команды `writeln(i)`) выполнилось, значит, переменная-счётчик цикла `i` увеличивается на 1 и становится равна 6.

Проверяется, не превысило ли значение переменной `i` конечного значения 6.

Не превысило, поэтому выполняется тело цикла — на экран выводится текущее значение переменной `i` (6).

Тело цикла (состоящее из одной команды `writeln(i)`) выполнилось, значит, переменная-счётчик цикла `i` увеличивается на 1 и становится равна 7.

Проверяется, не превысило ли значение переменной `i` конечного значения 6.

Превысило, поэтому выполнение цикла на этом заканчивается и компьютер переходит к выполнению строки программы, стоящей после цикла.


Заметим, что при программировании на Паскале лучше не использовать значение переменной-счётчика цикла после окончания цикла. Потому что, с одной стороны, в середине 90-х был принят новый стандарт языка Паскаль, который регламентирует данное значение. С другой стороны, до этого момента считалось, что переменная-счётчик цикла `for` после его окончания не определена. На всякий случай, лучше подстраховываться и считать, что она не определена.

Существует другая форма оператора `for`. В ней вместо служебного слова `to` используется служебное слово `downto`. В этом случае счётчик цикла на каждом шаге изменяется не на +1, а на −1. И при сравнении с конечным значением, соответственно, проверяется обратный знак ($\geqslant$).

Например, цикл:

```
for i := 9 downto 1 do
  writeln(i)
```

выведет на экран последовательность цифр 9, 8, 7, ..., 1 вертикально.

В качестве тела цикла `for`, как и после `then` и после `else`, должен быть написан только один оператор. Чтобы в качестве тела цикла написать несколько операторов, их необходимо объединить в операторную скобку (`begin-end`).

Пример.

Фрагмент программы выводит на экран последовательность степеней числа 10 от 0-й до 8-й:

```
a := 1;
for i := 0 to 7 do
begin
  write(a,' ');
  a := a * 10
end;
```

На экран будет выведено:

1 10 100 1000 10000 100000 1000000 10000000

В приведённом фрагменте программы используется очень важная и популярная **технология порождения последовательности чисел**. Во вспомогательную переменную `a` записывается начальное значение, равное нужному начальному значению элемента последовательности. В цикле это значение сначала выводится на экран, потом изменяется таким образом, чтобы на следующем шаге цикла оказаться равным следующему элементу последовательности. В данном случае, каждый последующий элемент последовательности можно было получить из текущего, умножив его на 10. Поэтому в программе на каждом шаге значение переменной `a` умножается на 10.

В теле цикла очень важен порядок команд. Если поменять местами строки (вывод на экран и изменение переменной `a`), то выводимая последовательность будет неверной. Начальное значение последовательности (число 1) пропадёт, зато в конце будет выведено одно лишнее значение (10^8).

Разбор типовых задач

Задача 1. Запишите значение переменной s, полученное в результате работы следующей программы. Текст программы приведён на трёх языках программирования.

Алгоритмический язык	Бейсик

алг
нач
  цел s, k
  s := 0
  нц для k от 9 до 12
  s := s + 8
  кц
  вывод s
кон
	DIM k, s AS INTEGER
	s = 0
	FOR k = 9 TO 12
	s = s + 8
	NEXT k
	PRINT s

Паскаль

```
var s, k: integer;
begin
 s := 0;
 for k := 9 to 12 do
    s := s + 8;
 writeln(s);
end.
```

Решение

Среди приведённых языков программирования нужно, конечно, выбрать тот, который вы лучше всего знаете. В нашем случае это Паскаль.

Выполним подробную и полную трассировку программы.

Составим таблицу трассировки, в которую будем записывать оператор программы, выполняемый в настоящий момент, и условие, если оно проверяется в этом операторе. Заведём столбцы для всех переменных, использующихся в программе. Столбец пояснений при-

ведём только для обучения. При выполнении трассировки его записывать не надо.

Оператор	Условие	Переменные		Пояснения
		s	k	
`s := 0;`		0		Переменная `s` получает начальное значение, равное нулю.
`for k:=9 to 12 do`	$9 \leqslant 12$? Да		9	Переменная `k` (счётчик цикла) получает начальное значение цикла (9). И сразу проверяется, можно ли выполнять тело цикла (значение счётчика цикла `k` все ещё не больше конечного значения?). Да, не больше. Значит, выполняем тело цикла.
`s:=s+8;`		8		Тело цикла состоит из одной команды. К текущему значению переменной `s` (0) прибавляется число 8 и результат (8) записывается обратно в переменную `s`.
`for k:=9 to 12 do`	$10 \leqslant 12$? Да		10	Тело цикла закончилось. Программа снова возвращается в строку `for...`. Счётчик цикла `k` увеличивается на 1 (становится равным 10) и снова проверяется, можно ли выполнять тело цикла ($k \leqslant 12$?). Да. Снова выполняем тело цикла.

Оператор	Условие	Переменные		Пояснения
		s	k	
`s:=s+8;`		16		К текущему значению переменной `s` (8) прибавляется число 8 и результат (16) записывается обратно в переменную `s`.
`for k:=9 to 12 do`	11 ⩽ 12? Да		11	Тело цикла закончилось. Программа снова возвращается в строку `for`.... Счётчик цикла `k` увеличивается на 1 (становится равным 11) и снова проверяется, можно ли выполнять тело цикла (k⩽12?). Да. Снова выполняем тело цикла.
`s:=s+8;`		24		К текущему значению переменной `s` (16) прибавляется число 8. Результат (24) записывается обратно в переменную `s`.
`for k:=9 to 12 do`	12 ⩽ 12? Да		12	Тело цикла закончилось. Программа снова возвращается в строку `for`.... Счётчик цикла `k` увеличивается на 1 (становится равным 12) и снова проверяется, можно ли выполнять тело цикла (k⩽12?). Да. Снова выполняем тело цикла.

Оператор	Условие	Переменные		Пояснения
		s	k	
`s:=s+8;`		32		К текущему значению переменной `s` (24) прибавляется число 8. Результат (32) записывается обратно в переменную `s`.
`for k:=9 to 12 do`	13 ⩽ 12? Да		13	Тело цикла закончилось. Программа снова возвращается в строку `for`... Счётчик цикла `k` увеличивается на 1 (становится равным 13) и снова проверяется, можно ли выполнять тело цикла (`k`⩽12?). Нет. Значит, тело цикла больше не выполняется. Цикл закончился. Программа переходит на строчку, записанную после тела цикла.
`writeln(s);`				На экран выводится текущее значение переменной `s` — число 32.

Ответ: 32.

Возможно, использование такой подробной трассировки покажется трудоёмким и долгим способом, но он позволяет минимизировать ошибки и понять, что же делает програм-

ма. Мы рекомендуем сначала хорошо освоить полную трассировку. И только после устойчивого хорошего результата переходить к использованию сокращённых и быстрых методов решения подобных задач.

Другой способ решения

Анализируем программу и обнаруживаем, что к начальному значению переменной `s` на каждом шаге цикла прибавляется одно и то же число (8). Из курса начальной школы нам известно, что если одно и то же число складывается определённое число раз — это операция умножения. Значит, остаётся понять, сколько раз прибавляется число 8. Это столько же раз, сколько выполняется тело цикла. Тело цикла выполняется при значениях переменной `k` от 9 до 12.

Задача посчитать, сколько раз выполняется цикл от 9 до 12, к сожалению, часто вызывает ошибку у начинающих, которые выдают ответ 3 (12 минус 9). Нетрудно посчитать, что это неверно. Чтобы не ошибиться, мы рекомендуем просто загибать пальцы при каждом значении переменной `k`: 9, 10, 11, 12. Всего получилось 4. Правильная формула — конечное значение минус начальное значение плюс один!

Итак, мы получили, что цикл выполняется 4 раза, и каждый раз к переменной `s` прибавляется число 8. То есть, переменная s увеличивается на $4 \cdot 8 = 32$.

Её начальное значение равно 0.

$0 + 32 = 32$.

Ответ: 32.

Если в теле цикла значение переменной меняется более сложным образом (например, `s := s + k;`), мы рекомендуем также воспользоваться трассировкой. Научившись без ошибочно справляться с подробной трассировкой и хорошо понимать, как что делается, можете использовать сокращённую версию, без записи текущей команды и проверяемого условия.

Задача 2. Запишите значение переменной s, полученное в результате работы следующей программы. Текст программы приведён на трёх языках программирования.

Алгоритмический язык

```
алг
нач
 цел s, k
 s := 0
 нц для k от 7 до 10
    s := s + k
 кц
 вывод s
кон
```

Бейсик

```
DIM k, s AS INTEGER
s = 0
FOR k = 7 TO 10
s = s + k
NEXT k
PRINT s
```

Паскаль

```
var s, k: integer;
begin
 s := 0;
 for k := 7 to 10 do
    s := s + k;
 writeln(s);
end.
```

Решение

Покажем, как составляется сокращенная таблица сортировки:

Переменные		Пояснения
s	k	
0		Переменная s получает начальное значение, равное нулю.
	7	Переменная k (счётчик цикла) получает начальное значение цикла (7). И сразу проверяется, можно ли выполнять тело цикла (значение счётчика цикла k все ещё не больше конечного значения?). Да, не больше. Значит, выполняем тело цикла.
7		Тело цикла состоит из одной команды. К текущему значению переменной s (0) прибавляется значение переменной k (7) и результат (7) записывается обратно в переменную s.
	8	Тело цикла закончилось. Программа снова возвращается в строку for... Счётчик цикла k увеличивается на 1 (становится равным 8) и снова проверяется, можно ли выполнять тело цикла (k ⩽ 10?). Да. Снова выполняем тело цикла.
15		К текущему значению переменной s (7) прибавляется значение переменной k (8) и результат (15) записывается обратно в переменную s.
	9	Тело цикла закончилось. Программа снова возвращается в строку for... Счётчик цикла k увеличивается на 1 (становится равным 9) и снова проверяется, можно ли выполнять тело цикла (k ⩽ 10?). Да. Снова выполняем тело цикла.

Переменные		Пояснения
s	**k**	
24		К текущему значению переменной s (15) прибавляется значение переменной k (9) и результат (24) записывается обратно в переменную s.
	10	Тело цикла закончилось. Программа снова возвращается в строку for…. Счётчик цикла k увеличивается на 1 (становится равным 10) и снова проверяется, можно ли выполнять тело цикла (k ⩽ 10?). Да. Снова выполняем тело цикла.
34		К текущему значению переменной s (24) прибавляется значение переменной k (10) и результат (34) записывается обратно в переменную s.
11		Тело цикла закончилось. Программа снова возвращается в строку for…. Счётчик цикла k увеличивается на 1 (становится равным 11) и снова проверяется, можно ли выполнять тело цикла (k ⩽ 10?). Нет. Значит, тело цикла больше не выполняется. Цикл закончился. Программа переходит на строчку, записанную после тела цикла.
		На экран выводится текущее значение переменной s — число 34.

Ещё раз обратите внимание! Сначала следует научиться выполнять полную трассировку, и только потом переходить к использованию сокращённой!

Программирование. Обработка последовательностей

После изучения данного раздела вы сможете написать программу, обрабатывающую последовательность целых чисел.

Несмотря на то, что никакой новой теории здесь не рассматривается, мы постараемся создать у вас понимание того, как нужно писать подобные программы и из каких соображений это делается.

Разбор типовых задач

Подсчёт элементов последовательности

Задача 1. Напишите программу, которая:

– вводит с клавиатуры натуральное число N. После этого вводит ещё N целых чисел.

– выводит на экран количество положительных чисел среди введённых чисел.

Пример ввода	Пример вывода
5 5 -4 0 1 -3	2
3 -4 0 -18	0
3 5 4 7	3

Решение

Будем придумывать программу последовательно. Какие для программы понадобятся переменные, будет понятно после того, как мы придумаем алгоритм.

Программа должна ввести с клавиатуры число N. Это просто и понятно:

```
read(N);
```

Теперь нужно ввести с клавиатуры ещё N целых чисел. Количество чисел заранее неизвестно. Поэтому мы не можем просто завести нужное число переменных и ввести их одним `read`, перечислив, например, через запятую. Необходимо использовать цикл.

К моменту ввода этих N чисел известно, сколько раз цикл должен выполниться — N. Поэтому используем цикл `for`. Для работы цикла `for` требуется пемеренная-счётчик цикла. Например, назовём её `i`. Её значение будет меняться от 1 до N (самый простой способ сделать так, чтобы цикл выполнился N раз). То есть:

```
for i := 1 to N do
```

Мы используем обозначение переменной N с большой буквы. Язык Паскаль не различает заглавные и маленькие буквы. Можно было бы написать эту переменную как *n*. Но по условию она называется N. Чтобы программа была понятнее, используем именно такое обозначение — с большой буквы N.

Требуется ввести с клавиатуры N целых чисел. Делаем это в цикле, который выполняется N раз. Значит, на каждом шаге цикла нужно вводить по одному числу. Используем для этого переменную `a`.

Получаем:

```
for i := 1 to N do
  read(a);
```

Однако, такой цикл только вводит с клавиатуры числа одно за другим, каждый раз в одну и ту же переменную. То есть, после ввода одного числа оно тут же пропадает и в переменную попадает следующее число. Это не совсем то, что нам нужно. Мы должны успевать анализировать каждое вводимое число на то, является ли оно положительным (ведь необходимо посчитать

количество положительных введённых чисел). Следовательно, каждое вводимое число проверим на положительность. То есть, на каждом шаге цикла нужно не только вводить число (`read(a)`), но и проверять условие. Тело цикла будет состоять не из одной команды, а, по меньшей мере, из двух. Чтобы написать такое на языке Паскаль, нужно использовать операторную скобку:

```
for i := 1 to N do
begin
  read(a);


end;
```

После ввода числа будем проверять его на положительность:

```
 if a > 0 then
```

Пора подумать, что делать, когда мы обнаружили положительное число. Чтобы посчитать количество положительных чисел, используем счётчик. Принцип его работы такой же, как загибание пальцев на руках, когда вы хотите что-то посчитать. При обнаружении нужного предмета, загибаем на руках один палец. При обнаружении следующего — ещё один. То есть, количество загнутых пальцев становится каждый раз на 1 больше. Так и работает счётчик.

Вы можете возразить — у нас уже есть один счётчик! Это счётчик цикла `for`.

Верно. Но счётчик цикла `for` нужен для обеспечения работы цикла `for`. Нам же нужен счётчик для подсчёта положительных элементов. Это должна быть отдельная переменная. При нахождении каждого положительного числа эта переменная будет увеличиваться на 1. Назовём этот счётчик буквой `k`. Тогда, увеличение переменной на 1: `k := k + 1`. То есть:

```
for i := 1 to N do
begin
  read(a);
  if a > 0 then
    k := k + 1
end;
```

В приведённом фрагменте программы отсутствует одна очень важная вещь — задание начального значения счётчика k! Когда программа обнаружит первое по счёту положительное число, она должна будет выполнить команду «k := k + 1», т. е. взять текущее значение переменной k, прибавить к нему 1 и сделать результат сложения новым значением переменной k. Но к этому моменту переменная k ничему не равна! Это ошибка. Обязательно перед циклом нужно задавать начальное значение для подобных переменных.

В нашем случае до начала цикла мы ещё не ввели ни одного элемента последовательности и, соответственно, не нашли ни одного положительного среди них. Значит, количество найденных положительных элементов перед началом цикла должно быть равно нулю:

```
k := 0;
```

Это действие необходимо выполнить перед циклом.

Программа почти готова. Осталось только не забыть вывести на экран ответ:

```
writeln(k)
```

Теперь самое время уточнить, что для работы программы все использованные переменные должны быть описаны в разделе var. Вспоминаем, какие переменные мы использовали: N — количество вводимых элементов, i — счётчик цикла for, a — вводимый и проверяемый элемент последовательности, k — счётчик количества положительных элементов. Получаем все вместе:

```
var N, i, a, k : integer;
begin
  read(N);
  k := 0;
  for i := 1 to N do
  begin
    read(a);
    if a > 0  then
      k := k + 1
  end;
  writeln(k)
end.
```

Очень важное (и часто забываемое) правило: **перед началом цикла обязательно нужно задать начальные значения для всех переменных, которые накапливаются в цикле или с которыми в цикле происходят сравнения!**

Поиск максимума (минимума) последовательности

Задача 1. Напишите программу, которая:

– вводит в клавиатуры натуральное число N. После этого вводит ещё N целых чисел;

– выводит на экран максимальное среди введённых чисел.

Пример ввода	Пример вывода
5 5 -4 0 1 -3	2
3 -4 0 -18	0
3 5 4 7	3

Решение

Для нахождения максимума последовательности воспользуемся такой технологией.

Заведём отдельную переменную, в которой будем хранить значение текущего максимума. То есть, на каждом шаге цикла в этой переменной будет храниться самое большое значение среди всех значений, встретившихся к текущему моменту.

На каждом шаге цикла сравниваем текущий элемент последовательности (введённое с клавиатуры число) со значением текущего максимума. Если окажется, что только что введённое число лучше (больше) текущего максимума, то это число положим новым текущим значением максимума. К моменту окончания цикла значение, лежащее в текущем максимуме будет максимальным значением элементов последовательности.

В этом алгоритме пока отсутствует один важный момент — неизвестно, какое начальное значение нужно положить в текущий максимум (перед циклом), чтобы с ним можно было сравнивать все элементы последовательности!?

Ответ на этот вопрос зависит от условия задачи. Если в задаче нам известно ограничение на диапазон значений элементов последовательности, то в качестве начального значения максимума нужно взять значение, выходящее (в меньшую сторону) за указанный диапазон.

Если же такое ограничение не известно, в качестве начального значения максимума нужно взять первый элемент последовательности. Так как начальное значение нужно задавать перед циклом, для обеспечения этой технологии необходимо перед циклом отдельно ввести с клавиатуры значение первого элемента и положить его в качестве начального значения максиму-

ма. Затем запустить цикл, который начинается уже не с 1-го элемента, а со второго.

Представленный нами алгоритм не будет работать, если требуется найти максимум с условием (например, максимальный отрицательный элемент последовательности), но при этом не задано ограничение на диапазон значений элементов последовательности. В этом случае нужно сначала найти первый элемент, удовлетворяющий условию, принять его значение в качестве начального значения максимума, а затем уже, начиная со следующего элемента, искать максимум с условием. Однако, такую задачу обычно не задают учащимся на данном этапе обучения.

В условии нашей задаче не задано ограничение на диапазон возможных значений элементов массива, поэтому воспользуемся второй технологией — возьмём в качестве начального значения максимума первый элемент последовательности.

Приступим к написанию программы.

Для хранения текущего значения максимума используем переменную `max`. Остальные переменные возьмём из тех же соображений, что и при решении задачи о подсчёте элементов последовательности: `N` — количество вводимых чисел (длина последовательности), `i` — счётчик цикла `for`, `a` — вводимое в цикле число (текущий элемент последовательности).

Перед циклом побеспокоимся о том, чтобы начальное значение максимума было равно первому элементу последовательности.

Для этого введём первый элемент последовательности и положим его значение в `max`:

```
read(a);
max := a;
```

Можно сократить эти две команды до одной:

```
read(max);
```

В данном случае вводимое число сразу попадает в переменную max.

Так как первый элемент последовательности уже ввели и обработали, цикл for будет теперь начинаться с двух:

```
for i := 2 to N do
```

В цикле нужно будет вводить числа и анализировать их. То есть, выполнять более одного оператора. Значит, нужно не забыть использовать операторную скобку (begin-end). Проверяем введённое число (a), не является ли оно больше, чем текущее значение max:

```
 if a > max then
```

Если оказывается, что введённое число a больше текущего значения max, то число a записываем в качестве нового значения max:

```
    max := a
```

После окончания цикла выводим ответ на экран.

Все вместе:

```
var N, i, a, max : integer;
begin
  read(N);
  read(max);
  for i := 2 to N do
  begin
    read(a);
    if a > max then
      max := a
  end;
  writeln(max)
end.
```

Благодаря чему эта программа ищет именно наибольшее значение последовательности, а не наименьшее? Благодаря тому, что мы назвали переменную именем max? Конечно, нет. Мы использовали такое имя, чтобы программа была понятнее. Поиск именно

максимума происходит из-за знака, стоящего в условном операторе `if`. Там проверяется, не является ли введённая переменная `a` больше текущего максимума.

Если в задаче потребуется искать наименьшее значение последовательности вместо наибольшего, то нужно просто поменять знак сравнения с «больше» на «меньше». А для понятности программы желательно, кроме этого, ещё переименовать переменную `max` в переменную `min`.

Задача 2. Напишите программу, которая в последовательности натуральных чисел определяет минимальное число, оканчивающееся на 4. Программа получает на вход количество чисел в последовательности, а затем сами числа.

В последовательности всегда имеется число, оканчивающееся на 4.

Количество чисел не превышает 1000. Введённые числа не превышают 30 000.

Программа должна вывести одно число — минимальное число, оканчивающееся на 4.

Пример ввода	Пример вывода
3 24 14 34	14
3 85 14 24	14
3 14 24 8	14

Решение

За основу алгоритма возьмём алгоритм нахождения максимума, рассмотренный нами выше. Только при сравнении с текущим значением будем, проверять условие не «больше», а «меньше».

В данной задаче требуется найти не просто минимальное число последовательности, а минимальное число с условием (число должно оканчиваться на 4). Значит, при вводе очередного числа нужно не только сравнивать его с максимумом, а и не забывать проверять, что это число подходит под условие (оканчивается на 4).

Очень важное обстоятельство, которое необходимо помнить при поиске максимума с условием — ни в коем случае нельзя принимать значение первого элемента последовательности в качестве начального значения максимума! Потому что, если первый элемент не удовлетворяет условию, но при этом является, например, наибольшим элементом последовательности, то такая программа найдёт именно первый элемент в качестве искомого, что неверно.

Например, в данной задаче для входных данных:

```
3
56 14 24
```

такая программа возьмёт в качестве начального значения максимума число 56. При сравнении с ним числа 14 и 24 (оканчивающиеся на 4) не будут больше 56, и поэтому не будут найдены.

Однако, в задаче задано ограничение на диапазон возможных значений элементов последовательности: известно, что введённые числа — натуральные и не превышают 30 000.

Значит, в качестве начального значения максимума можно взять значение, выходящее за допустимый диапазон значений элементов.

Так как мы должны найти минимум, нужно брать значение, выходящее за диапазон значений сверху, т. е. такое значение, которое гарантированно больше значения любого элемента последовательности. Для того, чтобы при сравнении начального значения минимума с любым подходящим под условие элементом последовательности, условие «меньше» обязательно выполнилось и в качестве нового значения минимума с этого момента можно было взять текущий, подходящий под условие, элемент. Таким значением в нашем случае можно взять число 30 001. Оно как раз больше любого элемента последовательности.

В результате, перед циклом в качестве начального значения минимума возьмём 30 001:

```
min := 30001;
```

Так как в качестве начального значения минимума используется просто постоянное значение, а не первый элемент последовательности, то цикл по вводу и проверки элементов нужно запускать с первого элемента до последнего (от 1 до N).

В цикле после ввода очередного элемента не забудем проверить, что элемент оканчивается на 4. Для этого необходимо проверять на равенство четырём последнюю цифру числа. Чтобы извлечь из числа последнюю цифру, нужно найти остаток от деления числа на 10. То есть:

```
if a mod 10 = 4 then
```

Объединяем эти соображения в одну программу:

```
var N, i, a, min : integer;
begin
  read(N);
  min := 30001;
  for i := 1 to N do
  begin
    read(a);
```

```
      if a mod 10 = 4 then
        if a < min then
          min := a
    end;
    writeln(min)
  end.
```

Программирование. Обработка массивов

Конспект

Введение

При написании программ достаточно частая задача — хранить значительное количество однотипных данных. Например, нужно хранить сведения о росте всех 30-ти учащихся класса. С одной стороны, можно воспользоваться уже известной нам технологией — описать 30 отдельных переменных и в каждую положить рост соответствующего учащегося. Однако, такой подход очень неудобен при дальнейшем использовании. В любых операциях с этими 30-ю переменными (например, нахождением наибольшего роста), следует перечислять их всех и с каждой делать совершенно одинаковые действия.

Мы знаем, что для повторения одинаковых действий в программировании специально придумали циклы. Однако, при хранении данных в отдельных переменных использовать цикл не получится — нужно каждый раз обращаться к разным переменным, и в цикл это не оформить.

Для того, чтобы можно было не только хранить, но и удобно обрабатывать однотипную информацию, придумали **массивы**. В примере с ростами 30-ти учащихся, при использовании массива тоже выделяется 30 ячеек памяти. Только все эти ячейки имеют общее имя, и при том у каждой из них — свой собственный номер.

Массив — совокупность однотипных данных, расположенных подряд и имеющих общее имя. При этом каждый элемент массива имеет собственный номер, называемый **индексом элемента массива**, а то, что хранится в этом элементе — **значением элемента массива**.

При использовании массивов имеется некоторое количество типовых операций, которые осуществляются с массивом и его элементами. Большинству программ, использующих массивы, необходимо для выполнения задачи, как правило, какое-то количество этих операций. Мы рассмотрим их далее. При написании программы, обрабатывающей массив, достаточно просто выбрать из списка нужный набор фрагментов программы, и составить их друг за другом.

Описание массива

Любые переменные, включая массивы, должны быть описаны в программе в разделе описания переменных `var`. При описании массива следует указывать: имя новой переменной (массива), диапазон значений индексов элемента массива и тип данных, который хранится в элементах массива.

Пример

```
var A : array [1..10] of integer;
```

Здесь описан массив с именем A, состоящий из 10-ти элементов, пронумерованных числами от 1 до 10. В элементах массива хранятся целые числа. Обра-

тите внимание, слова **array** (переводится как **массив**) и **of** (переводится как **из**) являются ключевыми словами языка программирования Паскаль.

Повторимся: при **описании массива** нужно указать:

– имя массива (правила записи имени массива такие же, как для имён переменных);

– символ «:» (как и при описании переменных, между именем/именами описываемых переменных и именем их типа данных необходимо ставить двоеточие);

– ключевое слово «**array**» (**массив**);

– в квадратных скобках два числа — начальное и конечное значение индексов, которыми будут нумероваться элементы массива. Между этими значениями должно быть написано «. .» (две точки). Как правило, в качестве начального значения индекса используют число 1, а в качестве конечного значения индекса используют требуемое количество элементов массива;

– ключевое слово «**of**» (**из**);

– имя типа данных, который будут иметь все элементы этого массива. В справочнике разбирается только тип данных `integer` (см. стр. 88).

Задание начальных значений элементов массива

Перед тем как массив можно будет использовать/обрабатывать, элементы массива должны получить значения. Как и для одиночных переменных, эти значения могут быть присвоены оператором присваивания или введены с клавиатуры. Однако, в отличие от отдельных переменных, элементов массива много.

Если способ задания значения для каждого элемента одинаковый, почти всегда можно использовать для этого цикл `for`, перебирающий индексы элементов массива от первого до последнего.

Обнуление всех элементов массива.

Перебираем все элементы массива и в каждый кладём ноль:

```
for i := 1 to 10 do
  A[i] := 0;
```

Обратите внимание — эта строка (`for i := 1 to 10 do`), перебирающая все элементы массива от первого индекса до последнего — будет использоваться практически во всех алгоритмах, обрабатывающих массив.

Ввод массива с клавиатуры.

Как и выше, требуется перебрать индексы всех элементов массива и ввести каждый с клавиатуры:

```
for i := 1 to 10 do
  read(A[i]);
```

Задание элементов массива определённой последовательностью.

Заполним элементы массива последовательностью степеней числа 10 от нулевой до 9-й. Используем для этого вспомогательную переменную, в которой будем последовательно получать значение очередного элемента.

```
b := 1;
for i := 1 to 10 do
begin
  A[i]:=b;
  b := b * 10
end;
```

Немного другой способ похожего решения — использовать для вычисления очередного элемента значение предыдущего элемента массива. Значение первого элемента зададим перед циклом отдельно.

Каждый последующий получаем, умножая значение предыдущего элемента на 10:

```
A[1] := 1;
for i := 2 to 10 do
  A[i] := A[i-1] * 10;
```

Заметим, что это решение, менее понятно только начинающим программировать.

Вывод массива на экран

Под термином «вывод массива на экран» понимается, вывод на экран всех элементов массива. В Паскале нельзя написать «`writeln(A)`», чтобы все значения массива вывелись на экран. Как и в предыдущих случаях, требуется перебрать по очереди индексы всех элементов массива и вывести каждый на экран:

```
for i := 1 to 10 do
  write(A[i],' ');
writeln;
```

В данном примере в цикле перебираются все элементы массива и каждый выводится на экран. Чтобы значения элементов не выводились вплотную друг к другу и не «склеивались» (т. е., чтобы отделить визуально на экране одни значения от других), после вывода каждого элемента выводим символ пробела.

По окончании цикла ставим переход на следующую строку, чтобы последующие выводы программы на экран происходили с новой строки и не путались, попадая в конец строки, в который мы вывели массив.

Ещё раз обратите внимание! Программа должна выполнять именно то, что написано. Например, можно вывести на экран требуемые числа, просто запустив цикл от 10 до 1 в обратном порядке. Или заполнить массив числами от 1 до 10, а потом вывести элементы массива на экран в обратном

порядке. Первое и второе решения — неверные. Программа должна заполнить массив указанными числами, а потом вывести результирующий массив в прямом порядке.

Подсчёт количества элементов массива

Посчитаем количество нечётных элементов массива. Для этого заведём переменную-счётчик. Перед циклом обнулим эту переменную. В цикле будем перебирать по очереди все элементы массива и проверять текущий элемент массива на нечётность. Если условие выполняется — увеличиваем счётчик на 1.

```
k := 0;
for i := 1 to 10 do
  if A[i] mod 2 <> 0 then
    k := k + 1;
writeln(k);
```

Вычисление суммы положительных элементов массива

Как и в случае с подсчётом количества элементов, создадим переменную для подсчёта суммы нужных элементов. Перед циклом не забудем присвоить ей начальное значение — ноль. В цикле переберём по очереди все элементы массива и проверим их на положительность. Если условие выполняется, добавляем значение проверенного элемента к сумме.

```
sum := 0;
for i := 1 to 10 do
  if A[i] > 0 then
    sum := sum + A[i];
writeln(sum);
```

Нахождение максимального элемента массива

Как и при обработке последовательностей чисел, поиск максимального элемента массива выполняется по той же технологии. Создаётся переменная, предназначенная для хранения текущего значения максимального элемента массива. В качестве начального значения для этой переменной выбирается либо значение первого элемента массива, либо значение, выходящее за допустимый диапазон значений элементов массива.

После этого перебираются все оставшиеся элементы массива.

Каждый из них сравнивается с текущим значением максимума. Если текущий элемент оказывается больше, то текущее значение максимума меняется.

Рассмотрим основные реализации этой технологии.

1. Диапазон значений элементов не задан. Ищем максимум среди всех элементов. Начальное значение максимума — первый элемент массива:

```
max := A[1];
for i := 2 to 10 do
  if A[i] > max then
    max := A[i];
```

2. Диапазон значений задан (например, −1000... +1000). Ищем максимум среди чётных элементов. Начальное значение максимума — выходящее за диапазон число (−1001):

```
max := -1001;
for i := 1 to 10 do
  if A[i] mod 2 = 0 then
    if A[i] > max then
      max := A[i];
```

3. При поиске максимума в массиве иногда требуется найти не значение максимального элемента, а номер максимального элемента. Если такую задачу нуж-

но решать для последовательности чисел, то придётся хранить оба значения — и номер максимума, и значение максимума. Но в массиве достаточно хранить только номер, потому что зная номер элемента массива, всегда можно получить значение этого элемента, написав «A[номер]».

Ещё одна особенность при поиске номера максимального элемента — таких максимальных элементов в массиве может быть несколько. Поэтому в задаче должно быть отдельно сказано, что либо все элементы массива различные, либо при наличии нескольких максимальных элементов нужно найти, например, наименьший номер максимального элемента.

Найдём наименьший номер минимального элемента массива (используем для хранения этого номера переменную imax):

```
imax := 1;
for i := 2 to 10 do
  if A[i] > A[imax] then
    imax := i;
```

4. Если нужно найти номер максимального элемента массива с условием, то решение усложняется, потому что нельзя использовать обращение к текущему максимальному элементу как «A[imax]», пока imax не будет хранить номер элемента массива, удовлетворяющего условию.

В этом случае удобнее не экономить на одной переменной, а хранить как номер максимального элемента, так и его значение.

Пример.

Найдём наименьший номер максимального нечётного элемента массива. Известно, что значения всех элементов массива лежат в диапазоне −1000...+1000.

```
max := -1001;
imax := 0;
for i := 1 to 10 do
  if A[i] mod 2 <> 0 then
    if A[i] > max then
    begin
      imax := i;
      max := A[i]
    end;
```

Задание значений элементов массива числами, не связанными закономерностью

Если массив надо заполнить числами, которые никак друг с другом не связаны, то всегда можно написать столько операторов присваивания, сколько имеется элементов массива.

Допустим, требуется заполнить массив числами: 8, −102, 0, 754, 25, 71, 333, 94, 7, 19:

```
A[1]:=8;  A[2]:=-102; A[3]:=0;   A[4]:=754;
A[5]:=25; A[6]:=71;   A[7]:=333; A[8]:=94;
A[9]:=7;  A[10]:=19;
```

Разбор типовых задач

В таблице `Dat` представлены данные о количестве голосов, поданных за 10 исполнителей народных песен (`Dat[1]` — количество голосов, поданных за первого исполнителя; `Dat[2]` — за второго и т. д.). Определите, какое число будет напечатано в результате работы следующей программы.

Текст программы приведён на трёх языках программирования.

Алгоритмический язык	Бейсик

алг
нач
целтаб Dat[1:10]
цел k, m
Dat[1] := 16
Dat[2] := 20
Dat[3] := 20
Dat[4] := 41
Dat[5] := 14
Dat[6] := 21
Dat[7] := 28
Dat[8] := 12
Dat[9] := 15
Dat[10] := 35
m := 0
нц для k от 1 до 10
если Dat[k] > m то
m := Dat[k]
все
кц
вывод m
кон
	DIM Dat(10) AS INTEGER
	DIM k, m AS INTEGER
	Dat(1) = 16
	Dat(2) = 20
	Dat(3) = 20
	Dat(4) = 41
	Dat(5) = 14
	Dat(6) = 21
	Dat(7) = 28
	Dat(8) = 12
	Dat(9) = 15
	Dat(10) = 35
	m = 0
	FOR k = 1 TO 10
	IF Dat(k) > m THEN
	m = Dat(k)
	ENDIF
	NEXT k
	PRINT m

Паскаль

```
var k, m: integer;
Dat: array[1..10] of integer;
begin
 Dat[1] := 16;
 Dat[2] := 20;
 Dat[3] := 20;
 Dat[4] := 41;
 Dat[5] := 14;
 Dat[6] := 21;
 Dat[7] := 28;
 Dat[8] := 12;
 Dat[9] := 15;
 Dat[10]:= 35;
 m := 0;
 for k := 1 to 10 do
  if Dat[k]>m then
  begin
    m := Dat[k]
  end;
 writeln(m);
end.
```

Решение

Выполним подробную трассировку программы. В таблицу трассировки мы не станем записывать значения всех 10-ти элементов массива и выделять под это, соответственно, 10 столбцов, так как очевидно, что эти значения задаются один раз в начале программы, и в дальнейшем не меняются.

Оператор	Условие	Переменные		Пояснения
		m	k	
m:=0;		0		Переменная m получает начальное значение, равное нулю.
for k:=1 to 10 do	1⩽10? Да		1	Начальное значение счётчика цикла
if Dat[k]>m then	16>0? Да			В этот момент k = 1. Обращаемся к элементу Dat[k], то есть Dat[1]. Элемент Dat[1] равен 16.
m:=Dat[k]		16		Меняем переменную m
for k:=1 to 10 do	2⩽10? Да		2	Значение счётчика цикла равно 2
if Dat[k]>m then	20>16? Да			В этот момент k = 2. Обращаемся к элементу Dat[k], то есть Dat[2]. Элемент Dat[2] равен 20.
m:=Dat[k]		20		Меняем переменную m
for k:=1 to 10 do	3⩽10? Да		3	Значение счётчика цикла равно 3
if Dat[k]>m then	20>20? Нет			В этот момент k = 3. Обращаемся к элементу Dat[k], то есть Dat[3]. Элемент Dat[3] равен 20. Значение m не меняется.

Оператор	Условие	Переменные		Пояснения
		m	k	
`for k:=1 to 10 do`	4⩽10? Да		4	Значение счётчика цикла равно 4
`if Dat[k]>m then`	41>20? Да			В этот момент `k` = 4. Обращаемся к элементу `Dat[k]`, то есть `Dat[4]`. Элемент `Dat[4]` равен 41.
`m:=Dat[k]`		41		Меняем переменную `m`
`for k :=1 to 10 do`	5⩽10? Да		5	Значение счётчика цикла равно 5
`if Dat[k]>m then`	14>41? Нет			В этот момент `k` = 5. Обращаемся к элементу `Dat[k]`, то есть `Dat[5]`. Элемент `Dat[5]` равен 14. Значение `m` не меняется.
`for k:=1 to 10 do`	6⩽10? Да		6	Значение счётчика цикла равно 6
`if Dat[k]>m then`	21>41? Нет			В этот момент `k` = 6. Обращаемся к элементу `Dat[k]`, то есть `Dat[6]`. Элемент `Dat[6]` равен 21. Значение `m` не меняется.
`for k:=1 to 10 do`	7⩽10? Да		7	Значение счётчика цикла равно 7

Оператор	Условие	Переменные		Пояснения
		m	k	
`if Dat[k]>m then`	28>41? Нет			В этот момент `k` = 7. Обращаемся к элементу `Dat[k]`, то есть `Dat[7]`. Элемент `Dat[7]` равен 28. Значение `m` не меняется.
`for k:=1 to 10 do`	8⩽10? Да		8	Значение счётчика цикла равно 8
`if Dat[k]>m then`	12>41? Нет			В этот момент `k` = 8. Обращаемся к элементу `Dat[k]`, то есть `Dat[8]`. Элемент `Dat[8]` равен 12. Значение `m` не меняется.
`for k:=1 to 10 do`	9⩽10? Да		9	Значение счётчика цикла равно 9
`if Dat[k]>m then`	15>41? Нет			В этот момент `k` = 9. Обращаемся к элементу `Dat[k]`, то есть `Dat[9]`. Элемент `Dat[9]` равен 15. Значение `m` не меняется.
`for k:=1 to 10 do`	10⩽10? Да		10	Значение счётчика цикла равно 10

Оператор	Условие	Переменные		Пояснения
		m	k	
`if Dat[k]>m then`	35>41? Нет			В этот момент k = 10. Обращаемся к элементу Dat[k], то есть Dat[10]. Элемент Dat[10] равен 35. Значение m не меняется.
`for k:=1 to 10 do`	11⩽10? Нет		11	Значение счётчика цикла равно 11. Цикл заканчивается.
`writeln(m);`				На экран выводится значение переменной m — 41

Ответ: 41.

Другой вариант решения задачи

Анализируем предлагаемую программу и видим, что приведённый алгоритм совпадает с уже знаковым нам алгоритмом нахождения максимального элемента массива. Среди значений элементов массива находим наибольшее значение. Это 41.

Ответ: 41.

Второй метод решения хорош только в том случае, когда приведённая в условии программа совпадает, с точностью до обозначений, с известным нам алгоритмом. Если же программа в условии содержит хоть ка-

кие-то, пусть даже на первый взгляд незначительные отличия, пользоваться таким методом опасно. Легко может оказаться, что это небольшое отличие в корне изменит результат работы программы. Например, на приведённых в условии данных.

Метод выполнения трассировки программы кажется нам гораздо надежнее и предпочтительнее.

Если вы хорошо умеете делать подробную трассировку и желаете сэкономить время, воспользуйтесь краткой трассировкой, без выписывания выполняемой команды и проверяемого условия. Просто заносите в столбцы таблицы изменяющиеся значения переменных.

Программирование робота

В этом разделе рассмотрим задачу, где требуется написать программу для исполнителя **Робот**, который ездит по прямоугольному полю со стенками.

Разбор типовых задач

Исполнитель **Робот** умеет перемещаться по лабиринту, начерченному на плоскости, разбитой на клетки. Между соседними (по сторонам) клетками может стоять стена. Через неё Робот пройти не может.

У Робота есть девять команд. Четыре команды — это команды-приказы:

вверх вниз влево вправо

При выполнении любой из этих команд Робот перемещается на одну клетку соответственно: вверх ↑,

вниз ↓, влево ←, вправо →. Если Робот получит команду передвижения сквозь стену, то он разрушится.

Также у Робота есть команда **закрасить**, при которой закрашивается клетка, где Робот находится в настоящий момент.

Ещё четыре команды — команды проверки условий. Эти команды проверяют, свободен ли путь для Робота в каждом из четырёх возможных направлений:

сверху свободно
снизу свободно
слева свободно
справа свободно

Данные команды можно использовать вместе с условием **«если»**, имеющим следующий вид:

если *условие* **то**
последовательность команд
все

Здесь **условие** — одна из команд проверки условия.

Последовательность команд — это одна или несколько любых команд-приказов.

Например, для передвижения на одну клетку вправо, если справа нет стенки, и закрашивания клетки можно использовать такой алгоритм:

если справа свободно то
вправо
закрасить
все

В одном условии можно использовать несколько команд проверки условий, применяя логические связки **и, или, не**, например:

если (справа свободно) и (не снизу свободно) то
вправо
все

Для повторения последовательности команд можно использовать цикл **«пока»**, имеющий следующий вид:

нц пока *условие*
последовательность команд
кц

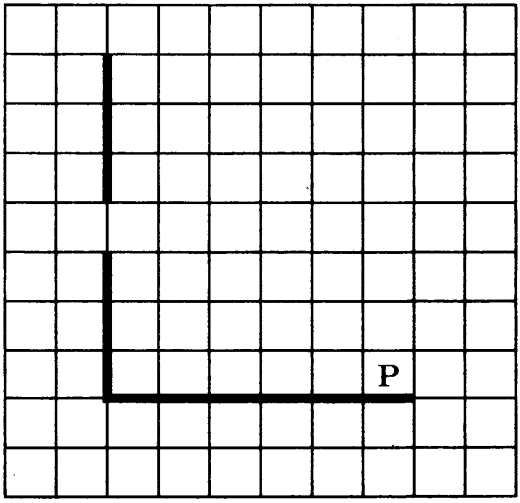
Например, для движения вправо, пока это возможно, используется следующий алгоритм:

нц пока справа свободно
вправо
кц

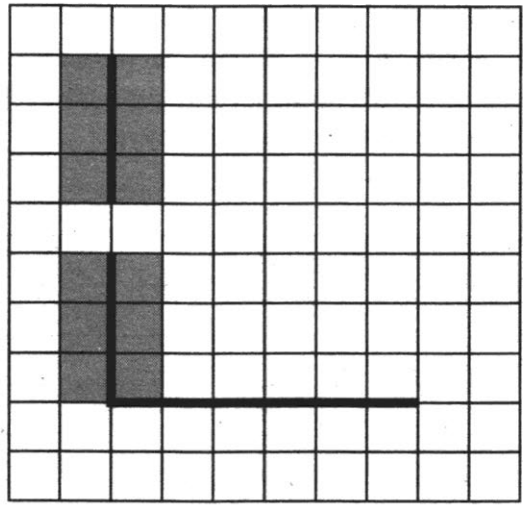
Выполните задание.

На бесконечном поле есть горизонтальная и вертикальная стены. Левый конец горизонтальной стены соединён с верхним концом вертикальной стены. Длины стен неизвестны. В вертикальной стене есть ровно один проход, точное место прохода и его ширина неизвестны. Робот находится в клетке, расположенной непосредственно над горизонтальной стеной у её правого конца.

На рисунке указан один из возможных способов расположения стен и Робота (Робот обозначен буквой «Р»).



Напишите для Робота алгоритм, закрашивающий все клетки, расположенные непосредственно левее и правее вертикальной стены. Робот должен закрасить только клетки, удовлетворяющие данному условию. Например, для приведённого выше рисунка Робот должен закрасить следующие клетки:



При исполнении алгоритма Робот не должен разрушиться, иначе выполнение алгоритма должно завершиться. Конечное расположение Робота может быть произвольным.

Алгоритм должен решать задачу для любого допустимого расположения стен.

Алгоритм может быть выполнен в среде формального исполнителя или записан в текстовом редакторе.

Сохраните алгоритм в текстовом файле.

В рассматриваемой задаче необходимо уметь писать программу на языке команд Робота, а также уметь пользоваться такими алгоритмическими конструкциями, как условие и цикл. И то и другое подробно раз-

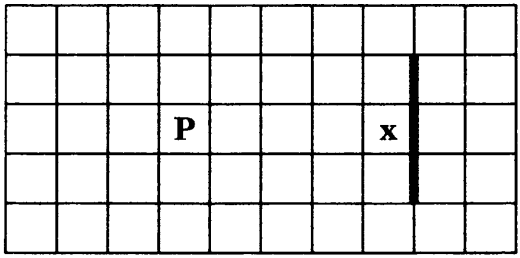
бирается в главах про Паскаль (см. стр. 83). Попробуйте применить навыки их использования и для этого исполнителя.

Для начала рассмотрим основные подзадачи, которые возникают при решении задач с Роботом.

Подзадача 1.

Робот находится на некотором (неизвестном) расстоянии до стены. Он должен доехать до стены. Требуемая позиция отмечена знаком «**x**».

Пример:



Решение подзадачи 1

На каждом шаге Робот может двигаться только на одну клетку. В данной подзадаче Роботу требуется двигаться вправо, пока он не доедет до стены.

Данный русский текст возьмём за основу программы.

Требуется выполнять повторяющиеся действия (движение вправо), значит, будем использовать цикл.

Нам (и Роботу тоже) заранее не известно, сколько шагов вправо нужно сделать Роботу, чтобы добраться до стены. Однако, в списке команд Робота именно для таких целей есть специальная инструкция — цикл **пока** с неизвестным числом повторений. Он выполняется до тех пор, пока условие, указанное после слова **пока**, является истинным.

Итак, нам ясно, что требуется использовать цикл **пока**, что в теле цикла нужно двигаться по одной клетке **вправо**. Остаётся только определить, какое условие следует написать после слова **пока**, а именно: какое условие должно выполняться, чтобы Робот мог сделать очередной шаг вправо. Очевидно, справа от Робота для этого должна быть свободная клетка. Если это не будет, а Робот сделает шаг вправо, то он врежется в стену и сломается.

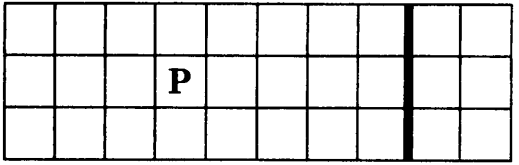
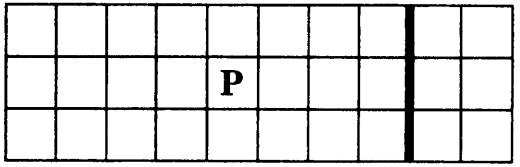
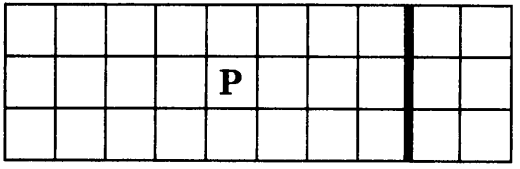
Получаем такую идею:

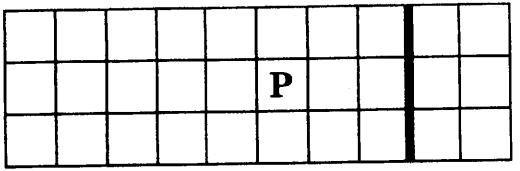
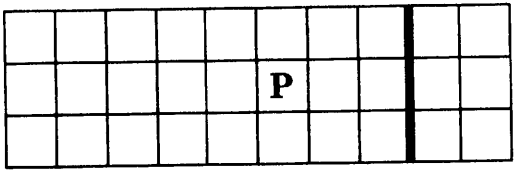
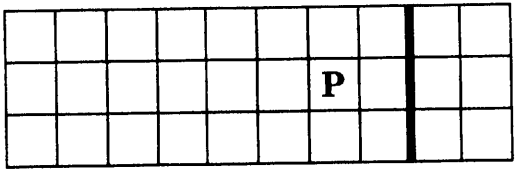
нц пока справа свободно
вправо
кц

Посмотрим, верно ли она работает. Ведь мы не знаем, на каком расстоянии справа от Робота находится стена.

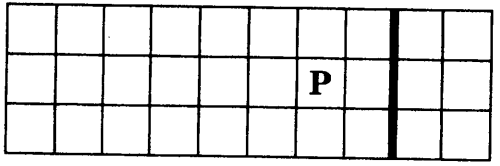
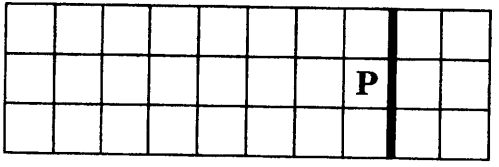
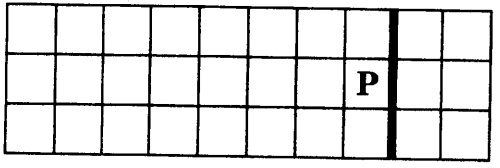
Проверим её для начала на приведённом примере (см. таблицу на стр. 156-158).

Таблица к примеру

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Да	Начинаем цикл **пока**. Сразу проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла
	**вправо**		В результате команды **вправо** Робот передвинулся на соседнюю клетку. Тело цикла закончилось.
	**нц пока справа свободно**	справа свободно? Да	Возвращаемся на строку проверки условия продолжения цикла. Оно снова выполнилось. Снова выполняем тело цикла.

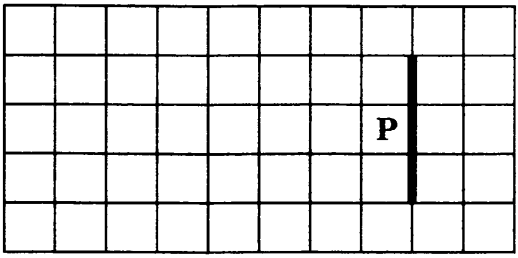
Положение Робота	Команда	Проверка условия	Пояснение
	**вправо**		Робот передвинулся ещё правее.
	**нц пока справа свободно**	справа свободно? Да	Справа, по-прежнему, свободно. Можно выполнять цикл ещё раз.
	**вправо**		Робот передвинулся ещё правее.

Окончание таблицы

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Да	Справа, по-прежнему, свободно. Можно выполнять цикл ещё раз.
	**вправо**		Робот передвинулся ещё правее.
	**нц пока справа свободно**	справа свободно? Нет	Справа от Робота не свободно (стена). Цикл **пока** заканчивается. Робот достиг цели.

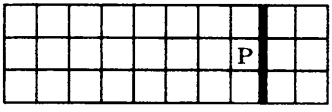
Очевидно, что рассмотренный нами пример не уменьшает общности при разном расстоянии от Робота до стены. Можно предполагать, что при другом расстоянии от Робота до стены он так же будет корректно работать.

Однако, есть одно начальное положение Робота, которое необходимо обязательно проверить, так как оно существенно отличается от остальных. В этом начальном положении Робот сразу стоит у стены:



По условию расстояние от Робота до стены неизвестно и может быть любым, например равным нулю. Этот случай надо обязательно проверить. Несмотря на то, что Робот уже находится в требуемой позиции, он всё равно должен будет выполнить ту программу, которую мы для него написали. Следовательно, нужно убедиться, что в результате её выполнения Робот не разрушится, программа завершится, и Робот после этого останется стоять в требуемой позиции.

Проверяем:

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Нет	Справа от Робота не свободно (стена). Цикл пока заканчивается. Робот достиг цели.

В рассмотренном нами случае приведённое решение правильно. Программа заканчивает свою работу (не зацикливается — не входит в цикл, который никогда не заканчивается), Робот не разрушается, и оказывается в требуемом месте.

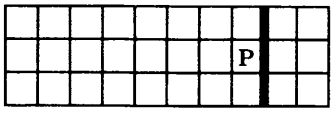
Рассмотрим примеры неверного решения той же задачи:

Неверный пример 1 решения подзадачи 1

вправо
нц пока справа свободно
вправо
кц

На первый взгляд, это совершенно такое же решение. Единственное отличие — перед циклом Робот делает один шаг вправо. Ничего ведь страшного! (?) Роботу нужно сделать довольно-таки много шагов вправо, чтобы достичь стены.

Практически для всех случаев начального положения Робота это так и есть. Программа прекрасно работает. Однако, в одном случае это не так:

Положение Робота	Команда	Проверка условия	Пояснение
	**вправо**		Робот сразу находится возле стены. Первая же команда программы требует его передвинуться на одну клетку вправо. А там — стена. Робот разрушается.

Неверный пример 2 решения подзадачи 1

нц пока справа свободно
вправо
кц
нц пока не справа свободно
влево
вправо
кц

Первая часть программы совпадает с правильным решением, рассмотренным нами выше.

В результате первого цикла Робот окажется возле стены в требуемой точке. Однако (из каких-то странных соображений) автор этой программы решил написать ещё один фрагмент.

К началу выполнения второго цикла Робот находится у стены. Поэтому второй цикл начинает выполняться. В нём Робот делает один шаг влево, после чего сразу делает один шаг вправо, возвращаясь в ту же клетку, где начался второй цикл. Снова проверяется условие второго цикла. Оно опять выполняется. Снова Робот делает шаги влево и вправо. И так бесконечно.

Возможно, данный пример покажется вам слишком надуманным. Но, к сожалению, учащиеся регулярно совершают странные ошибки.

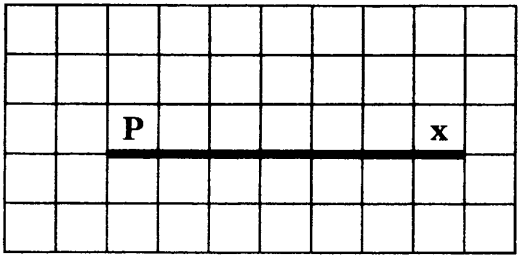
Этим примером мы хотели, с одной стороны, представить пример программы, которая никогда не заканчивается. С другой, обратить ваше внимание на то, что необходимо каждый раз проверять написанную программу на правильность. Наиболее надежный для способ проверки — выполнить трассировку для нескольких начальных положений, не забыв при этом рассмотреть граничные (вырожденные) примеры. Например, с нулевым начальным расстоянием до стены.

Программа считается верной только тогда, когда она успешно выполняется для любых начальных положений Робота. Если есть хотя бы одно положение, в котором это не так (Робот разрушается или программа не заканчивается из-за бесконечного цикла), вся программа считается целиком неверной.

Подзадача 2.

Робот находится возле стены. Длина стены неизвестна. Он должен доехать до края стены (в данном примере — до правого края). Требуемая позиция отмечена знаком «x».

Пример



Решение подзадачи 2

В этот раз ничего не мешает Роботу двигаться вправо. Однако, движение необходимо будет прекратить, когда Робот доберётся до края стены.

Заметим, что написать программу, в которой Робот будет двигаться вдоль стены вправо и остановится ровно в клетке «x» — нельзя. Потому что с точки зрения условий, которые умеет проверять Робот, клетка «x» ничем не отличается от любой другой клетки возле стены (сверху от стены).

Действительно, для любой клетки над стеной проверка любого из трёх условий **слева свободно, справа свободно, сверху свободно** — ложь, а проверка условия **снизу свободно** — истина. Единственным отличи-

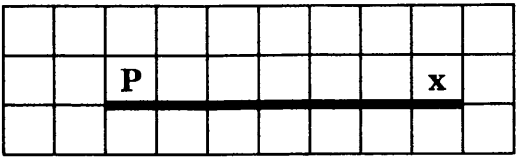
ем клетки «х» от остальных клеток над стеной является клетка, которая расположена справа от клетки «х». Для неё условие **снизу свободно** — ложно.

То есть, чтобы отправить Робота к клетке «х», нужно сделать так — двигать Робота вправо до тех пор, пока снизу от Робота есть стена. В результате Робот остановится в клетке, расположенной справа от клетки «х». Остаётся только вернуть Робота на одну клетку назад, т. е. вправо.

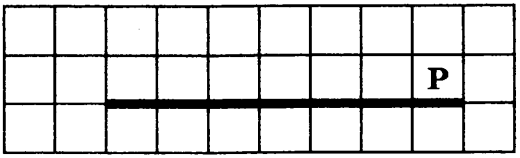
Получаем программу решения этой подзадачи:

нц пока не снизу свободно
вправо
кц
влево

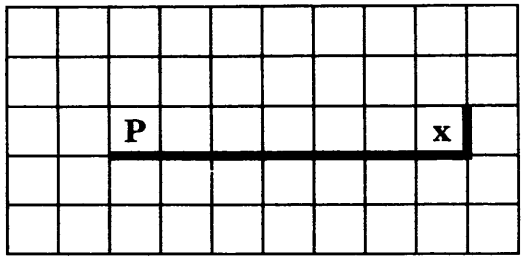
Проверьте сами, что эта программа верно работает и не ломается в обоих вариантах, для которых необходима проверка. То есть в приведённом варианте начального расположения Робота:



И в варианте, когда Робот уже находится в нужной клетке:



Заметим, что приведённая программа не будет правильно работать в ситуации, когда задача немного другая. Например:

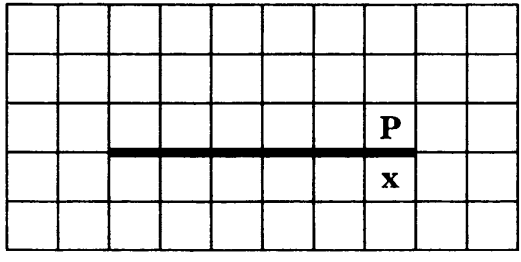


Если попытаться запустить приведённую программу решения подзадачи 2 для данного поля со стенами, Робот разрушится при попытке пройти сквозь стену из клетки «х» вправо.

Однако, это не должно смущать, так как задача является не разновидностью подзадачи 2, а разновидностью подзадачи 1. Здесь вместо того, чтобы проверять наличие стены снизу, достаточно проверять отсутствие стены справа.

Подзадача 3.

Робот находится непосредственно возле края стены (в данном примере — возле правого края). Он должен оказаться по другую сторону от стены, возле того же края, но с другой стороны. Требуемая позиция отмечена знаком «х». *Пример*:



Решение подзадачи 3

В данной подзадаче Роботу не требуется двигаться в какую-либо сторону неизвестное количество дей-

ствий. Ему достаточно обойти стену. Это делается за ограниченное количество действий-команд, не требующих повторяющихся действий. То есть, цикл здесь не требуется.

Решение задачи простое:

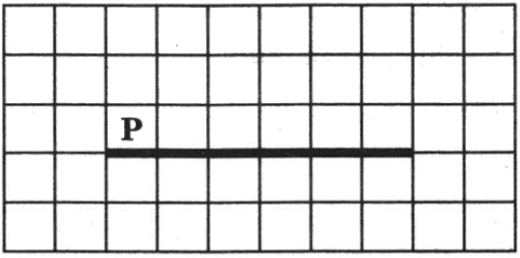
вправо
вниз
влево

Подзадача 4.

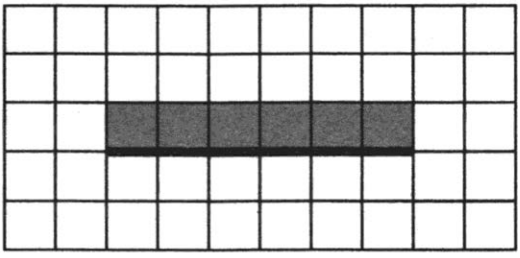
Робот находится непосредственно возле стены (в данном примере — возле левого края). Он должен закрасить все поля от своего текущего положения вплоть до края стены (в данном примере — вплоть до правого края стены).

Пример:

Исходное положение:



Требуется закрасить:



Решение подзадачи 4

С точки зрения движений, Робот должен сделать всё то же самое, что и при решении подзадачи 2 — двигаться вправо, пока снизу от него есть стена.

Возьмём за основу программу решения подзадачи 2:

нц пока не снизу свободно
вправо
кц
влево

Чтобы при этом клетки, через которые проходит Робот, оказались закрашенными, в программу следует добавить команду **закрасить**.

Так как закрасить нужно много клеток, команду **закрасить** будем выполнять много раз. Для этого лучше использовать цикл. Удобнее всего (если получится) совместить цикл закрашивания с циклом перемещения Робота до правого края стены.

Вот решение этой подзадачи:

нц пока не снизу свободно
закрасить
вправо
кц

Заметим, мы добавили команду **закрасить** в тело цикла, чтобы клетки, находящиеся над стеной, которые проезжает Робот, оказывались закрашенными.

Также мы убрали команду **влево**, стоявшую в решении подзадачи 2 после конца цикла, потому что в этой подзадаче неважно, в какой клетке окажется Робот.

В нашем случае Робот окажется в клетке, расположенной сразу справа от самой правой закрашенной клетки. Это хорошо понимать. Однако, по заданию нет необходимости возвращать Робота в какую-либо клетку.

Выполните самостоятельно трассировку этой программы для двух типовых случаев (стена длины 4 и стена длины 1) и убедитесь, что для них программа верно решает данную подзадачу.

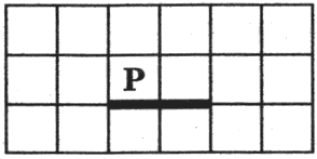
Обратите внимание на то место программы, где располагается команда **закрасить**! Мы поставили её в цикл **перед** командой **вправо**. Хотя, на первый взгляд, нет никакого принципиального отличия от решения, когда команда **закрасить** стоит **после** команды **вправо**.

Рассмотрим это решение:

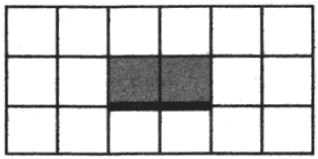
нц пока не снизу свободно
вправо
закрасить
кц

Попытаемся проверить его на таком простом примере.

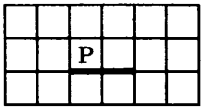
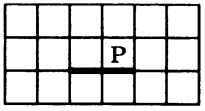
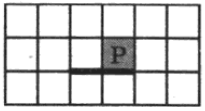
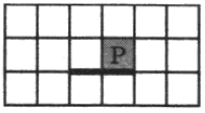
Исходное положение:

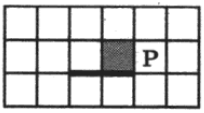
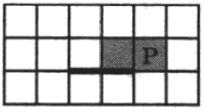


Требуется закрасить:



Выполним эту программу по шагам (см. таблицу на стр. 168–169).

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока не снизу свободно**	не снизу свобод-но? Да	Начинаем цикл пока. Сразу проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла
	**вправо**		Робот двигается на одну клетку вправо. Уже можно видеть, что клетка, в которой Робот находился изначально, осталась незакрашенной
	**закрасить**		Закрашиваем клетку, где стоит Робот. Заканчивается тело цикла. Снова переходим к проверке условия продолжения цикла
	**нц пока не снизу свободно**	не снизу свобод-но? Да	Проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла

Положение Робота	Команда	Проверка условия	Пояснение
	**вправо**		Робот двигается на одну клетку вправо
	**закрасить**		Закрашиваем клетку, где стоит Робот. Видим, что Робот закрасил лишнюю клетку

Трассировка программы показала, что перестановка команд в теле цикла даёт сразу две ошибки — остаётся незакрашенной клетка, где Робот начинает движение, и закрашивается лишняя клетка, в которой Робот оказывается после цикла.

Предположим, мы заметили, что это решение оставляет незакрашенной начальную клетку.

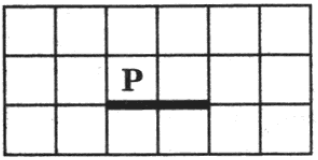
Решаем исправить эту ошибку, добавив в программу «заплатку» — закрасив начальную клетку до начала цикла:

закрасить
нц пока не снизу свободно
вправо
закрасить
кц

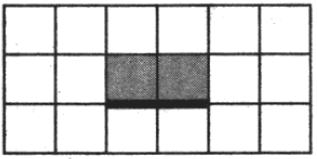
Данная программа, к сожалению, также работает неправильно.

Выполним трассировку для простого примера со стеной длинной в две клетки.

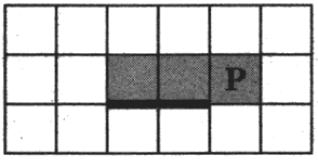
Исходное положение:



Требуется закрасить:



Получим результат:



Все нужные клетки закрашены. Но ещё закрашена одна лишняя. Добавим в программу вторую «заплатку» — не закрашивать клетку, если снизу от Робота нет стены:

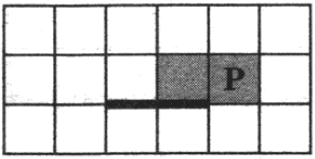
закрасить
нц пока не снизу свободно
вправо
если не снизу свободно то
закрасить
все
кц

Данная программа решает подзадачу 4 правильно. Но, как видите, она существенно длиннее и сложнее для понимания.

Как же можно было в данном случае получить красивое решение, без заплаток, пытаясь исправить программу:

нц пока не снизу свободно
вправо
закрасить
кц

Выполнив программу на примере и обнаружив вот такой результат её работы:



хорошо заметить, что программа закрашивает правильное количество клеток, но совершает неверное действие в начале работы (оставляет незакрашенной начальную клетку) и в конце работы (закрашивает лишнюю (конечную) клетку). Это и есть верный признак того, что действия в цикле выполняются в неверном порядке и их нужно поменять местами, например, поставить последнюю команду тела цикла первой или, наоборот, первую команду тела цикла сделать последней. В случае, когда в теле цикла всего две команды, они просто меняются местами.

Как правило, после этого программа работает верно.

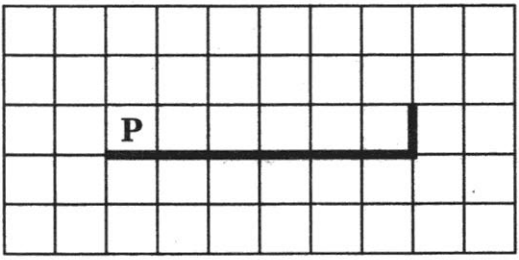
Один из способов сразу определить, какую команду следует поставить первой в теле цикла, выполнить действие с той клеткой, в которой Робот находится к моменту начала цикла. В данном случае Робот сразу был на клетке, которая должна оказаться закрашенной. Значит, первым действием тела цикла требуется красить текущую клетку!

Подзадача 5.

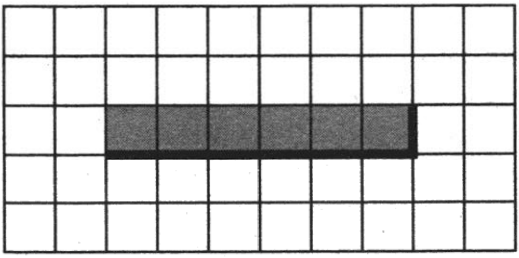
Робот находится непосредственно возле стены (в данном примере — возле левого края). Он должен закрасить все поля от своего текущего положения вплоть до края стены (в данном примере — вплоть до правого края стены).

Пример.

Исходное положение:



Требуется закрасить:



Решение подзадачи 5

Как и в случае с последним примером к решению подзадачи 2, неправильным решением будет совершать цикл, пока снизу от Робота есть стена, потому что в этом случае Робот разрушится при попытке на последнем шаге пройти сквозь стену командой **вправо**.

Условием для остановки Робота должна быть проверка того, что справа от Робота свободно.

Получаем, что за основу решения нужно взять программу из решения подзадачи 1 (двигаться вправо, пока справа свободно), но при этом добавить в неё закрашивание тех клеток, через которые проходит Робот.

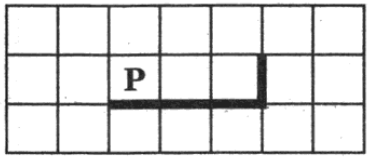
Сделаем это:

нц пока справа свободно
закрасить
вправо
кц

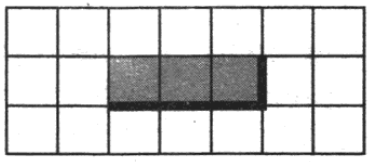
Мы поставили команду **закрасить** перед командой **вправо**, потому что Робот изначально находился в клетке, которая должна оказаться закрашенной, и её нужно, соответственно, закрасить до того, как Робот сдвинется вправо.

Сделаем трассировку данной программы на простом примере.

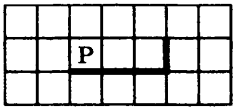
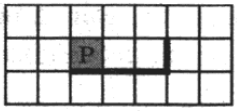
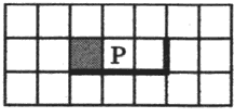
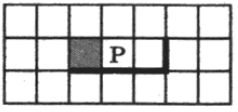
Исходное положение:

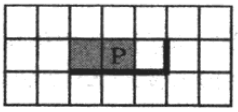
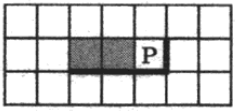
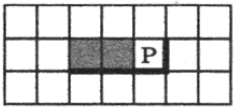


Требуется закрасить:



Выполним программу по шагам (см. таблицу на стр. 174–175):

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Да	Начинаем цикл пока. Сразу проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла
	**закрасить**		Закрасили начальную клетку
	**вправо**		Передвинулись в соседнюю клетку. Тело цикла закончилось. Снова переходим к проверке условия продолжения цикла
	**нц пока справа свободно**	справа свободно? Да	Проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла

Положение Робота	Команда	Проверка условия	Пояснение
	**закрасить**		Закрасили текущую клетку
	**вправо**		Передвинулись в соседнюю клетку. Тело цикла закончилось. Снова переходим к проверке условия продолжения цикла
	**нц пока справа свободно**	справа свободно? Нет	Проверяем условие продолжения. Оно не выполнилось. Заканчивается цикл и программа

Программа закончилась, но одна клетка, которую нужно было закрасить, осталась незакрашенной. Нужно ли переписывать программу или достаточно просто исправить одну ошибку, закрасив последнюю клетку? Сравним доводы в ту или иную сторону.

Клетка, которая осталась незакрашенной — это именно та клетка, где находится Робот после окончания работы программы. То есть, если в конец приведённой программы, после цикла, дописать команду **закрасить**, программа, возможно, станет работать верно.

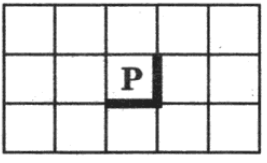
Не нужно ли переписать программу, например, поменяв местами команды в теле цикла, как это было рассмотрено при обсуждении решения подзадачи 4? В данном случае, доводов в эту сторону нет, потому что количество клеток, которые оказались закрашенными в результате работы цикла, неверное. Не хватает закрасить ещё одну клетку. Значит, поменяв команды в теле цикла, закрасится столько же клеток, но останется незакрашенной начальная клетка. Так сделать тоже можно, но «заплатку» — закрашивание клетки вне цикла — всё равно придётся делать. Только в этом случае до цикла, а не после.

В итоге, правильное решение будет такое:

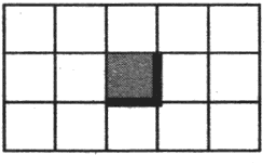
нц пока справа свободно
закрасить
вправо
кц
закрасить

Обязательно нужно проверить это решение для граничного случая, когда Робот уже стоит возле правой стены.

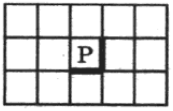
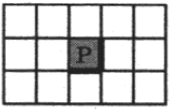
Исходное положение:



Требуется закрасить:



Выполним трассировку:

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Нет	Начинаем цикл пока. Сразу проверяем условие продолжения. Оно не выполнилось. Тело цикла не выполняется. Переходим к команде после цикла
	**закрасить**		Закрашивается текущая клетка. Программа заканчивается

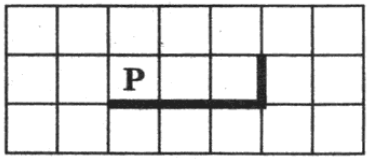
Программа чудесно справилась с поставленной задачей. Робот не разрушился. Программа закончилась. Все нужные клетки закрашены. Лишние — не закрашены. Успех!

Теперь рассмотрим неудачный пример решения этой подзадачи:

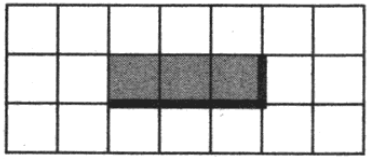
нц пока справа свободно
закрасить
вправо
закрасить
кц

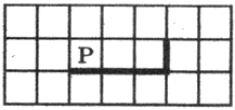
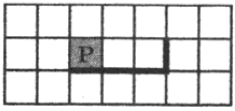
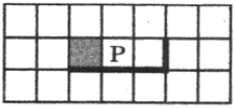
Пример на первый взгляд работает.
Выполним трассировку на простом примере.

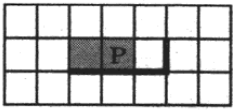
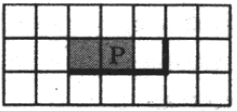
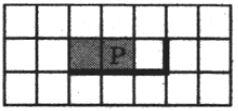
Исходное положение:

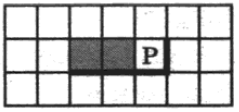
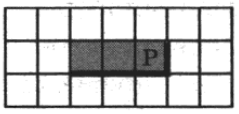
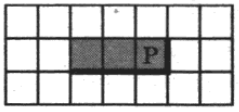


Требуется закрасить:



Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Да	Начинаем цикл пока. Сразу проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла
	**закрасить**		Закрасили начальную клетку
	**вправо**		Передвинулись в соседнюю клетку

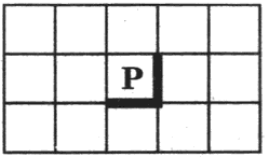
Положение Робота	Команда	Проверка условия	Пояснение
	**закрасить**		Закрасили текущую клетку. Тело цикло закончилось. Снова переходим к проверке условия продолжения цикла
	**нц пока справа свободно**	справа свободно? Да	Проверяем условие продолжения. Оно выполнилось, значит, выполняем тело цикла
	**закрасить**		Команда закрасить текущую клетку. Клетка уже закрашена. Однако по условию нет ограничений на количество способов красить клетку. Почему бы и нет!? Ничего плохого не произошло. Закрасили текущую клетку

Положение Робота	Команда	Проверка условия	Пояснение
	**вправо**		Передвинулись в соседнюю клетку
	**закрасить**		Закрасили текущую клетку. Тело цикла закончилось. Снова переходим к проверке условия продолжения цикла
	**нц пока справа свободно**	справа свободно? Нет	Проверяем условие продолжения. Оно не выполнилось. Заканчивается цикл и программа

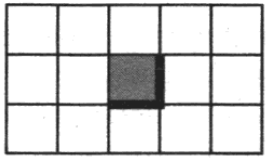
Все требуемые клетки закрашены. Вроде бы, всё чудесно.

Проверим, однако, работу этой программы на граничном примере — когда Робот сразу находится у правой стены.

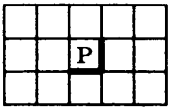
Исходное положение:



Требуется закрасить:



Выполним трассировку:

Положение Робота	Команда	Проверка условия	Пояснение
	**нц пока справа свободно**	справа свободно? Нет	Начинаем цикл пока. Сразу проверяем условие продолжения. Оно не выполнилось. Тело цикла не выполняется. После цикла команд нет. Программа заканчивается

Неуспех! Единственная клетка, которую нужно было закрасить, осталась незакрашенной.

Это случилось оттого, что автор данной программы для исправления ошибки в одной клетке поставил команду-«заплатку» в цикл! Тем самым нарушил одну из рекомендаций по программированию циклов:

1. Повторяющиеся действия помещаем в цикл!

Это, кстати, все и так понимают. Если в цикл не поместить повторяющиеся действия, то эти действие выполнятся только один раз.

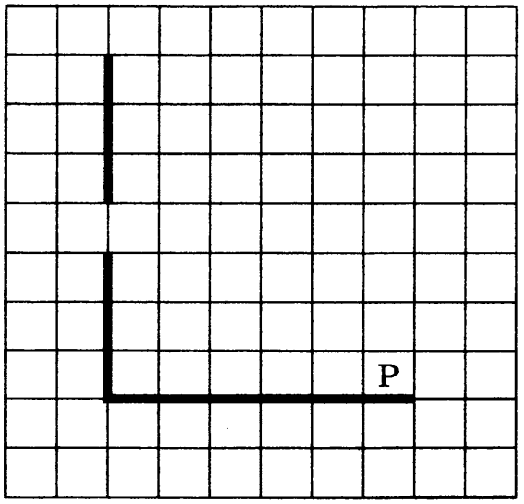
2. Действия, которые нужно выполнить только один раз, не нужно помещать в цикл!

А вот этим правилом часто пренебрегают, к сожалению, даже опытные программисты. Его невыполнение не всегда приводит к ошибкам программы, но почти всегда — к плохо понятному коду программы. Что, в свою очередь, затрудняет поиск ошибок.

Из подзадач, решение которых мы рассмотрели к настоящему моменту, можно сложить решение практически любой задачи Робота.

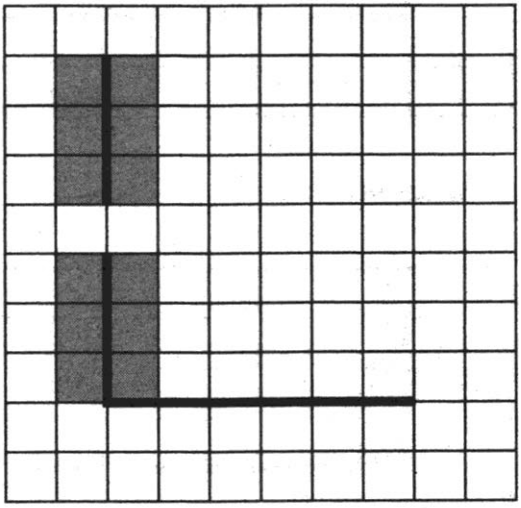
Рассмотрим решение задачи, приведённой в начале раздела: на бесконечном поле есть горизонтальная и вертикальная стены. Левый конец горизонтальной стены соединён с верхним концом вертикальной стены. Длины стен неизвестны. В вертикальной стене есть ровно один проход, точное место прохода и его ширина неизвестны. Робот находится в клетке, расположенной непосредственно над горизонтальной стеной у её правого конца.

На рисунке указан один из возможных способов расположения стен и Робота (Робот обозначен буквой «Р»).



Напишите для Робота алгоритм, закрашивающий все клетки, расположённые непосредственно левее и правее вертикальной стены.

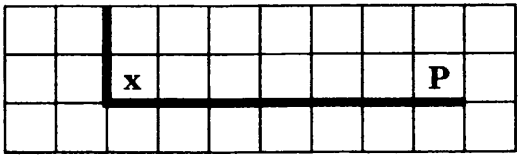
Робот должен закрасить только клетки, удовлетворяющие данному условию. Например, для приведённого выше рисунка Роботу необходимо закрасить следующие клетки:



Решение

Разобьём решение данной задачи на несколько подзадач:

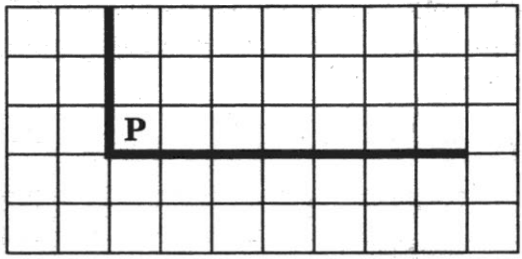
1. Переместить Робота в угол, в ту клетку, у которой есть стена слева и снизу:



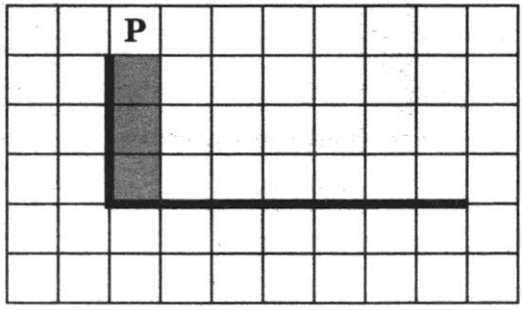
Как вы понимаете, это подзадача 1 — передвигать Робота, пока он не упрётся в стену.

2. Закрасить участок справа от нижней части вертикальной стены.

Из положения:



перейдём к положению:



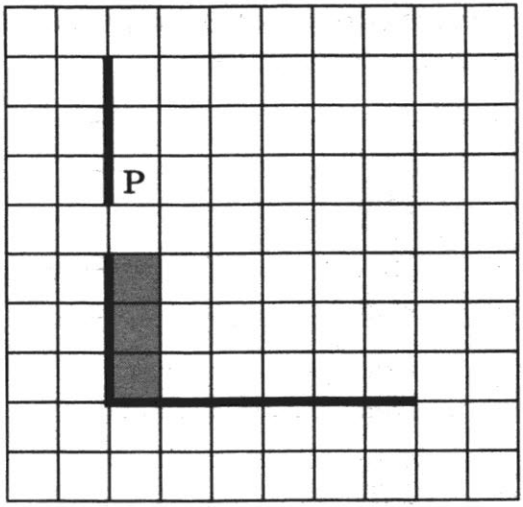
Это подзадача 4 — закрасить клетки справа от стены.

3. Переведём Робота к следующему участку стены, чтобы справа от верхней части вертикальной стены повторить подзадачу 4.

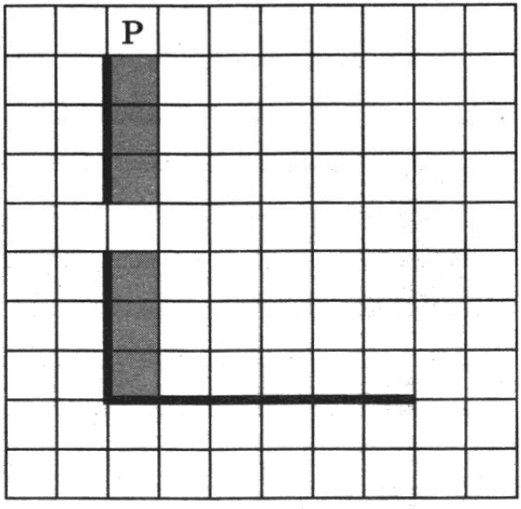
Для этого нужно передвигать Робота на одну клетку вверх, пока слева от него свободно.

4. Закрасить участок справа от верхней части вертикальной стены.

Из положения:



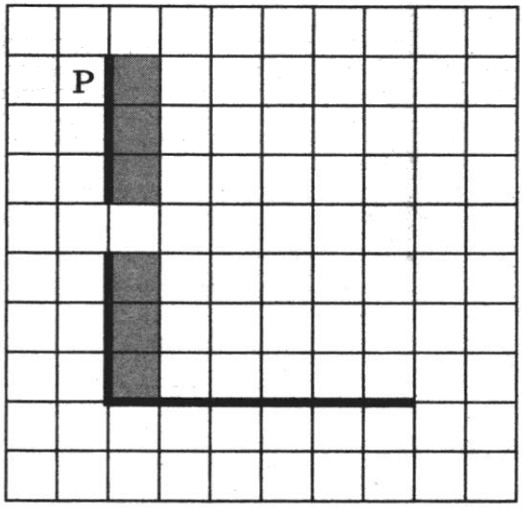
перейдём к положению:



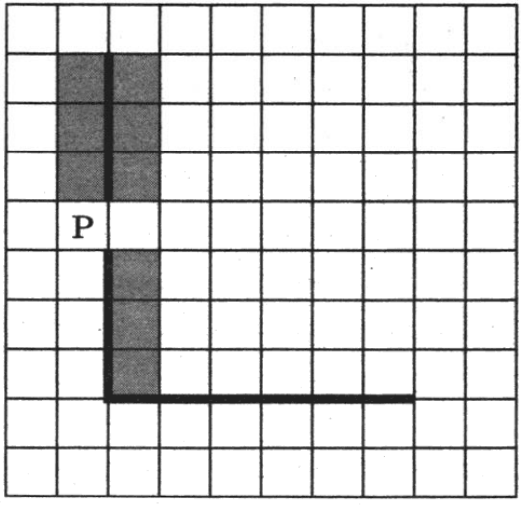
5. Передвинем Робота к следующему участку стены, чтобы слева от верхнего участка вертикальной стены повторить подзадачу 4.

Для этого достаточно передвинуть Робота влево и вниз.

6. Закрасить участок слева от верхней части вертикальной стены. Из положения:



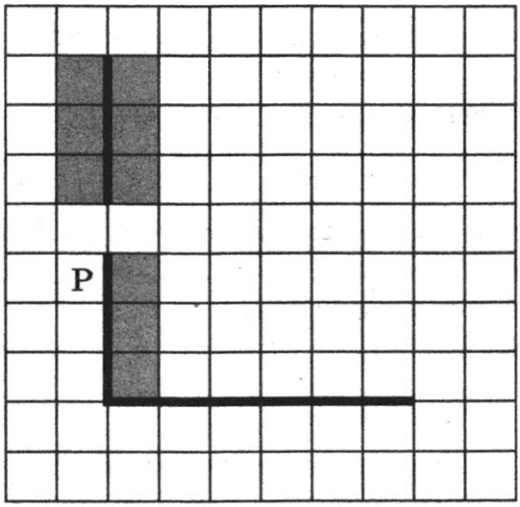
перейдём к положению:



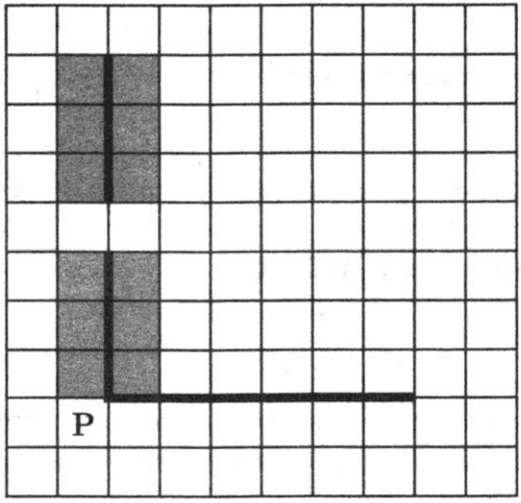
7. Переведём Робота к следующему участку стены, чтобы слева от нижней части вертикальной стены повторить подзадачу 4.

Для этого нужно передвигать Робота вниз, пока справа от него свободно.

8. Закрасить участок слева от нижней части вертикальной стены. Из положения:



перейдём к положению:



Осталось только вместо каждого из приведённых пунктов поставить друг за другом соответствующие

решения подзадач, не забывая менять в них условия и передвижения, в зависимости от положения стены относительно Робота и того, в какую сторону Робот должен двигаться.

Получаем:

| 1. Переместить Робота в угол, в ту клетку, у которой | есть стена слева и снизу:

нц пока слева свободно
влево
кц

| 2. Закрасить участок справа от нижней части | вертикальной стены.

нц пока не слева свободно
закрасить
вверх
кц

| 3. Переведём Робота к следующему участку стены, | чтобы справа от верхней части вертикальной стены | повторить подзадачу 4.

| Будем передвигать Робота вверх, пока слева от него | свободно

нц пока слева свободно
вверх
кц

| 4. Закрасить участок справа от верхней части | вертикальной стены.

нц пока не слева свободно
закрасить
вверх
кц

| 5. Передвинем Робота к следующему участку стены, | чтобы слева от верхнего участка вертикальной стены | повторить подзадачу 4.

влево
вниз

| 6. Закрасить участок слева от верхней части
| вертикальной стены.

нц пока не справа свободно
закрасить
вниз
кц

| 7. Переведём Робота к следующему участку стены,
| чтобы слева от нижней части вертикальной стены
| повторить подзадачу 4.
| Будем передвигать Робота вниз, пока справа от нег
| свободно

нц пока справа свободно
вниз
кц

| 8. Закрасить участок слева от нижней части
| вертикальной стены.

нц пока не справа свободно
закрасить
вниз
кц

ИНФОРМАЦИОННЫЕ И КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ

4 Основные устройства, используемые в ИКТ

Файловая система компьютера

Конспект

Для долговременного хранения данных на компьютере используется так называемая **внешняя память.** Это специальные устройства хранения информации, на которые можно записывать информацию, достаточно долгое время хранить и, при необходимости, считывать. При этом для хранения не требуется питания. Такие устройства, как правило, достаточно большого объёма и могут работать по разным технологиям. В операционных системах MS Windows устройства для работы с внешней памятью представлены в виде так называемых **дисков.** Данное название исторически произошло от традиционных устройств внешней памяти во время появления Windows. Сейчас физически это может быть организовано как диск или просто микросхема, но называется так же, как и раньше.

Каждый диск, который планируется использовать в Windows, должен иметь имя **(имя диска).** Имя диска состоит из буквы латинского алфавита и двоеточия после него. Например, **C:**, **E:**, **H:**. Традиционно основной диск компьютера называется диском **C:**, именно на нём находится операционная система Windows. Буквы **A:** и **B:**, опять же традиционно, выделены для устройств чтения гибких дисков (дискет), которые на современные компьютеры уже не устанавливают. Несмотря на это, имена принято начинать с буквы **C:**. В зависимости от количества других подключённых устройств

(устройств чтения оптических дисков, USB-флешек и прочего) в операционной системе вы можете видеть ещё некоторое количество букв-дисков.

Любая информация, которая хранится на диске, должна быть организована в виде **файла**. Файл — это область информации, хранящаяся на внешней памяти (на диске) и имеющая **имя**.

В операционной системе Windows имя файла состоит из не более чем 255 символов. В имени не допустимы некоторые символы (: / | \ < > « * ?).

Часть имени файла после последней точки называется **расширением имени файла** и обычно состоит из трёх символов, по которым операционная система считает, что содержимое файла записано в определённом формате. Например, файл с расширением .jpg содержит картинку, а файл с расширением .txt — текст.

На диске содержится много файлов (зачастую, несколько десятков тысяч). Чтобы все эти файлы не были «свалены в кучу» и с ними можно было удобно работать, применяют специальную технологию, которая называется **иерархическая структура файловой системы**. На диске создают разделы: **каталоги** или **папки**. В каждой папке можно снова создать какое-то количество папок. Они будут называться **вложенными папками** по отношению к той папке, в которой они находятся. Файлы могут находиться как просто на диске, так и внутри любой папки диска.

Место диска, которое не находится ни в какой папке и в котором создаются главные папки диска, называется **корнем диска** и обозначается символом \. У каждого диска имеется свой собственный корень диска. Например, корень диска C: обозначается как C:\.

Для обозначения того, в каком именно месте диска находится конкретный файл, применяют так называемое **полное имя файла**. Оно состоит из трёх частей —

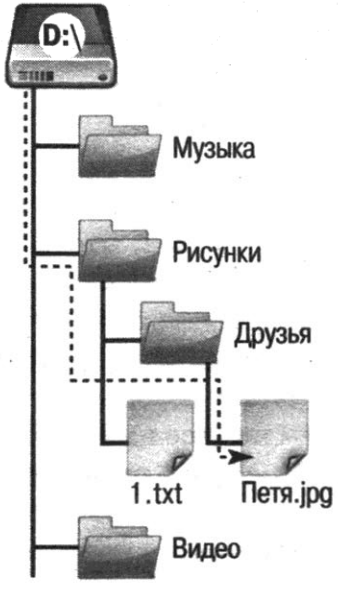
имени диска, на котором находится файл, **пути к файлу** на диске и собственно имени файла.

Так, например, файл **Вася.txt**, который хранится непосредственно в корне диска **C:**, имеет полное имя: **C:\Вася.txt**.

Другой пример: файл **Петя.jpg** лежит в папке **Друзья**, которая, в свою очередь, располагается в папке **Рисунки**. Эта папка находится, в свою очередь, в корне диска **D:**. Тогда полное имя этого файла будет:

D:\Рисунки\Друзья\Петя.jpg.

То есть, между именем диска и собственно именем файла перечисляется последовательность папок, в которые нужно «войти» от корня диска, для того, чтобы «добраться» до указанного файла. Эти имена папок разделяются символами \ **(бэкслеш)**. Последовательность этих папок, разделённых символами \, называется **путём к файлу на диске**. Обычно структуру папок и файлов на диске изображают в виде дерева.



Обратим ещё раз внимание на полное имя файла. Например, **D:\Рисунки\Друзья\Петя.jpg**. Анализируя это полное имя можно сделать вывод, что папка (каталог) **Друзья** является подпапкой (подкаталогом) папки **Рисунки**, папка **Рисунки** расположена непосредственно в корне диска **D:**, а файл **Петя.jpg** расположен в папке **Друзья**.

Разбор типовых задач

Задача 1. В некотором каталоге хранился файл **Хризантема.doc**, имевший полное имя **D:\2013\Осень\Хризантема.doc**.

В этом каталоге создали подкаталог **Ноябрь** и файл **Хризантема.doc** переместили в созданный подкаталог.

Укажите полное имя этого файла после перемещения.

1) D:\2013\Осень\Ноябрь\Хризантема.doc
2) D:\Ноябрь\Хризантема.doc
3) D:\2013\Осень\Хризантема.doc
4) D:\2013\Ноябрь\Хризантема.doc

Решение

Анализируя полное имя файла: **D:\2013\Осень\Хризантема.doc**, делаем вывод, что файл **Хризантема.doc** хранится в папке **Осень**. По условию задачи в этой папке создали подкаталог **Ноябрь**. То есть, полное имя этого подкаталога будет: **D:\2013\Осень\Ноябрь**. Файл **Хризантема.doc** переместили в этот подкаталог. Значит, получаем полное имя файла: **D:\2013\Осень\Ноябрь\Хризантема.doc**.

Ищем указанное полное имя файла в списке вариантов ответа и находим его под номером 1.

Ответ: 1.

Задача 2. Пользователь работал с каталогом **Апельсины**. Сначала он поднялся на один уровень вверх, затем спустился на один уровень вниз, потом ещё раз спустился на один уровень вниз.

В результате он оказался в каталоге **E:\Плоды\Фрукты\Цитрусовые**.

Запишите полный путь каталога, с которым пользователь начинал работу.

1) **E:\Апельсины**
2) **E:\Плоды\Апельсины**
3) **E:\Плоды\Фрукты\Апельсины**
4) **E:\Плоды\Фрукты\Цитрусовые\Апельсины**

Решение

Постараемся выполнить последовательность перемещений пользователя по структуре папок в обратную сторону.

В результате перемещений по каталогам пользователь оказался в каталоге **E:\Плоды\Фрукты\Цитрусовые**.

Последним действием пользователь спустился на один уровень вниз. То есть, перешёл из каталога, в котором он находился, в подкаталог. Значит, перед этим действием он находился одним уровнем выше, в подкаталоге **E:\Плоды\Фрукты**.

Предпоследним действием пользователь тоже спустился на один уровень вниз. Следовательно, перед этим он находился ещё одним уровнем выше (ближе к корню диска). Значит, перед этим действием пользователь был в каталоге **E:\Плоды**. Предыдущим (первым) своим действием пользователь поднялся на один уровень вверх и оказался при этом, как мы только что выяснили, в каталоге **E:\Плоды**. Следовательно, перед этим пользователь находился в каком-то подкаталоге каталога **E:\Плоды**. Но по условию пользователь изна-

чально работал с каталогом **Апельсины**. Значит, именно в этом подкаталоге каталога **E:\Плоды** он изначально и был. Получаем полное имя начального каталога: **E:\Плоды\Апельсины**. Ищем это имя среди вариантов ответа и находим его под номером 2.

Ответ: 2.

Задача 3. Для групповых операций с файлами используются маски имён файлов. Маска представляет собой последовательность букв, цифр и прочих допустимых в именах файлов символов, в которых также могут встречаться следующие символы:

Символ «?» (вопросительный знак) означает ровно один произвольный символ.

Символ «*» (звёздочка) означает любую последовательность символов произвольной длины, в том числе «*» может задавать и пустую последовательность.

Определите, какое из указанных имён файлов удовлетворяет маске:

?ba*r.?xt

1) bar.txt 2) obar.txt 3) obanar.xt 4) barr.txt

Решение

Проанализируем маску имени файла.

Первый символ «?», с которого начинается маска имени файла и символы «ba» после него, означают, что нужное имя файла должно начинаться с ровно одного любого символа, сразу после которого обязательно должны стоять символы «ba». То есть, символы «b» и «a» стоят на втором и третьем месте имени файла.

Первый и четвёртый варианты ответа начинаются с символов «ba». Значит, эти варианты нам не подходят (ведь перед символами «ba» в имени файла должен стоять ещё один символ, а в этих вариантах его нет).

Продолжаем анализ маски. В первой части маски, до точки, стоит ещё символы «*» (сразу после символов «ba») и «r». Значит, в имени файла после символов «ba», стоящих на втором и третьем месте, может стоять ещё любое количество символов, но обязательно непосредственно перед точкой должен быть символ «r». В обоих оставшихся вариантах имени файла именно так и есть. Во втором варианте ответа на месте «*» нет символов вовсе, а в третьем варианте ответа на месте «*» стоят символы «na». Оба этих варианта подходят под маску перед точкой.

Анализируем последнюю часть маски, после точки. Там написан символ «?», после которого следуют символы «xt». Это означаетт, что в последней части имени файла, после точки, должно стоять ровно 3 символа. Первый из них может быть любым, а оставшиеся два должны быть символами «xt». Под это описание подходит второй вариант ответа (на месте символа «?» стоит символ «t»), а третий вариант — не подходит (после точки должно быть 3 символа, а в третьем варианте ответа их только два).

Ответ: 2.

5 Запись средствами ИКТ информации об объектах и о процессах окружающего мира

Кодирование информации. Принцип двоичного кодирования. Кодирование текстовой, графической, звуковой информации

Конспект

При хранении и обработке информации на компьютере используется **двоичная система счисления** (подробнее о двоичной системе счисления см. стр. 17). Это значит, что любая обрабатываемая компьютером информация кодируется при помощи двоичных кодов. Рассмотрим, как это происходит для различных типов информации.

Целые неотрицательные числа переводятся в двоичную систему счисления и именно в этом виде записываются. Следует только понимать, что любая информация на компьютере записывается в ячейки определённого размера. Размер ячейки составляет некоторое количество байт (напомним, байт = 8 бит). Чаще всего используют размеры в 1, 2, 4 и 8 байт (как видите, в компьютере даже размеры ячеек принято делать равными какой-нибудь степени числа 2). То есть, целое неотрицательное число хранится в ячейке из некоторого количества бит (обычно 8, 16, 32, 64 бита

соответственно). Это значит, что хранимые на компьютере числа не могут быть любыми, произвольного размера. Размер максимального хранимого числа зависит от размера ячейки, в которой это число хранится. В ячейке размером n бит можно хранить максимальное неотрицательное число — $2^n - 1$.

Например, в ячейке размером 8 бит максимальное неотрицательное число, которое можно сохранить — это 255 ($= 2^8 - 1$).

Несколько сложнее хранятся числа со знаком. Из всех бит, которые выделяются для хранения числа, один бит выделяется для хранения знака числа (0 — число положительное, 1 — число отрицательное). Таким образом, для хранения значения числа остаётся на 1 бит меньше. То есть, наибольшее знаковое число, которое хранится в ячейке размером n, — это $2^{n-1} - 1$.

Например, в ячейке размером 16 бит можно хранить числа до 32767 ($= 2^{16-1} - 1$).

Для хранения вещественных чисел используется более хитрая технология. Изучение этой темы не входит в курс основной школы.

При кодировании других видов информации используется примерно такой принцип: придумывается, как пронумеровать все возможные состояния, которые нужно хранить (т. е. в соответствие каждому хранимому состоянию ставится определённый числовой номер-код). Потом коды переводятся в двоичную систему счисления и хранятся в виде двоичных чисел.

Кодирование текстовой информации

Все символы, которые собираются использовать для записи текста, выписываются в ряд и нумеруются последовательными целыми неотрицательными числами, начиная с нуля. Это получило название **кодовой таблицы символов.**

Самая распространённая кодовая таблица символов называется **ASCII (American standard code of Information Interchange — американский стандартный код обмена информации**. Принято читать это как «Аски»).

Изначально данный код был придуман как 7-мибитный (8-й бит использовался для контроля чётности, так как в те времена были очень частые ошибки хранения и передачи информации). Поэтому кодовая таблица содержала в себе только $2^7 = 128$ различных символов. Подобная таблица получила название **базовой таблицы ASCII**. Она содержит в себе только латинские символы (строчные и прописные), цифры, знаки препинания и служебные символы.

Впоследствии методы хранения и передачи информации стали существенно лучше и появилась возможность использовать дополнительный (восьмой) бит, что сразу увеличило количество возможных хранимых знаков вдвое, до 256 ($2^8 = 256$). Эту вторую половину таблицы ASCII стали называть **дополнительной таблицей ASCII**. А всю 8-битную таблицу — **расширенной таблицей ASCII**.

Именно среди 128 дополнительных символов и находятся коды большинства национальных алфавитов, в том числа и кириллица, которой мы пользуемся.

К сожалению, при появлении дополнительной таблицы ASCII разные производители операционных систем не смогли (или не сочли нужным) договориться и придумали свои, различные способы распределения символов кириллицы по 128 кодам дополнительной таблицы. Больше всего «отличилась» фирма Microsoft, которая придумала целых 2 способа этого кодирования.

На данный момент имеется 5 способов хранения кириллицы:

КОИ8 (код обмена информации 8-битный)	Используется в основном в Unix-подобных системах.
MS-DOS	Используется в операционной системе MS DOS.
Windows-1251	Используется в операционных системах MS Windows.
MAC	Используется в операционных системах MAC OS.
ISO	Попытка стандартизировать все предыдущее не получила широкого распространения.

Способ применения дополнительной таблицы задаётся для всего текста. Используя ASCII, невозможно одновременно увидеть текст, например, на русском и на французском языках, потому что оба языка применяют примерно одни и те же коды дополнительной таблицы ASCII. То есть, последовательность кодов следует трактовать либо как русский, либо как французский язык. То же обстоятельство приводит к тому, что иногда на компьютере отображается текст из кириллицы, выглядящий как полная абракадабра. Это связано с тем, что для просмотра текста используется неверная кодовая таблица (либо, что ещё хуже, текст в процессе передачи несколько раз перекодировался из одной кодировки в другую).

Для решения задач на кодирование текста важно знать золотое правило — **независимо от используемой кодировки таблицы ASCII, одному символу соответствует один байт (8 бит).**

Для таблицы ASCII: 1 символ = 1 байт!

Чтобы кардинально решить проблему однозначного кодирования букв всех (практически) алфавитов и стандартизировать кодовые таблицы, в начале 1990-х годов был предложен иной способ кодирования текста — **Unicode (Юникод)**. В нём разработчики решили пожертвовать количеством ради качества — использовать для хранения не один байт, а два, т. е. получить $2^{16} = 65536$ различных символов. Это, с одной стороны, увеличило вдвое объём памяти, необходимый для хранения текстов, а также время, необходимое для их передачи. С другой стороны, убрало все вышеуказанные проблемы и противоречия.

В действительности, Юникод не всегда использует именно 2 байта для хранения кодов символов. Но в рамках основного образования по информатике мы предлагаем считать, что это так. Во всяком случае, в задачах экзамена (ОГЭ), к которому мы надеемся этой книжкой подготовить вас, дорогой читатель, вам встретится именно двухбайтовое кодирование в Юникоде.

Кодирование графической информации

При хранении на компьютере изображений различают два основных способа — **растровый** и **векторный**.

При *растровом* способе изображение хранится в виде прямоугольного массива точек. Эти точки обычно называют **пикселями** (pixel от pictel — picture element — элемент изображения). Каждый пиксель весьма мал, пикселей много, и они находятся близко друг к другу. Человек обычно не различает их по отдельности, а воспринимает всю картинку целиком. Такой способ хранения изображения как правило получается при использовании фотоаппаратов или

сканеров. Подобные устройства разбивают изображения реальных объектов на прямоугольный набор точек-пикселей и запоминают цвет каждого пикселя.

Второй способ хранения изображений — *векторный*. При этом изображение хранится в виде **отдельных объектов** — прямоугольников, эллипсов, линий, кривых разной формы, каждый из которых закрашен определённым образом. Картинка при этом получается менее реалистичной, чем окружающий нас мир, зато такие изображения проще изменять, масштабировать, преобразовывать. Проверка способов кодирования векторных изображений не входит в текущую версию ОГЭ, и мы не будем их далее рассматривать.

Кодирование цвета.

При кодировании изображений важное значение имеет способ кодирования цвета. Самый простой способ — сделать так же, как при кодировании символов. То есть, выписать в столбик все возможные цвета, пронумеровать и хранить эти номера как коды цвета. Данный способ хорош до тех пор, пока количество различных цветов не очень большое. Например, 16 или 256. Такая технология существует и используется, но только для указанного небольшого количества цветов в изображении. Обычно изображение состоит из гораздо большего количества цветов.

Считается, что обычный человеческий глаз способен различить 16 миллионов оттенков цвета. Соответственно, хотелось бы сопоставить всем (или почти всем) этим цветам свои коды. Только хранить таблицу из 16 миллионов оттенков цвета очень неудобно. Поэтому используется другая технология, которая называется цветовая модель. **Цветовая модель** — это способ представления цвета при помощи нескольких базовых цветов. Смешивая оттенки различной ярко-

сти базовых цветов можно получить все возможные цвета, которые способна хранить цветовая модель.

Наиболее известной цветовой моделью является цветовая модель **RGB**. Её название состоит из первых букв трёх базовых цветов этой модели — **R**ed, **G**reen, **B**lue (красный, зелёный, синий). Смешивание именно этих трёх цветов позволяет получить любой из 16,7 миллионов цветов, который умеет хранить цветовая модель RGB.

В этой цветовой модели работают такие окружающие нас электронные устройства, как телевизоры, сканеры, мониторы (в том числе мобильных телефонов и планшетов), проекторы, фотоаппараты. То есть, все устройства, которые получают картинку по принципу светящихся точек. Так, чёрная (изначально) поверхность подсвечивается очень маленькими точками трёх базовых цветов — красного, зелёного и синего. Эти три точки расположены очень близко к друг другу и каждая из них может светиться различным оттенком яркости — от нуля до максимума (255). Именно три точки базовых цветов и образуют те элементы изображения (*пиксели*), из которых состоит растровая картинка.

Так как при восприятии человеческим глазом яркости базовых цветов складываются, то эта цветовая модель называется **яркостной** или, по-другому, **аддитивной** (от английского add — складывать).

Не все компьютерные устройства работают в цветовой модели RGB. Например, принтеры работают в цветовой модели CMYK. Точки базовых цветов наносятся на белую поверхность (например, бумагу) и каждая из них поглощает какое-то количество спектра белого света, который на бумагу падает. Отражённые оттенки спектра мы и видим. Данная модель — субтрактивная (вычитательная, от subtract — вычитать). Цвет вычитается из белого света.

Проверка кодирования цвета в цветовых моделях, отличных от RGB, не входит в ОГЭ по информатике.

Для понимания того, какой цвет получится в цветовой модели RGB при сложении базовых цветов (они называются **компонентами**), нужно понимать **принцип сложения и цветовой шестиугольник**.

Принцип сложения цветов в модели RGB состоит в том, что исходная поверхность — чёрная. То есть, если все три компоненты не будут светиться (будут равны нулю), то получится чёрный цвет.

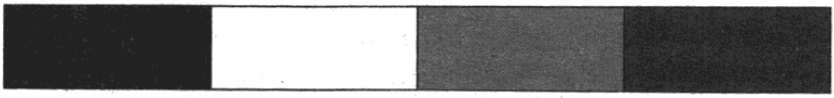
Если же, наоборот, все три компоненты будут светиться по максимуму, то получится максимальная яркость — белый цвет.

Когда все три компоненты будут одинаковыми, получается один из оттенков серого цвета — от чёрного (все три нуля) до белого (все три 255).

Для обозначения кодов цвета принято использовать запись яркостей всех трёх компонент — RGB — в шестнадцатеричной системе счисления двухразрядными числами. Нулевая яркость записывается как 00, средняя яркость — как 80, а максимальная яркость — как FF. Перед кодом цвета принято ставить значок решетки #.

Таким образом, код чёрного цвета будет #000000, код белого цвета — #FFFFFF, код средне-серого цвета — #808080, а код тёмно-серого — #404040.

Например:



чёрный	белый	средне-серый	тёмно-серый
#000000	#FFFFFF	#808080	#404040

Нетрудно понять, что красный цвет — это максимум красной компоненты и остальные 0. То есть,

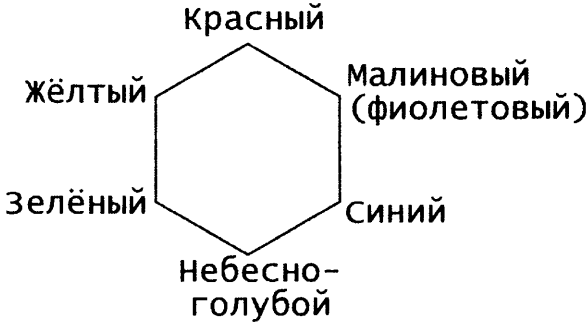
#FF0000. Соответственно, зелёный — #00FF00, а синий — #0000FF.

При необходимости сделать более тёмный цвет, нужно уменьшить общую яркость. Например, тёмно-красный — #800000.

Если требуется сделать более светлый цвет, то нужно, соответственно, увеличить общую яркость.

Однако в случае, например, с красным цветом, яркость красной компоненты уже установлена на максимум. Значит, следует увеличивать яркость других компонент, причём одновременно, т. е. светло-красный будет, например, #FF8080.

Остальные цвета получаются более хитро — смешиванием компонент в разных пропорциях. Для вопросов, которые могут возникнуть на экзамене, достаточно понимать основные цвета на цветовом шестиугольнике:



Из него видно, что смешение красного и синего цвета даст малиновый, а смешение красного и зелёного цвета — жёлтый. Соответственно, для того, чтобы получить цвет, лежащий между указанными, нужно взять базовые цвета в соответствующей пропорции. Так, например, оранжевый цвет лежит посередине между красным и жёлтым. Красный — #FF0000, жёлтый — #FFFF00. Значит, для получения нужного оранжево-

го следует брать среднее арифметическое по каждой компоненте. Получаем, что красного нужно взять (FF + FF)/2 = FF, зелёного (00 + FF)/2 ≈ 80, синего (00 + 00)/2 = 00. То есть, оранжевый — #FF8000.

Исходя из всего вышеизложенного, запомните, что для хранения изображения в цветовой модели RGB используется 3 раза по 8 бит (для каждой компоненты), то есть, всего 3 байта или 24 бита.

Вычисление объёма растрового изображения.

При хранении на компьютере растровых изображений (если не применяются методы сжатия информации) используется следующий принцип: для каждого пикселя хранится его код. Количество пикселей во всем изображении — это количество пикселей по ширине, умноженное на количество пикселей по высоте. Так как все пиксели для своего представления используют одинаковое количество бит, получаем формулу для расчёта объёма растрового изображения:

$$V = H \cdot W \cdot i \,/\, 8,$$

где V — объём неупакованного растрового изображения (в байтах), H и W — ширина и высота растрового изображения (в пикселях), i — количество бит, которое тратится на один пиксель.

Так как объём изображения принято измерять в байтах, а объём памяти, выделяемый для одного пикселя — в битах, правая часть выражения делится на 8.

Кодирование звука

Для понимания принципов кодирования звука лучше всего для начала усвоить принцип работы цифрового микрофона, т. е. как компьютер получает от окружающего мира звуковую информацию. В микрофоне

установлена мембрана — тоненькая плёнка, которая способна отклоняться от своего среднего (нейтрального) положения при колебаниях окружающего воздуха. Не будем вдаваться в функционирование этого явления с физической точки зрения. Для нас важно, что отклонения этой мембраны можно записывать в цифровом виде много раз в секунду. Количество измерений в секунду называется **частотой дискретизации**. Например, 40 тысяч раз в секунду происходит измерение текущего отклонения мембраны. Говорят, что частота дискретизации при этом — 40 кГц, а значение отклонения мембраны от нейтрального положения записывают при помощи определённого количества бит. Чем это значение больше, тем точнее будет запись звука. Правило кодирования следует из формулы Хартли (более подробно о формуле см. на стр. 9): $2^{\text{количество\_бит}}$ = количество_различных_значений отклонения мембраны. Эти различные значения называют **уровнями квантования**.

Ещё нужно понимать, что звук зачастую записывается мультиканальный. То есть, когда запись идёт только с одного микрофона, это называется **моно**. Когда с двух микрофонов — **стерео**. Иногда используют большее количество каналов. Например, квадро (4) или даже 7.1 (8 каналов).

Итак, при кодировании звука происходит запись по каждому каналу с определённой частотой дискретизации в течение определённого времени. Для каждого измерения используется определённое количество бит.

Получаем формулу для вычисления объёма неупакованного звукового файла:

$$V = Ch \cdot \nu \cdot i \cdot t \,/\, 8,$$

где V — объём неупакованного звукового файла (в байтах), Ch — количество каналов (1 — моно, 2 — сте-

рео, 4 — квадро, и т. д.), ν — частота дискретизации, i — количество бит на одно измерение (разрешение), t — время записи в секундах. Так как левая часть — в байтах, а правая — в битах, правую часть делим на 8.

Разбор типовых задач

Вычисление объёма текстовой инфоромации

Задача 1. В одной из кодировок Unicode каждый символ кодируется 16 битами.

Определите размер следующего предложения в данной кодировке:

Я к вам пишу — чего же боле? Что я могу ещё сказать?

1) 52 байт
2) 832 бит
3) 416 байт
4) 104 бит

Решение

В этой задаче используется простая формула для вычисления количества информации в сообщении (более подробно см. стр. 9): $I = k \cdot i$. Для её использования необходимо знать количество информации в одном символе (дано — 16 бит) и количество символов в сообщении. Важно понимать, что под символами имеются ввиду не только печатаемые символы, но ещё и пробелы, их тоже требуется передавать. Значит, для каждого из них будем использовать те же 16 бит, что и для остальных символов. Аккуратно посчитаем количество символов в приведённом сообщении, включая пробелы и знаки препинания. Подсчитываем, и получаем 52.

Подставляем в формулу: $I = 52 \cdot 16 = 832$ бит.

Ответ: 2.

Если бы в списке предлагаемых вариантов ответа не было бы этого значения — 832 бит, имело бы смысл поискать ответ в виде количества байт. То есть, $52 \cdot 2 = 104$ байт.

Задача 2. Некоторый текст, изначально записанный в кодировке КОИ-8, был перекодирован и записан в 16-битной кодировке Unicode. Известно, что в процессе этого перекодирования объём текста, занимаемый им на диске, увеличился на 640 бит. Определите количество символов в тексте. Ответ запишите в виде числа.

Решение

В процессе перекодирования текста количество символов в нём не изменяется. Обозначим количество символов в тексте через k. Найдём количество бит, которое занимал текст до перекодирования. Воспользуемся известной формулой для вычисления количества информации в сообщении: $I = k \cdot i$. Для её использования необходимо знать количество бит, затраченное на один символ. По условию известно, что текст записан в кодировке КОИ-8. Отсюда мы понимаем, что это разновидность таблицы ASCII. Следовательно, один символ кодируется 8-ю битами. Считаем объём информации в сообщении: $I = 8 \cdot k$.

Теперь посчитаем объём информации в перекодированном тексте.

По этой же формуле получаем $I = 16 \cdot k$. Найдём изменение этого количества. Вычтем из полученного объёма исходный: $16k - 8k = 8k$.

В условии объём текста увеличился на 640 бит. Это по нашим расчётам $8k$. Приравняем эти две величины и находим отсюда k: $8k = 640 \rightarrow k = 640 / 8 = 80$ символов.

Ответ: 80.

Данную задачу можно было решить проще. В процессе перекодирования каждый символ из исходных 8-ми бит (таблица ASCII) стал кодироваться 16-ю битами (Юникод), т. е. занимать на 8 бит больше. Так как весь текст увеличился на 640 бит, можно найти, сколько было символов:

$$640 / 8 = 80.$$

Вычисление объёма растрового изображения.

Задача 3. Какой объём на диске (в Мбайт) занимает неупакованное растровое изображение размером 1024×2048 пикселей, если для хранения кода цвета используется 16 бит.

Решение

Подставим имеющиеся данные в формулу: $V = H \cdot W \cdot i \,/\, 8 = 1024 \cdot 2048 \cdot 16 \,/\, 2 = 1024 \cdot 1024 \cdot 4$. Так как ответ нужно дать в Мбайт, а величина V по этой формуле вычисляется в байтах, нужно величину V перевести в Мбайты. То есть, поделить дважды на 1024. Получится ответ: 4.

Ответ: 4.

Может так случиться, что в задаче не будет явно указано количество бит, которое используется для кодирования цвета одного пикселя. При этом будет указано максимальное количество цветов, которое используется в данном изображении. В таком случае, для вычисления количества бит, которое требуется для кодирования цвета пискеля, необходимо воспользоваться уже знакомой нам формулой Хартли: $2^i \geqslant N$ (более подробно см. на стр. 9).

В качестве количества равновероятных событий здесь выступает максимальное количество цветов изображения. То есть, видоизменение формулы Хартли применительно к подобной задаче будет иметь вид: $2^{\text{число\_бит}} \geqslant$ число_цветов_в_изображении.

Задача 4. На диске хранится неупакованное 16-цветное изображение размером 512×4096 пикселей. Какой объём на диске оно занимает? Считать, что для хранения кода пикселя используется минимально возможное количество бит. Ответ укажете в Мбайт.

Решение

Сразу воспользоваться формулой $V = H \cdot W \cdot i \,/\, 8$ мы не можем, потому что нам не известно i — количество бит, используемое для хранения кода одного пикселя. Найдём это количество, используя формулу Хартли: $2^{\text{число\_бит}} \geqslant 16 \rightarrow$ Число бит $i = 4$. Теперь у нас есть значения всех параметров, которые нужно подставить в формулу:
$V = H \cdot W \cdot i \,/\, 8 = 512 \cdot 4096 \cdot 4 \,/\, 8 = 2^9 \cdot 2^{12} \cdot 2^2 \,/\, 2^3 =$
$= 2^{23} \,/\, 2^3 = 2^{20}$ байт. Переводим эту величину в Мбайты. Для этого поделим её на 2^{20}.

Ответ: 1.

Кодирование звука.

Задача 5. Происходит четырёхканальная запись звука с частотой дискретизации 4 кГц и разрешением 16 бит. Запись длится полминуты. Определите ближайшее целое количество Мбайт, которое нужно примерно потратить на диске для хранения неупакованного звукового файла. В ответе запишите только число.

Решение

Подставим величины в формулу:
$V = Ch \cdot \nu \cdot i \cdot t \,/\, 8 = 4 \cdot 40\,000 \cdot 16 \cdot 30 \,/\, 8 = 960000$ байт.
Переводим полученную величину в Мбайты:
960 000 / (1024 · 1024). Данная величина чуть меньше 1. Значит, округляем до ближайшего целого и получаем ответ: 1.

Ответ: 1.

Иногда в задаче может быть не указано разрешение записи. Вместо этого будет указано количество уровней квантования. В этом случае необходимо сначала найти разрешение по формуле Хартли: $2^i \geqslant N$ (более подробно см. на стр. 9), где i — разрешение записи (в битах), N — количество уровней квантования.

Задача 6. Производится двухканальная (стерео) звукозапись с частотой дискретизации 16 кГц. При записи используется количество уровней квантования, равное 32. Запись длится 30 минут, её результаты записываются в файл, сжатие данных не производится. Определите объём полученного звукового файла в Мбайт. В качестве отчета укажите ближайшее число, кратное 5.

Решение

В данном случае нам не дано разрешение, с которым производится запись. Найдём его, зная количество уровней квантования: $2^i \geqslant 32 \rightarrow i = 5$.

Теперь подставим все полученные величины в формулу объёма звукового файла: $V = Ch \cdot \nu \cdot i \cdot t \ / \ 8 = 2 \cdot 16\,000 \cdot 5 \cdot 30 \cdot 60 \ / \ 8 = 36\,000\,000$ байт.

Преобразуем полученную величину в Мбайты: $36\,000\,000 \ / \ (1024 \cdot 1024)$. Ближайшая полученная величина, кратная 5 = 35.

Ответ: 35 Мбайт.

6 Создание и обработка информационных объектов

Базы данных

Конспект

В данном разделе мы рассмотрим некоторые технологии, которые используются для хранения на компьютере структурированной информации.

Наиболее традиционным способом структурирования информации является табличный способ. Это значит, что в хранимой информации выделяются отдельные объекты, данные о которых мы собираемся хранить. Эти данные (о каждом объекте) разбиваются на отдельные свойства.

Например, нам потребовалось сохранить данные о канцелярских принадлежностях. Выделяем среди них пишущие принадлежности. Понимаем, что одним объектом будет одна пишущая принадлежность. Анализируем свойства пишущих принадлежностей и выделяем среди них такие, например, как тип (карандаш, фломастер, авторучка, ...), цвет, одноразовость применения, толщину линии.

После этого всё готово для формирования таблицы.

Столбцы таблицы назовём так же, как свойства пишущих принадлежностей.

В строках будем хранить информацию об отдельных объектах.

Тип	Цвет	Однора-зовость	Толщина линии, мм
Карандаш	Красный	Да	2
Авторучка	Синий	Нет	1
Фломастер	Фиолетовый	Да	3

Строки такой таблицы принято называть **записями**. Столбцы — **полями**. Каждое поле имеет свой тип данных, т. е. информация в столбце всегда одного типа. Каждая строка таблицы хранит информацию об одном объекте.

Конечно, весьма часто требуется хранить информацию не только об объектах одного вида. И при этом объекты разного вида описываются разными наборами свойств и связаны ещё определенными отношениями. Например, ученики и учителя школы.

В этом случае для каждой категории объектов создаётся отдельная таблица и эти таблицы связываются между собой нужными отношениями. Такой метод называется **реляционной базой данных**. Его подробное изучение не является целью нашего пособия, так как не применяется в заданиях на экзамене по ОГЭ.

Для нас достаточно понимать, что информация хранится в виде таблицы и из этой таблицы информацию необходимо уметь определённым образом извлекать. Например, анализировать количество записей, обладающих указанными свойствами.

Разбор типовых задач

Задача 1. Ниже в табличной форме представлен фрагмент базы данных «Отправление поездов дальнего следования».

Пункт назначения	Категория поезда	Время в пути	Вокзал
Махачкала	скорый	39.25	Павелецкий
Махачкала	скорый	53.53	Курский
Мурманск	скорый	35.32	Ленинградский
Мурманск	скорый	32.50	Ленинградский
Мурманск	пассажирский	37.52	Ленинградский
Мурманск	пассажирский	37.16	Ленинградский
Назрань	пассажирский	40.23	Павелецкий
Нальчик	скорый	34.55	Казанский
Нерюнгри	скорый	125.41	Казанский
Новосибирск	скорый	47.30	Ярославский
Нижневартовск	скорый	52.33	Казанский
Нижний Тагил	фирменный	31.36	Ярославский

Сколько записей в данном фрагменте удовлетворяют условию:

(Категория поезда = «скорый») И (Время в пути > 36.00)?

В ответе укажите одно число — искомое количество записей.

Решение

При выборе метода решения задачи очень важным фактором является его надёжность получения правильного от-

вета. Мы прекрасно понимаем, что эту задачу (и ещё многие другие задачи) можно решить устно. Но при этом достаточно велик риск получения ошибки. На экзамене будет достаточно много времени, чтобы решать предлагаемые задачи спокойно, методично и не торопясь. Поэтому предлагаем способы решения, которые сводят риски получения ошибок к минимуму.

Именно такой способ мы настоятельно рекомендуем вас использовать и в данной задаче.

Условие, которое в этой задаче нам предлагают проанализировать, является логическим выражением. Нам нужно вычислить это логическое выражение для каждой строки таблицы и посчитать, для какого количества строк получился истинный результат. Наиболее надежным способом нахождения результатов логического выражения, как мы уже писали в главе про логические операции и выражения, является построение таблицы истинности.

Анализируем приведённое логическое выражение на предмет количества логических утверждений и логических операций. Видим в нём два логических утверждения — **(Категория поезда = «скорый»)** и **(Время в пути > 36.00)**. Также замечаем одну логическую операцию — **И**. Очевиден порядок выполнения этих действий: сначала проверяются логические утверждения (они записаны в скобках, у них высший приоритет), затем выполняется операция логическое **И** над результатами проверки данных логических утверждений.

Получаем порядок выполнения действий:

1 3 2

(Категория поезда = «скорый») И (Время в пути > 36.00)?

Дорисовываем рядом с данной нам в условии таблицей ещё три столбца и подписываем номера операций:

				1	2	3
Пункт назначения	**Категория поезда**	**Время в пути**	**Вокзал**	**Категория поезда = «скорый»**	**Время в пути > 36.00**	**(1) И (2)**
Махачкала	скорый	39.25	Павелецкий			
Махачкала	скорый	53.53	Курский			
Мурманск	скорый	35.32	Ленинградский			
Мурманск	скорый	32.50	Ленинградский			
...	...	...				
Нижний Тагил	фирменный	31.36	Ярославский			

Последовательно, столбец за столбцом, вычисляем значения логических утверждений и логической операции **И**:

				1	2	3
Пункт назначения	**Категория поезда**	**Время в пути**	**Вокзал**	**Категория поезда = «скорый»**	**Время в пути > 36.00**	**(1) И (2)**
Махачкала	скорый	39.25	Павелецкий	1	1	1
Махачкала	скорый	53.53	Курский	1	1	1
Мурманск	скорый	35.32	Ленинградский	1	0	0
Мурманск	скорый	32.50	Ленинградский	1	0	0
Мурманск	пассажирский	37.52	Ленинградский	0	1	0
Мурманск	пассажирский	37.16	Ленинградский	0	1	0
Назрань	пассажирский	40.23	Павелецкий	0	1	0
Нальчик	скорый	34.55	Казанский	1	0	0
Нерюнгри	скорый	125.41	Казанский	1	1	1
Новосибирск	скорый	47.30	Ярославский	1	1	1
Нижневартовск	скорый	52.33	Казанский	1	1	1
Нижний Тагил	фирменный	31.36	Ярославский	0	0	0

Подсчитываем количество истинных значений (количество единиц) в последнем столбце.

Ответ: 5.

 Если вы считаете, что приведённый метод слишком трудоёмок с точки зрения оформления, можете использовать другой способ: вместо подробного рисования дополнительных столбцов и отмечания в них нулей и единиц, пририсуйте рядом с таблицей три дополнительных столбца и отметьте «галочками» в первом дополнительном столбце те строки, для которых выполняется первое условие (Категория поезда = «скорый»). Затем отметьте «галочками» во втором дополнительном столбце те строки, для которых выполняется второе условие (Время в пути > 36.0). После в третьем дополнительном столбце отметьте «галочками» те строки, в которых уже записано две «галочки» в первом и втором дополнительных столбцах. Подсчитайте количество «галочек» в последнем столбце.

7 Поиск информации

Поисковые запросы

Конспект

При использовании всемирной компьютерной сети Интернет достаточно часто требуется найти в ней сайт или страницу, которую вы раньше не посещали, но которая, возможно, содержит нужную вам информацию. Для этого существуют специальные поисковые службы, называемые **поисковые серверы**. Наиболее известные из них — yandex.ru, google.ru и другие.

Тот текст, который пользователь пишет в специальное поле поиска, называется **поисковым запросом**. Полезно понимать, что в тот момент, когда пользователь отправляет поисковый запрос на поисковый сервер, тот не начинает в этот момент рыскать по всем просторам Интернета в поисках нужной информации. Поиск осуществляется только в базе ключевых слов, которая хранится в поисковой системе. Поисковая система анализирует слова, написаные в поисковом запросе, выделяет из них важные (**ключевые**) слова и по ним уже осуществляет поиск в собственных хранилищах. **Ссылки** на найденные страницы выдаются пользователю в качестве результата поискового запроса.

Откуда же при этом поисковая система знает эту информацию (ключевые слова и ссылки), которая хранится у неё?

Для этого работают две основные технологии.

Первая — владельцы сайтов, которые хотят, чтобы их сайты/страницы часто посещались и приноси-

ли соответствующий доход/престиж/предоставляли информационные услуги, сами регистрируют в поисковых системах свои сайты/страницы по ключевым словам.

Вторая — поисковые системы сами «просматривают» сайты и страницы сети Интернет, анализируя их, выделяя ключевые слова и сохраняя ссылки на них. Но, ещё раз уточним, что это происходит не в момент обращения пользователя.

Для того чтобы обеспечить гибкость поиска, поисковые серверы позволяют указывать ключевые слова, по которым осуществляется поиск с использованием различных логических операций (о логических операциях подробнее рассмотрено на стр. 47).

Среди логических операций нас будут интересовать:

– логическое **ИЛИ** (дизъюнкция), обозначаемая в поисковых запросах символом «|» (вертикальная черта);

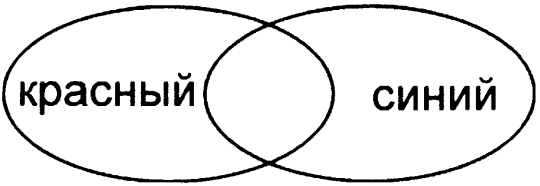
– логическое **И** (конъюнкция), обозначаемая в поисковых запросах символом «&».

Рассмотрим влияние этих логических операций на результат (количество найденных страниц) поискового запроса.

Для начала сделаем это на примере двух ключевых слов — «красный» и «синий».

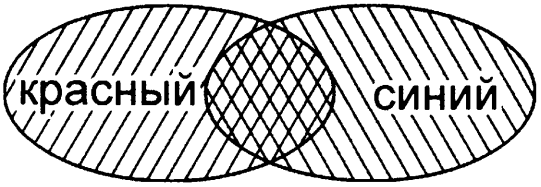
Запрос каждого из этих слов по отдельности выдаёт страницы, которые содержат, соответственно, те, где встречается слово «красный» и те, где встречается слово «синий». Нужно понимать, что есть страницы, которые содержат слово «красный», но не содержат слово «синий»; есть те, которые, наоборот, содержат только слово «синий» и не содержат слово «красный»; а есть такие, которые содержат и то, и другое слово вместе.

Изобразим множества этих страниц при помощи диаграммы:

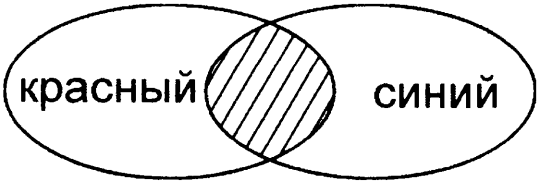


На диаграмме можно видеть, что множества страниц, содержащих слова «красный» и «синий» частично пересекаются.

Рассмотрим результат операции логическое **И** на примере результата запроса «красный & синий». Это те страницы, которые одновременно содержат слова и «красный» и «синий». На диаграмме это та область, которая одновременно входит в оба нарисованных овала. Заштрихуем область «красный» левой штриховкой, а область «синий» правой штриховкой. Операция логическое **И** требует одновременного наличия обоих слов. То есть, нас интересует пересечение штриховок:



А именно, та область, которая лежит на пересечении между ними:

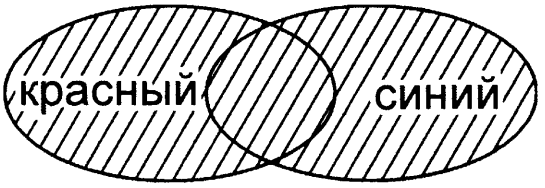


Важно понимать, что операция логическое **И** (&) приводит к уменьшению количества найденных стра-

ниц (требуется найти страницы, удовлетворяющие более строгому условию).

Теперь рассмотрим результат операции логическое **ИЛИ** на примере результата запроса «красный | синий». Это те страницы, которые содержат или только слово «красный», или только слово «синий», или слова «красный» и «синий» одновременно. На диаграмме это та область, которая входит хотя бы в один нарисованный овал. Также заштрихуем область «красный» и область «синий». Только теперь будем штриховать с наклоном в одну сторону (при операции логическое **ИЛИ** необходимо объединять области, поэтому можно выбрать одинаковую штриховку).

Объединение областей даст такой результат:



Важно понимать, что операция логическое **ИЛИ** (|) приводит к увеличению количества найденных страниц (требуется найти страницы, которые содержат хотя бы одно из условий).

Возможно, вам известно из курса математики, что приведённые диаграммы называются **диаграммами Эйлера-Венна**.

Разбор типовых задач

Задача 1. Приведены запросы к поисковому серверу. Для каждого запроса указан его код — соответствующая буква от А до В. Запишите в таблицу коды запросов слева направо в порядке **возрастания** количества страниц, которые нашёл поисковый сервер по

каждому запросу. По всем запросам было найдено разное количество страниц.

Для обозначения логической операции **ИЛИ** в запросе используется символ «|», а для логической операции «И» — символ «&».

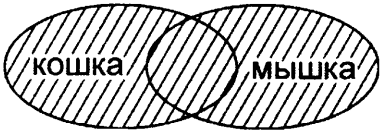
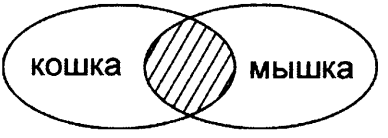
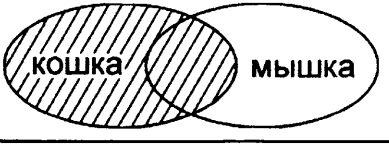
Код	Запрос
А	кошка | мышка
Б	кошка & мышка
В	кошка

Решение

• Первый способ решения задачи.

Используем диаграммы Эйлера-Венна.

Построим их для каждого из запросов и сравним площади заштрихованных областей.

А. кошка | мышка	**Б. кошка & мышка**
Это логическое **ИЛИ** — объединение. В результат попадут все области, которые входят хотя бы в один овал.	Это логическое **И** — пересечение. В результат попадёт только область, которая входит в оба овала.
	
В. кошка	
Это просто овал, соответствующий «кошке».	
	

Соотносим размеры заштрихованных областей и понимаем, что наименьшая площадь — у области Б, следующая — у области В, и самая большая — у области А. Получаем ответ: БВА.

Ответ: БВА.

• Второй способ решения задачи.

Составим таблицу истинности для каждого выражения.

В качестве столбцов-аргументов используем «кошка» и «мышка». В качестве результатов этих столбцов используем 0 для обозначения того, что это слово не присутствует на страницы, и 1, если присутствует. В таблице будут два столбца исходных данных (в запросе только два различных слова — «кошка» и «мышка»), столбец для результата операции А («кошка | мышка») и столбец для результата операции Б («кошка & мышка»). Отдельного столбца для операции В («кошка») не требуется, потому что он такой же, как первый столбец. Заполняем таблицу и вычисляем результаты операций.

кошка	мышка	кошка | мышка	кошка & мышка
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1
В		А	Б

Теперь подсчитываем количество единиц в каждом интересующем нас столбце:

В		А	Б
2		3	1

Эти количества располагаем, как требуется в условии задачи, в порядке возрастания: 1–2–3 и записываем в качестве ответа соответствующие им коды запросов: БВА.

Ответ: БВА.

• Третий вариант решения.

Быстрый, но не очень надёжный. Рекомендуется только при высоком навыке решения подобных задач.

Просто анализируем варианты запросов на предмет логическое **И**/логическое **ИЛИ**. Понимаем, что логическое **И** уменьшает количество найденных страниц, а логическое **ИЛИ** — увеличивает. Получаем, что меньше всего страниц будет найдено для логического **И**. Значит, этот запрос нужно ставить в ответе на первое место (требуется упорядочить запросы по возрастанию). А больше всего страниц будет найдено для логического **ИЛИ**. Значит, этот запрос нужно ставить в ответе на последнее место. Посередине остаётся просто запрос «кошка».

Ответ: БВА.

Возможно, при чтении вариантов решения у вас возник вопрос, что делать, если решение не однозначно. Например, при построении диаграмм Эйлера-Венна необходимо сравнивать площади областей, не вложенных друг в друга или при подсчёте числа единиц в таблице истинности получаются одинаковые количества в разных столбцах. Однако, такая ситуация не случится, потому что подобная задача не имеет чёткого решения. Например, нельзя решить задачу, отвечая на вопрос, какой запрос выдаст большее количество страниц — «красный» или «синий». Возможно и так, и так, в зависимости от текущей ситуации. В общем, если у вас получился один или другой перечисленный

случай — ищите ошибку в решении (или, если это не на экзамене, спросите у составителя задачи, не ошибся ли он).

Также заметим, что решение задачи для четырёх переменных при помощи диаграмм Эйлера-Венна существенно сложнее, чем для трёх переменных, потому что корректно построить в этом случае диаграмму не очень просто. Впрочем, решение задачи для четырёх переменных при помощи таблицы истинности тоже трудоёмко. Ведь при этом количество строк в таблице становится равно 16 ($= 2^4$).

Задача 2. Приведены запросы к поисковому серверу. Для каждого запроса указан его код — соответствующая буква от А до Г. Запишите в таблицу коды запросов слева направо в порядке убывания количества страниц, которые нашел поисковый сервер по каждому запросу. По всем запросам было найдено разное количество страниц.

Для обозначения логической операции «ИЛИ» в запросе используется символ «|», а для логической операции «И» — символ «&».

Код	Запрос
А	яблоки | груши | сливы
Б	яблоки | сливы
В	(яблоки | груши) & сливы
Г	сливы | (сливы & груши)

Решение (1 способ)

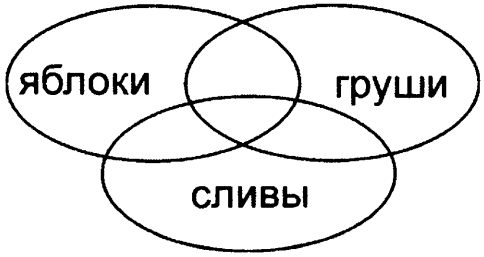
Построим диаграммы Эйлера-Венна для каждого случая. В запросах Б и В используется только по два различных слова — «яблоки» и «сливы» в запросе Б,

«сливы» и «груши» в запросе В. При этом в запросах А и Г используется уже по три слова. Чтобы после построения всех диаграмм можно было удобно сопоставить площади заштрихованных областей, рекомендуем во всех четырёх случаях рисовать диаграммы сразу для всех трёх слов.

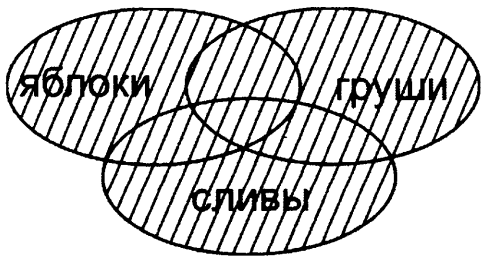
Построение диаграмм для запросов А и Б не представляет никакой сложности — это обычные диаграммы для операции логическое **ИЛИ**.

• Случай **А**.

Рисуем овалы для всех трёх слов. Причём так, чтобы они частично пересекались друг с другом. Обратите внимание — каждый овал должен частично пересекаться с каждым другим овалом.



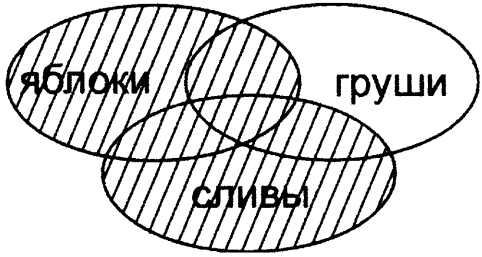
Теперь заштрихуем каждый из овалов. Можно осуществлять штриховку в одну сторону, ведь их все равно в результате операции логическое **ИЛИ** нужно будет объединять.



• Случай **Б**.

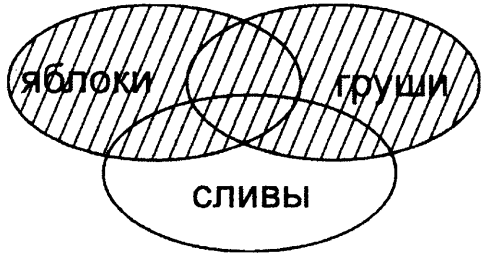
Построим те же частично пересекающиеся три овала, как сделали это в случае А. Только теперь заштри-

хуем только те области, которые нужны нам в запросе Б («яблоки» и «сливы»).

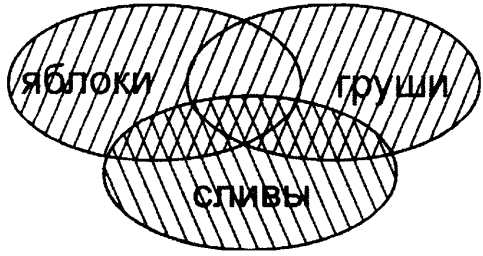


– Случай **В**.

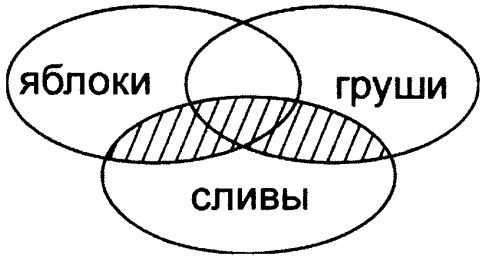
Как и ранее, начинаем с рисования точно таких же трёх частично пересекающихся овалов, как и в случаях **А** и **Б**. Но теперь у нас в поисковом запросе одновременно встречается оба действия — логическое **И** и логическое **ИЛИ**. Решаем задачу по действиям. Сначала выполним операцию в скобках. То есть, заштрихуем область для «яблоки | груши» с помощью обычной штриховки в одну сторону.



Теперь нужно пересечь эту область по «сливами». То есть, выполнить действие «(...) & сливы». Для этого заштрихуем «сливы» линиями в другую сторону.

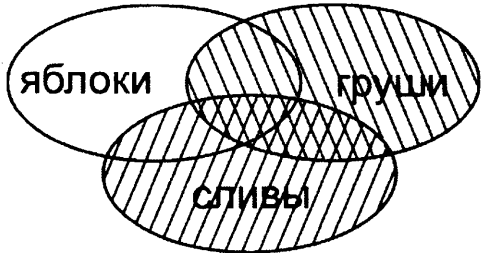


Интересующая нас область лежит на пересечении штриховок. Это область:

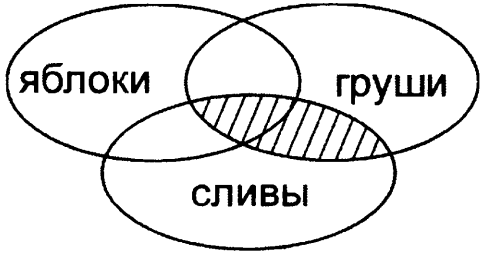


– Случай Г.

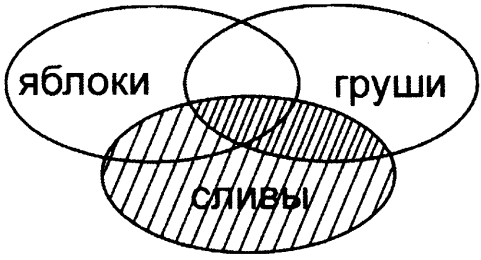
Как и ранее, строим сначала все три частично пересекающиеся области. Здесь тоже используются обе логические операции. Будем делать задание по действиям. Сначала строим выражение в скобках («сливы & груши»). Закрасим левой и правой штриховками области «сливы» и «груши».



Интересующая нас область лежит на пересечении штриховок. То есть:

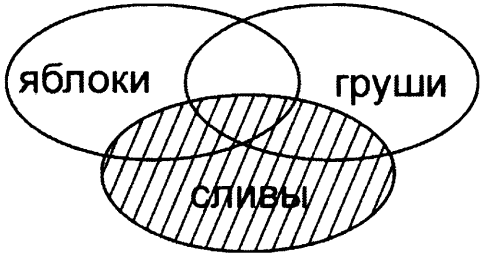


Теперь сделаем вторую операцию — «сливы & (...)». Для этого заштрихуем область «сливы» в ту же сторону и найдём область, которая будет заштрихована.

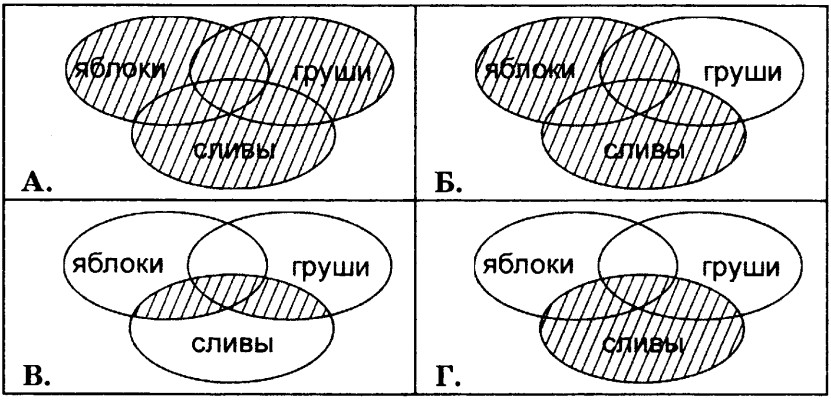


Получилось, что заштрихованная область совпадает с областью «сливы».

Результатом является область:



Сравниваем между собой области, полученные во всех четырёх случаях:



Нам нужно расположить запросы в порядке убывания количества найденных страниц. Значит, сначала нужны области большей площади, а потом меньшей.

Самая большая площадь в случае А. Затем — в случае Б. Затем — в случае Г. Самая маленькая — в случае В. Получаем *ответ*: АБГВ.

(2 способ)

Второй способ решения задачи — построение таблицы истинности. Так как в запросах встречается три различных слова, таблица будет содержать 8 строк ($= 2^3$).

Каждый из запросов будем строить последовательно. В запросе А две операции. Обе операции — логическое **ИЛИ**. Выполняем их слева направо, сначала «(яблоки | груши)», затем «(...) | сливы».

			(1)	
яблоки	**груши**	**сливы**	**яблоки | груши**	**(1) | сливы**
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Операция в запросе Б самая простая. Это просто логическое **ИЛИ** для столбцов «яблоки» и «сливы».

яблоки	**груши**	**сливы**	**яблоки | сливы**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

В запросе В выполняем сначала операцию в скобках «(яблоки | груши)», затем «(...) & сливы».

(1)

яблоки	груши	сливы	яблоки | груши	(1) & сливы
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

В запросе Г также сначала выполняем операцию в скобках «сливы & груши». Затем «сливы | (...)».

(2)

яблоки	груши	сливы	сливы & груши	сливы | (2)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Мы привели все четыре таблицы по отдельности, чтобы понятнее было, как это делается по действиям. В действительности гораздо удобнее (и быстрее) строить одну общую таблицу сразу для всех четырёх запросов.

			(1)				(2)	
яблоки	**груши**	**сливы**	**яблоки | груши**	**(1) | сливы**	**яблоки | сливы**	**(1) & сливы**	**сливы & груши**	**сливы | (2)**
0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	0	0	0
0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	0	0	0
1	0	1	1	1	1	1	0	1
1	1	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1
				А	Б	В		Г

Подсчитываем количество единиц в каждом запросе:

				А	Б	В		Г
				7	6	3		4

Располагаем коды запросов в порядке убывания чисел (7–6–4–3).

Ответ: АБГВ.

 Решение этой задачи устно, исходя из соображений больше страниц при логическом ИЛИ/меньше страниц при логическом И, чревато потенциальными ошибками. Потому что запросы В и Г достаточно сложны для устного решения.

Задача 3. Приведены запросы к поисковому серверу. Для каждого запроса указан его код — соответствующая буква от А до Г. Запишите в таблицу коды запросов слева направо в порядке возрастания количества страниц, которые нашёл поисковый сервер по каждому запросу. По всем запросам было найдено разное количество страниц.

Для обозначения логической операции **ИЛИ** в запросе используется символ «|», а для логической операции **И** — символ «&».

Код	Запрос
А	солнце & воздух
Б	солнце | воздух | вода
В	солнце | воздух | вода | огонь
Г	солнце | воздух

Решение

Построение диаграмм Эйлера-Венна в данном случае трудоёмко и не очень целесообразно. В запросах используется четыре различных слова. Диаграмма будет сложной. А запросы при этом даны простые.

Решим задачу построением таблицы истинности. В ней нужно строить 16 ($= 2^4$) строк.

По условию в каждом запросе встречается только одно действие. Они легко выполняются слева направо. Единственная неприятность — долго строить саму таблицу.

					(1)	(2)	
солнце	**воздух**	**вода**	**огонь**	**солнце & воздух**	**солнце | воздух**	**(1) | вода**	**(2) | огонь**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	1
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	1
0	1	1	0	0	1	1	1
0	1	1	1	0	1	1	1
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
				А	Г	Б	В
Количество единиц в запросе				4	12	14	15

Количество единиц в порядке возрастания: 4–12–14–15.

Ответ: АГБВ.

Заметим, что ввиду простоты запросов, используемых в приведённой задаче, тот же результат можно легко и быстро получить, исходя из тех соображений, что логическое **ИЛИ** увеличивает количество найденных страниц, а логическое **И** — уменьшает. Чем больше слов объединяет логическое **ИЛИ**, тем больше страниц будет найдено. Чем больше слов пересекает логическое **И**, тем меньше страниц будет найдено.

В данном случае, пересечение (логическое **И**) только в одном запросе. Это даст самое меньшее количество страниц. Остальные запросы нужно расположить в порядке увеличения количества объединяемых операцией логическое **ИЛИ** поисковых слов. То есть:

солнце & воздух	А
солнце | воздух	Г
солнце | воздух | вода	Б
солнце | воздух | вода | огонь	В

Ответ: АГБВ.

8 Проектирование и моделирование

Графическое представление информации

Конспект

Введение

Достаточно часто для представления информации о связях между объектами, процессами или явлениями используют **графические способы представления информации.** Одним из весьма распространенных графических способов является **граф.**

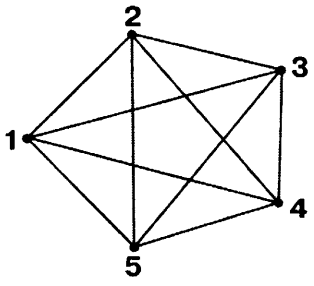
Формально граф — это совокупность некоторого количества объектов (обычно называемых **вершинами**) и соединяющих их связей (обычно называемых **рёбрами**). Без сомнения, вы много раз видели графы. Обычно в виде графов рисуют схемы метрополитена, железных дорог/электричек или городов и связывающих их дорог.

Итак, ещё раз более простыми словами. Граф состоит из некоторого количества вершин (называемых также **узлами** или **пунктами**). В привычных для нас перечисленных выше схемах вершинами графа являются станции или населённые пункты. Соединяющие их связи — это дороги между станциями/населёнными пунктами. Их, как мы уже упомянули, называют рёбрами графа.

Для того, чтобы описать, как вершины графа соединяются рёбрами, применяют либо **графический**, либо **табличный способы.** Графический способ весьма наглядный, именно его мы представляем себе, ког-

да работаем с графом. Но для обработки информации о графе на компьютере такой способ представления графа (в виде рисунка) чаще всего неудобен. При обработке на компьютере используют обычно какой-нибудь из табличных способов. Один из самых популярных табличных способов — так называемая **матрица смежности**. Пусть вас не пугают эти слишком заумные слова — «матрица» и «смежности». Матрица — в данном случае всего лишь квадратная таблица. Смежность — представление о том, существует ли ребро между парой вершин (то есть, являются ли они соседними — смежными). В этой матрице-таблице столько строк и столбцов, сколько вершин в графе. На пересечении строки номер X и столбца номер Y хранится число 0 или 1. Число 1 — если существует связь (ребро) между вершинами X и Y. Число 0 — если такого ребра не существует.

Пример графа и соответствующей ему матрицы смежности.



	1	2	3	4	5
1		1	1	1	1
2	1		1	1	1
3	1	1		1	1
4	1	1	1		1
5	1	1	1	1	

 В действительности, правильнее сказать, что на пересечении строки X и столбца Y записано количество рёбер, которые выходят из вершины X и входят в вершину Y. Отличие этого определения от того, что мы привели чуть выше, оказывается в случае, когда из вершины X в вершину Y идёт более одного ребра. Либо в случае, когда ребро из вершины X в вершину Y есть, а в обратную сторону (из вершины Y в вершину X) такого ребра нет.

Деревья

В случае, если в графе из любой вершины в любую другую можно добраться единственным образом, граф можно считать **деревом**. Другое определение дерева — это граф, в котором отсутствуют циклы. То есть, нельзя «выйти» из какой-либо вершины дерева и, проходя по любому ребру не более одного раза, вернуться обратно в ту же вершину.

Когда все рёбра дерева имеют направления (по ним можно двигаться только в направлении, указанном стрелкой), дерево называется **направленным** (или **ориентированным**). В дальнейшем мы будем пользоваться только направленными деревьями, хотя и не будем для простоты рисовать на каждом ребре стрелку.

В направленном дереве выделяют специальную вершину — **корень дерева**. Это вершина, в которую не входит ни одно ребро. А также часть вершин называют **листьями**. Это вершины, из которых не выходит ни одного узла. Если из вершины выходит хотя бы одно ребро, вершина называется **узлом дерева**. Заметим, что корень дерева так же является узлом дерева.

Обычно деревья (структуры данных) рисуют сверху вниз или слева направо. То есть, корень дерева располагается сверху, а ветви направлены вниз, либо корень дерева располагается слева, а ветви направлены вправо.

Таблица расстояний

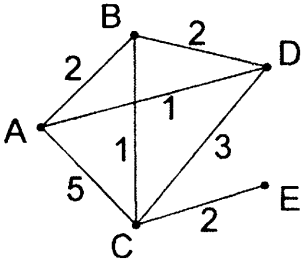
В ряде случаев при помощи графа требуется отобразить не только наличие связей между вершинами, но и расстояния между ними. Например, расстояния между соседними населёнными пунктами по дорогам.

При изображении графа в виде рисунка эти расстояния просто указываются прямо на рёбрах.

При хранении графа в табличном виде обычно используют таблицу расстояний. В ней на пересечении строки X и столбца Y записано число, которое показывает длину ребра из вершины X в вершину Y. Если из вершины X в вершину Y нет ребра, клетка в таблице остаётся пустой.

При обработке программным образом ячейки таблицы, для которых не существует соответствующего ребра, хранят обычно очень большое число, которое никак не влияет на алгоритм поиска в таблице. Либо хранят отрицательное число, которое алгоритм проверяет, чтобы понимать существование или не существование ребра.

Пример графа с подписанными длинами рёбер и соответствующая ему таблица расстояний:



	A	B	C	D	E
A		2	5	1	
B	2		1	2	
C	5	1		3	2
D	1	2	3		
E			2		

Одной из весьма распространённых задач обработки графов является задача поиска кратчайшего расстояния между двумя вершинами (например, населёнными пунктами). Обычно граф при этом задан таблицей расстояний.

Разбор типовых задач

Задача 1. Между населёнными пунктами A, B, C, D, E построены дороги, протяжённость которых (в километрах) приведена в таблице.

	A	B	C	D	E
A		2	5	1	
B	2		1	2	
C	5	1		3	2
D	1	2	3		
E			2		

Определить длину кратчайшего пути между пунктами A и E.

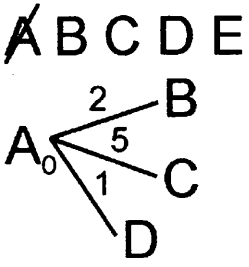
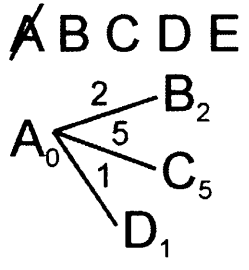
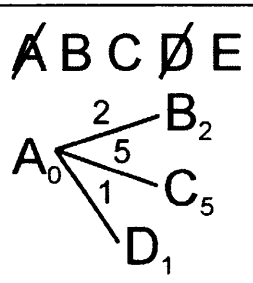
Передвигаться можно только по дорогам, протяжённость которых указана в таблице.

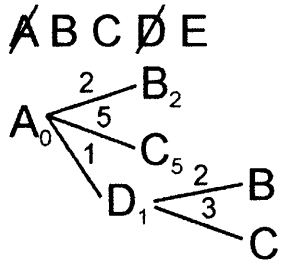
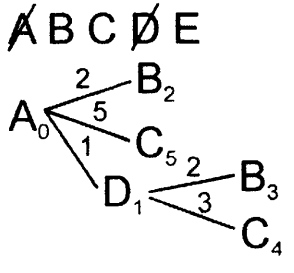
Решение

Для решения этой задачи воспользуемся специальным алгоритмом, который эффективно решает именно такого вида задачу — находит кратчайшее расстояние на графе от одной вершины до другой. Этот алгоритм называется **алгоритмом Дейкстры.** Побочным эффектом алгоритма Дейкстры является нахождение кратчайшего расстояния от некоторой стартовой вершины до всех остальных вершин графа. Но это никоим образом не ухудшает его качества — алгоритм Дейкстры ищет кратчайшее расстояние быстро, надёжно и эффективно.

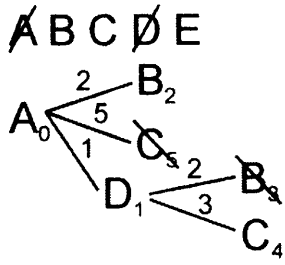
0. Выпишем в отдельный список все возможные вершины графа.	A B C D E

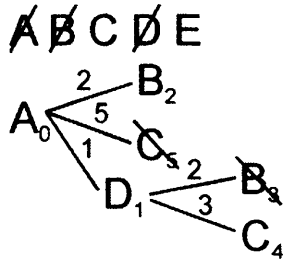
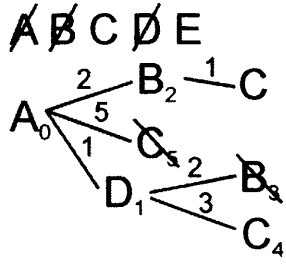
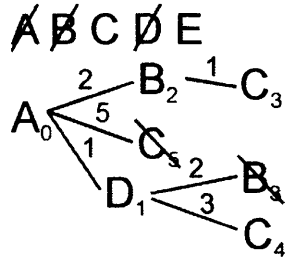
1. Будем строить дерево. Нарисуем стартовую вершину (в нашем случае это точка A) и подпишем рядом с ней расстояние, которое нужно «проехать», чтобы добраться до неё из стартовой точки A. То есть, 0. Эта вершина будет корнем нашего дерева.	A B C D E A_0
2. Из всех возможных листовых вершин дерева найдём вершину с наименьшим числом (в данном случае, такая вершина только одна — A). Вычеркнем эту вершину из списка доступных вершин. В дальнейшем её мы больше не будем рассматривать в качестве возможной вершины дерева. То есть, вычеркнутые вершины в дерево больше не дорисовываются! Расстояния от стартовой вершины до них подсчитаны окончательные и лучше уже не будут. Следующее действие будем делать из этой вершины.	~~A~~ B C D E A_0

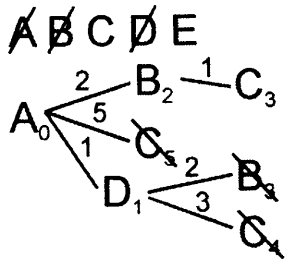
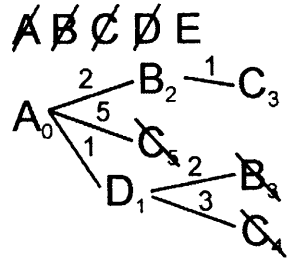
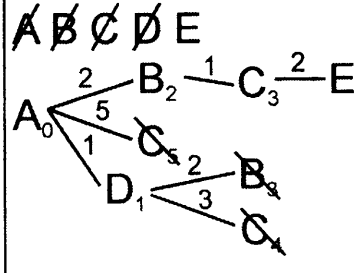
3. По таблице расстояний найдём все вершины, до которых идёт ребро из текущей вершины (сейчас это вершина A) и которые ещё не вычеркнуты. Это вершины B, C, D. Нарисуем из текущей вершины (A) ветви дерева для каждой из найденных смежных вершин. На концах ветвей укажем названия этих вершин. На ветвях подпишем длины рёбер по таблице расстояний.	
4. Для каждой полученной вершины посчитаем расстояние до неё как: расстояние до текущей вершины (подписано рядом с ней, сейчас для вершины A это 0) плюс расстояние (длина ребра) до каждой полученной вершины (подписано над ребром). Запишем эти расстояния рядом с каждой полученной вершиной.	
5. Среди листовых вершин найдём вершину с наименьшим расстоянием. В данном случае это вершина D. Вычеркнем эту вершину из списка доступных вершин. Дальнейшие действия будем делать из неё. Это такое же действие, как действие 2.	

6. По таблице расстояний найдём все вершины, до которых идёт ребро из текущей вершины (сейчас это вершина D) и которые ещё не вычеркнуты. Это вершины B и C. Нарисуем из текущей вершины (D) ветви дерева для каждой из найденных смежных вершин, нарисуем на концах ветвей названия этих вершин. Подпишем на ветвях длины рёбер по таблице расстояний. Это такое же действие, как действие 3.	
7. Для каждой полученной вершины посчитаем расстояние до неё как: расстояние до текущей вершины (подписано рядом с ней, сейчас для вершины D это 1) плюс расстояние (длина рёбра) до каждой полученной вершины (подписано над ребром). Запишем эти расстояния рядом с каждой полученной вершиной. Это такое же действие, как действие 4.	

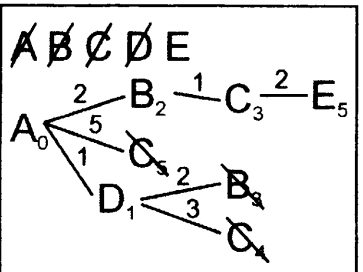
8. Среди всех листовых вершин будем искать одинаковые вершины.
Сейчас это пара вершин B2 и B3 и пара вершин C5 и C4.
Для каждой пары одинаковых вершин будем вычеркивать в дереве худшую вершину. То есть, вершину с бóльшим расстоянием до неё.
Сейчас это B3 и C5. Если в паре одинаковых вершин расстояния будут одинаковые, нужно вычеркнуть одну любую. Мы специально вычеркиваем вершины указанным образом (от левого верхнего к правому нижнему углу), чтобы перечеркнуть как название вершины, так и расстояние до неё. Это облегчит в дальнейшем поиск вершины с наименьшим расстоянием.
Заметим, такого действия (вычеркивания) мы раньше не делали, потому что у нас ещё не встречались одинаковые вершины. В действительности, это часть циклического алгоритма.
Действия 5–8 будем повторять, пока не вычеркнем конечную вершину из списка доступных вершин.



9. Среди листовых вершин найдём вершину с наименьшим расстоянием. В данном случае это вершина B. Вычеркнем эту вершину из списка доступных вершин. Дальнейшие действия будем делать из неё.	
10. По таблице расстояний найдём все вершины, до которых идёт ребро из текущей вершины (сейчас это вершина B) и которые ещё не вычеркнуты. Это вершина C. Нарисуем из текущей вершины (B) ветви дерева для каждой из найденных смежных вершин, нарисуем на концах ветвей названия этих вершин. Подпишем на ветвях длины рёбер по таблице расстояний.	
11. Для каждой полученной вершины посчитаем расстояние до неё как: расстояние до текущей вершины (подписано рядом с ней, сейчас для вершины B это 2) плюс расстояние (длина ребра) до каждой полученной вершины (подписано над ребром). Запишем эти расстояния рядом с каждой полученной вершиной. Сейчас это C3.	

12. Среди всех листовых вершин будем искать одинаковые вершины. Сейчас это С3 и С4. Для каждой пары одинаковых вершин будем вычеркивать в дереве худшую вершину. То есть, вершину с бóльшим расстоянием до неё. Сейчас это С4.	
13. Среди листовых вершин найдём вершину с наименьшим расстоянием. В данном случае это вершина С. Вычеркнем эту вершину из списка доступных вершин. Дальнейшие действия будем делать из неё.	
14. По таблице расстояний найдём все вершины, до которых идёт ребро из текущей вершины (сейчас это вершина С) и которые ещё не вычеркнуты. Это вершина Е. Нарисуем из текущей вершины (С) ветви дерева для каждой из найденных смежных вершин, нарисуем на концах ветвей названия этих вершин. Подпишем на ветвях длины рёбер по таблице расстояний.	

15. Для каждой полученной вершины посчитаем расстояние до неё как: расстояние до текущей вершины (подписано рядом с ней, сейчас для вершины C это 3) плюс расстояние (длина ребра) до каждой полученной вершины (подписано над ребром). Запишем эти расстояния рядом с каждой полученной вершиной. Сейчас это E5.

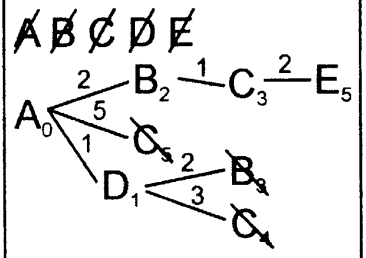


16-17. Среди всех листовых вершин будем искать одинаковые вершины. Сейчас таких пар нет.

Среди листовых вершин найдём вершину с наименьшим расстоянием. В данном случае это вершина E. Вычеркнем эту вершину из списка доступных вершин.

Это конечная вершина, расстояние до которой мы искали. Алгоритм закончен.

Ответ: 5.



Несколько замечаний относительно применения приведённого алгоритма.

Несмотря на ощущение объёма и громоздкости, которые могли возникнуть при изучении этого алгоритма, алгоритм выполняется просто и быстро.

Обратите внимание на количество записей, которые нужно сделать для его выполнения на черновике (это то, что приведено в правом столбце таблицы в последней строке). Всего несколько букв, линий и цифр. Выполняются они очень легко. Освоив эти действия, выполнение их надёжно приводёт к правильному результату.

Хотелось бы предостеречь вас от ощущения того, что правильный ответ гораздо быстрее найти, просто анализируя таблицу расстояний и перебирая всего несколько вариантов. Зачастую это, действительно, так. Но перебирание в уме нескольких вариантов может привести к потенциальным ошибкам. Запись промежуточных действий на черновике позволяет отследить ряд ошибок, используя не только устные рассуждения, но и визуальный контроль своих действий.

Ещё одной альтернативой приведённого алгоритма является построение по таблице расстояний исходного графа. По построенному графу уже можно визуально (или «водя пальцем») быстро найти нужный ответ.

Заметим, что время, затраченное на построение графа, сравнимо со временем построения дерева алгоритма Дейкстры. Однако, алгоритм Дейкстры даёт гораздо более надёжный результат.

Также выделим, что зачастую при решении задачи у вас есть возможность делать заметки прямо на условии (как, например, на экзамене, к которому мы с вами готовимся).

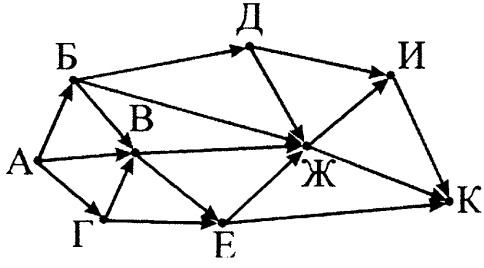
В этом случае не нужно заранее выписывать список доступных вершин. Удобнее использовать сразу

верхнюю строчку таблицы расстояний, где все вершины сразу перечислены. Вычеркивание вершин рекомендуется делать сразу в этой таблице. Это экономит немного времени. Ещё это удобно тем, что при поиске в таблице расстояний тех вершин, которые смежны с текущей и не являются уже вычеркнутыми, сразу видны вычеркнутые вершины.

В нашем примере это были шаги алгоритма 3, 6, 10, 14.

Количество путей на графе

Задача 2. На рисунке — схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И и К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?



Решение

Традиционный на первый взгляд метод решения этой задачи — водить пальцем по схеме от города к городу и подсчитывать, сколькими различными способами это можно сделать.

Этот способ по надёжности не выдерживает никакой критики — пропустить один или несколько вариантов очень легко, а проверить себя при этом практически невозможно.

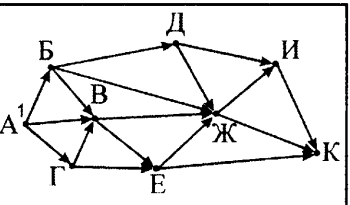
Для решения воспользуемся быстрым и эффективным способом решения под названием **динамическое программирование**. Пусть вас не смущает слово программирование. Никакого программирования при этом не будет. Просто способ используется в программировании при решении ряда подобных или похожих по принципу решения задач. Идея метода динамического программирования состоит в том, чтобы постепенно, шаг за шагом, получать частичные результаты для каждой рассматриваемой вершины. В результате работы метода на последнем шаге мы получим ответ для искомой вершины. В данном случае интересующий нас результат — это сколькими различными способами можно «доехать» из стартового города до текущей вершины (города).

Будем последовательно искать, сколькими различными способами можно добраться из города А до каждого из городов, нарисованных на схеме.

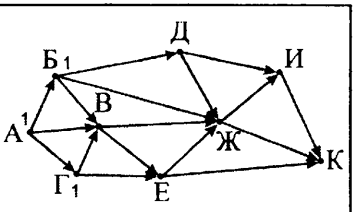
Прежде всего, для выполнения нужного нам решения необходимо понять, сколькими способами можно добраться из города А до самого города А.

Важно понимать, что это возможно осуществить только одним способом. Не нулём, как могло бы показаться поначалу, а именно одним способом — никуда не уезжая.

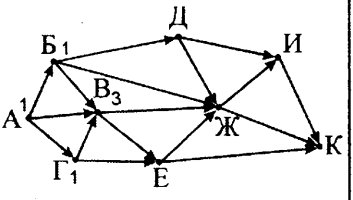
Пометим вершину А числом 1 (количество способов, которым можно добраться до данной вершины).

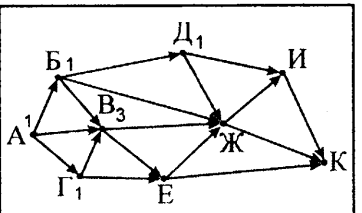
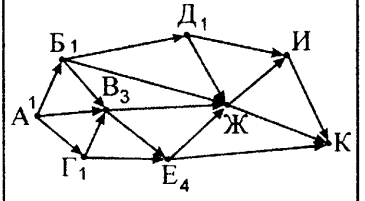
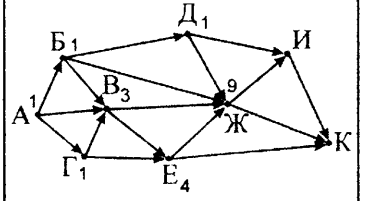


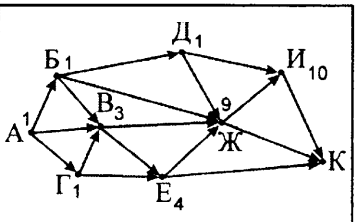
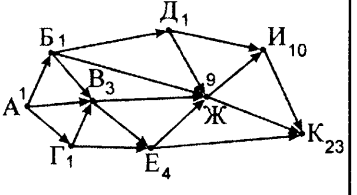
Теперь постараемся пометить какую-нибудь ещё вершину.
Это можно сделать только для тех вершин, про которые известно нужное количество всех стрелок, входящих в данную вершину.
На данном этапе нам известно это нужное количество только для вершины — А. Значит, будем искать вершины, в которые входит только одна стрелка дороги — из вершины А.
Таких городов на данной схеме два — Б и Г.
Пометим их числами 1. Это количество способов добраться из города А в города Б и Г.



Опять будем искать город, про который для всех стрелок, входящих в него, на схеме уже написаны числа. Один из таких городов — город В. В него входит три стрелки. Для каждой из стрелок (из городов А, Б, Г) уже написаны числа. Нужное число для текущего города В — это сумма чисел на начальных сторонах стрелок.
То есть, в город В можно прийти только по трём дорогам-стрелкам из города А, Б или Г. В каждый из них можно прийти одним способом. Значит, в город В можно прийти тремя способами. Пометим город В числом 3.



Найдём следующий город, для которого известны числа на началах всех входящих в него стрелках. Один из таких городов — Д. В него входит только одна стрелка из города Б. Значит, в город Д можно добраться столькими же способами — одним. Пометим город Д числом 1.	
Найдём следующий город, про который известны числа из всех сходящих стрелок. На данном этапе это только один город — Е. В него входит две стрелки-дороги (В и Г). Из города В — 3 способа, из города Г — 1 способ. Всего 4 способа. Пометим город Е числом 4.	
Ищем следующий город. На данном этапе это город Ж. В него входит 4 дороги. Про все эти стрелки-дороги известны числа для городов, из которых исходят стрелки. Это города Д, Б, В, Е. Рядом с ними написаны числа 1, 1, 3, 4. Сумма чисел — 9. Это количество способов (9) подписываем рядом с городом Ж.	

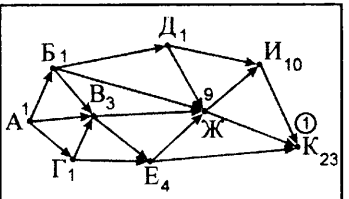
Ищем следующий город. Это город И. В него входит две дороги — из города Д и города Ж. Рядом с ними написаны числа 1 и 9. Их сумма — 10. Это количество напишем рядом с городом И.	
Последний город, для которого осталось посчитать нужное количество способов — город К. В него входит три стрелки-дороги (из городов И, Ж, Е). Рядом с ними написаны числа 10, 9, 4. Их сумма — 23. Это количество пишем рядом с городом К. Это и является ответом.	

Приведённый алгоритм простой и эффективный. Однако при его выполнении можно легко сделать ошибку в арифметических вычислениях или случайно пропустить какую-нибудь стрелку, входящую в вершину. Необходимо сделать проверку.

Выполним тот же алгоритм в обратную сторону. То есть, теперь будем искать количество способов, которыми можно добраться из каждого города, обозначенного на схеме, до города К.

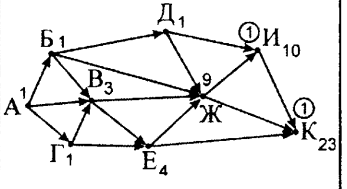
Прежде всего, обозначим количество способов, которым можно добраться из вершины К в вершину К. Как и ранее, это количество равно 1.

Подпишем это число (1) возле города К. Но, чтобы не перепутать числа обратного алгоритма с числами прямого алгоритма, будем обводить кружком числа обратного алгоритма.

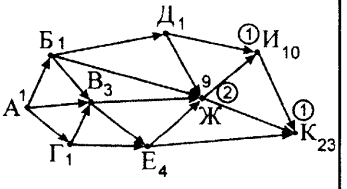


В прямом алгоритме мы искали все города, для которых были известны числа на началах всех входящих стрелок. Теперь будем искать города, для которых известны числа для всех исходящих стрелок.

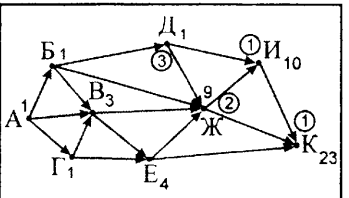
На данном шаге нас устроят только города, из которых выходит только одна стрелка — в город К. Такой город один — город И. Подпишем рядом с ним число 1 в кружке.



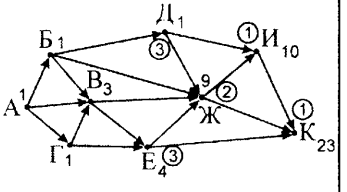
Теперь будем искать города, из которых все исходящие стрелки ведут в города с кружочками. Такая вершина одна — Ж. Из неё идут две стрелки — в города И и К. Рядом с ними в кружках написаны числа 1 и 1. Их сумма — 2. Подпишем рядом с вершиной Ж число 2 в кружке.



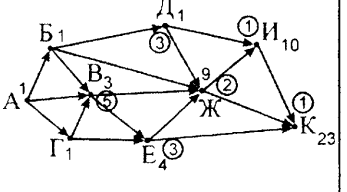
Ищем следующий город.
Один из городов, у которого каждая исходящая стрелка приходит в город с кружком — город Д. Из него выходит две стрелки (в города Ж и И). Числа на концах стрелок — 2 и 1. Их сумма — 3.
Пишем число 3 в кружке рядом с городом Д.



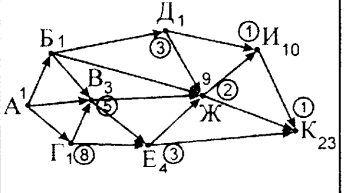
Следующий город — город Е.
Из него выходит две стрелки (в города Ж и К). Числа в кружках на концах стрелок — 2 и 1. Их сумма — 3.
Пишем число 3 в кружке рядом с городом Е.

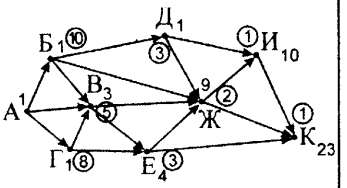
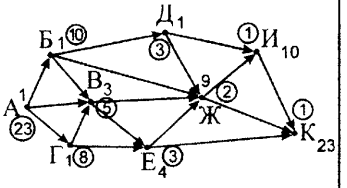


Теперь можно посчитать число вариантов для города В.
Из него выходит две стрелки (в города Е и Ж). Числа в кружках на концах стрелок — 3 и 2. Их сумма — 5.
Пишем число 5 в кружке возле города В.



Теперь можно посчитать число вариантов для городов Б и Г.
Начнём, например, с города Г.
Из него выходит две стрелки (в города В и Е). Числа в кружках на концах стрелок — 5 и 3. Их сумма — 8. Пишем число 8 в кружке возле города Г.



Теперь посчитаем число вариантов для города Б. Из него выходит три стрелки-дороги (в города В, Ж, Д). Числа в кружках на концах стрелок — 5, 2, 3. Их сумма — 10. Пишем число 10 в кружке возле города Б.	
Наконец, можно посчитать число вариантов для города А. Из него выходит три стрелки-дороги (в города Б, В, Г). Числа в кружках на концах стрелок — 10, 5, 8. Их сумма — 23. Пишем число в кружке возле города А. Это ответ. Результаты прямого и обратного алгоритма совпали. Значит, наш ответ — правильный.	

Настоятельно рекомендуем при решении этой задачи выполнять оба алгоритма — прямой и обратный — и сверять результаты их работы. Если результаты не совпадут — искать ошибку в обоих алгоритмах. Начинать искать ошибку следует с того алгоритма, результат которого дал меньшее число. Эта рекомендация дана на основании наиболее частой ошибки — забыть добавить в сумму какое-нибудь из чисел или не заметить одну из стрелок. Поэтому, как правило, меньший результат — неверный. Это не значит, что нужно сразу принять за верный ответ большее из двух результатов. Просто рекомендуется начать поиск ошибки с меньшего результата. Для принятия решения о том, что найденный ответ — верный, результаты двух алгоритмов должны совпасть.

9 Математические инструменты, динамические (электронные) таблицы

Электронные таблицы

Конспект

Данный раздел не ставит своей целью ознакомить учащихся со всеми возможностями электронных таблиц. К сожалению, это занимает достаточно много времени. Мы коснёмся только основных моментов, которые необходимы, чтобы получить общее представление об электронных таблицах и выполнить задания экзамена. Кое-где будем сознательно опускать тонкости и подробности, мешающие пониманию основных моментов, иногда допуская упрощённые определения и сведения.

В основе электронной таблицы лежат понятия таблицы и ячейки. **Таблица**, как вы прекрасно понимаете, это расчерченное горизонтальными и вертикальными линиями пространство. Горизонтальными линиями отделяются друг от друга **строки**, вертикальными линиями — **столбцы**. На пересечении строк и столбцов находятся **ячейки**. В современных электронных таблицах количество строк таблицы ограничено обычно примерно одним миллионом ($= 2^{20}$), а количество строк — 16-ю тысячами ($= 2^{14}$).

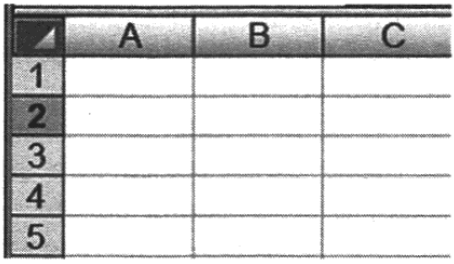
Заметим, что в электронных таблицах документ, как правило, состоит из нескольких таблиц, называ-

емых **листами**. Листы объединяются в **книгу**. Книга как раз и является документом с точки зрения электронной таблицы. Не будем подробно останавливаться на листах. Нам вполне хватит того, что предоставляет одна таблица (лист).

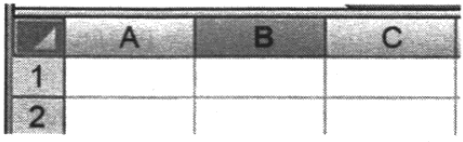
В каждой ячейке таблицы могут храниться такие основные данные, как текст, число, дата, время, формула.

Основное отличие электронной таблицы от обычной таблицы, например, в текстовом редакторе, состоит в наличии удобного способа использования формул.

Чтобы в формулах было удобнее обращаться к ячейкам, каждая строка таблицы имеет собственный **номер строки**. Номера строк можно видеть слева от каждой строки.



Каждый столбец таблицы также имеет собственный номер. Но чтобы не путать номера строк и номера столбцов, номера столбцов записываются буквами латинского алфавита по порядку. Эти буквы-номера вы можете видеть в верхней части таблицы, над каждым столбцом.



Вероятно, у вас возникает вопрос, как же можно пронумеровать 16 тысяч столбцов, используя всего

26 букв латинского алфавита? Это решается примерно таким же способом, как и в любой позиционной системе счисления. После того, как «буквы заканчиваются» (это в первый раз происходит после 26-го столбца, на букве Z), нумерация снова начинается с буквы A, но в левый разряд «прибавляется» очередная буква. То есть, после столбца Z следуют столбцы AA, AB, AC, ... и продолжается до столбца AZ, после которого следуют BA, BB, BC, ..., BZ, CA, CB, ..., ZZ, AAA, AAB, и так далее. Впрочем, для учебных задач вполне хватает первых 26-ти столбов от A до Z, поэтому вы в большинстве случаев можете считать, что номер столбца — это просто буква латинского алфавита.

Каждая ячейка электронной таблицы имеет собственный **адрес ячейки**. Он состоит из номера столбца и номера строки, на пересечении которых находится ячейка. Например, на пересечении строки 2 и столбца B находится ячейка B2.

Как мы уже писали, в каждой ячейке может храниться текст, число, дата, время или формула. Нам важно понимать, что формулы обрабатывают именно ту информацию, которая записывается в ячейки (чаще всего — числа). Когда в таблице изменяются какие-нибудь ячейки, значения формул автоматически пересчитываются.

Каждая формула в электронной таблице начинается с символа «=». Самая простая формула, которая может быть записана, — это «=число». Например, «=45» (кавычки в ячейку не пишутся; мы приводим их для отделения формулы от текста раздела). Ячейка, которая содержит формулу, имеет также значение, вычисляемое по этой формуле. В приведённом примере ячейка содержит формулу «=45» и имеет значение 45.

Можно записать в формуле выражение. Например, «=14+53». Результатом этой формулы будет, есте-

ственно, число 67. Такие формулы не очень интересно использовать.

Эта же технология позволяет сделать значением ячейки тем же, что уже является значением другой ячейки. Для этого необходимо записать «=адрес ячейки». Например, в ячейки D8 написана формула «=F4». Значение ячейки D8 будет одинаковым со значение ячейки F4. Если изменить значение ячейки F4, автоматически изменится значение ячейки D8.

Рассмотрим пример таблицы:

	A	B	C
1	10	30	
2	20	40	

Запишем в ячейку C2 формулу, которая вычисляет сумму ячеек A1 и B2. Это достаточно просто: «=A1+B2».

В данном случае в ячейке будет храниться формула «=A1+B2», а при просмотре таблицы в ячейке C2 будет отображаться её значение: 50.

	A	B	C
1	10	30	
2	20	40	50

В электронных таблицах, кроме приведённой операции сложения, допустимы также и другие арифметические операции:

– вычитание (знак «-»),
– умножение (знак «*»),
– деление (знак «/»).

Простыми арифметическими операциями электронные таблицы не ограничиваются. Они имеют ещё значительное количество специальных функций, осо-

бым образом обрабатывающих значения ячеек. Названия этих функций зависят от того, какой электронной таблицей вы пользуетесь.

В зависимости от того, насколько русифицирована электронная таблица, формулы могут быть записаны русскими или английскими названиями. Имена функций принято писать заглавными буквами. Например, СУММ() или SUM(). Чаще всего, это сокращения от названий соответствующих действий. Приведённое название функции суммирования, как видите, сокращено до нескольких первых букв.

Если у функции требуется использовать несколько аргументов, их следует перечислять через точку с запятой.

Обычно в электронной таблице требуется обрабатывать достаточно большие объёмы данных. Применять для сложения, например тысячи ячеек, операцию «+» достаточно утомительно (нужно привести адреса тысячи ячеек между ними поставить 999 значков «+»). К тому же, при обработке больших объёмов данных эти данные расположены в соседних относительно друг друга ячейках.

Для упорядочивания обработки смежных ячеек в электронных таблицах и используется понятие **диапазона ячеек**. Диапазон ячеек — это прямоугольная область смежных ячеек, задаваемая адресами противоположных углов прямоугольной области, разделённых двоеточием.

Как правило, указывают адрес левой верхней и правой нижней ячеек диапазона. Например, диапазон F4:H5 — это прямоугольная область, состоящая из шести ячеек (три столбца — F, G и H и две строки — 4 и 5). Адреса диапазонов используются в формулах, обобщающих значения ячеек диапазона.

Ниже в таблице представлен список наиболее часто используемых формул.

Русское обозначение	Английское обозначение	Смысл функции
СУММ	SUM	Сумма всех непустых и нетекстовых значений указанных ячеек
СЧЁТ	COUNT	Количество всех непустых и нетекстовых значений указанных ячеек
СРЗНАЧ	AVERAGE	Среднее арифметическое всех непустых и нетекстовых значений указанных ячеек
МАКС	MAX	Наибольшее значение среди всех непустых и нетекстовых значений указанных ячеек
МИН	MIN	Наименьшее значение среди всех непустых и нетекстовых значений указанных ячеек
ЕСЛИ	IF	Проверка условия. Содержит три аргумента. Первый аргумент должен быть логическим выражением. Если значение первого аргумента — истина, то значением функции является второй аргумент. Если ложно — третий аргумент. Если не указано значение того (второго или третьего) аргумента, который должен являться результатом, то значение функции ЕСЛИ становится пустым.

Русское обозначение	Английское обозначение	Смысл функции
СЧЁТЕСЛИ	COUNTIF	Количество непустых и нетекстовых ячеек в диапазоне, удовлетворяющих заданному условию. Первый аргумент — диапазон проверяемых ячеек. Второй аргумент — условие, пишется в кавычках.
СУММЕСЛИ	SUMIF	Сумма непустых и нетекстовых ячеек в диапазоне, удовлетворяющих заданному условию. Первый аргумент — диапазон проверяемых ячеек. Второй аргумент — условие, пишется в кавычках.

 В качестве логического выражения функции ЕСЛИ (IF) можно использовать числовое значение. Нулевое значение считается ложным, остальные значения — истинными. Мы не рекомендуем использовать эту возможность.

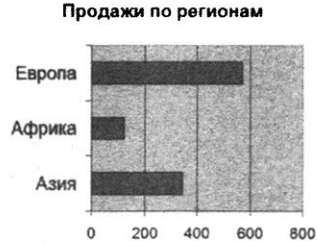
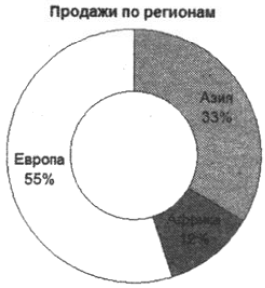
По значениям ячеек электронной таблицы могут быть построены различные **диаграммы**.

Как правило, диаграммы наглядно отображают зависимости между значениями ячеек таблицы. При построении диаграммы указывается диапазон ячеек,

по которому диаграмма строится и выбирается вид диаграммы.

Например, для отображения абсолютной числовой зависимости между значениями ячеек применяются **линейчатая диаграмма** и **гистограмма**. А для отображения долевой зависимости значений ячеек в общей сумме используются **кольцевая** и **круговая** диаграммы.

Примеры диаграмм:

Гистограмма	**Линейчатая**
	
Кольцевая	**Круговая**
	

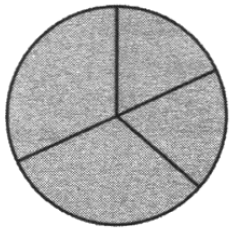
При построении диаграмм (и при решении задач на диаграммы) важно понимать, что они строятся по значениям диапазона ячеек. Эти значения на диаграмме отображаются в том же порядке, что и в диапазоне.

Разбор типовых задач

Задача 1. Дан фрагмент электронной таблицы.

	A	B	C	D
1	3	4	2	5
2		=D1-1	=A1+B1	=C1+D1

Какая из формул, приведённых ниже, может быть записана в ячейке A2, чтобы построенная после выполнения вычислений диаграмма по значениям диапазона ячеек A2:D2 соответствовала рисунку?



1) =D1-A1 **2)** =B1/C1 **3)** =D1-C1+1 **4)** =B1*4

Решение

Для построения диаграммы нужны значения всех ячеек диапазона, по которым диаграмма строится, то есть, значения всех ячеек A2, B2, C2 и D2.

Сначала вычислим значения тех ячеек, формулы для которых известны.

B2=D1-1=5-1=4

C2=A1+B1=3+4=7

D2=C1+D1=2+5=7

Получаем, что диаграмма построена по значениям:

неизвестно	4	7	7

Анализируем диаграмму. Это круговая диаграмма. На ней значение каждой ячейки диапазона, по которому строится диаграмма, отображается в виде секто-

ра (доли) такого размера, сколько его значение «вносит» в общую сумму значений ячеек этого диапазона. На диаграмме мы видим две пары секторов одинакового размера. Одна пара имеет значения побольше, другая — поменьше. Анализируем значения, которые нам известны. Это 4, 7 и 7. Семь больше четырёх. Понимаем, что обе семёрки — это значения, которые побольше. Значит, оставшееся значение — 4 — это значение долей/ячеек, которые поменьше. Следовательно, оставшаяся (неизвестная) ячейка имеет значение также 4.

Вычислим значения предлагаемых нам в качестве ответа формул и найдём среди них ту формулу, значение которой равно 4.

Для дополнительной проверки рекомендуется вычислить значения всех четырёх предлагаемых формул, а не останавливаться, если значение одной из них оказалось тем, которое искали (в нашем случае, 4). Ведь, если значения нескольких формул в варианте ответа будут содержать одинаковый (и нужный нам) ответ, то это станет поводом задуматься о возможной ошибке в примере.

Итак, вычисляем значения формул:

1) `=D1-A1` `=5-3=2`
2) `=B1/C1` `=4/2=2`
3) `=D1-C1+1` `=5-2+1=4` Это значение мы ищем.
4) `=B1*4` `=4*4=16`

Ответ: 3.

Теперь постараемся обсудить те навыки, которые нужны для выполнения практического задания по электронным таблицам. Сделаем это на примере электронной таблицы MS Excel. Если вы привыкли пользоваться другим пакетом (например, OpenOffice.org Calc или LibreOffice Calc), то эти отличия несущественны. Разница, в основном, только именах используемых

функций. В русских версиях MS Excel имена функций русифицированы, а в Calc — нет. Заметим, что английские имена функций удобнее в использовании, потому что имена ячеек требуется всё равно писать латинскими буквами и при русских именах функций приходится постоянно переключаться с одного языка на другой.

В практическом задании на экзамене даётся готовый файл, содержащий много (обычно около тысячи) строк с информацией об одном объекте в каждой строке.

Задача 2. В электронную таблицу занесли данные о калорийности продуктов. Ниже приведены первые пять строк таблицы.

	A	B	C	D	E
1	**Продукт**	**Жиры, г**	**Белки, г**	**Углеводы, г**	**Калорийность, Ккал**
2	Арахис	45,2	26,3	9,9	552
3	Арахис жареный	52	26	13,4	626
4	Горох отварной	0,8	10,5	20,4	130
5	Горошек зелёный	0,2	5	8,3	55

В столбце A записан продукт; в столбце B — содержание в нём жиров; в столбце C — содержание белков; в столбце D — содержание углеводов и в столбце E — калорийность этого продукта.

Всего в электронную таблицу были занесены данные по 1000 продуктам.

Выполните задание.

Откройте файл с данной электронной таблицей (возьмите файл из Демоверсии экзамена за 2016 год на

сайте ФИПИ). На основании данных, содержащихся в этой таблице, ответьте на два вопроса.

1. Сколько продуктов в таблице содержат меньше 50 г углеводов и меньше 50 г белков? Запишите число, обозначающее количество этих продуктов, в ячейку H2 таблицы.

2. Какова средняя калорийность продуктов с содержанием жиров менее 1 г? Запишите значение в ячейку H3 таблицы с точностью не менее двух знаков после запятой.

Полученную таблицу необходимо сохранить под именем, указанным организаторами экзамена.

Решение

По данному файлу нужно вычислить некоторую информацию и поместить результат в указанные ячейки.

Это задание (и подобные ему) можно выполнить массой всевозможных способов. Мы обсудим только некоторые из них, наиболее удобные, с нашей точки зрения.

Сначала найдём ответ на первый вопрос.

Выберем среди всех строк те, которые удовлетворяют приведённому условию. Условие, которое нужно проверить (меньше 50 г углеводов и меньше 50 г белков) — сложное, состоящее из двух условий, объединённых союзом И, т. е. логической операцией И.

Данное условие можно проверить несколькими способами.

Способ первый. С использованием вспомогательных столбцов.

Так как при решении нам понадобится несколько вспомогательных столбцов, а ответ нужно поместить в ячейку H2, будем строить эти столбцы после столбца H.

Условие про каждую строку будем записывать в ней же.

В столбце I проверим первое условие (меньше 50 г углеводов).

Для этого запишем в ячейку I2 (проверка условия для строки 2) формулу «=ЕСЛИ(D2<50;1;0)». (Не забудем после написания формулы нажать на [Enter].) В этой формуле проверяем ячейку D2 — именно она хранит информацию о количестве углеводов для продукта строки 2, которое должно быть меньше 50 грамм. В заголовке столбца D написано, что это количество углеводов и что оно приведено в граммах. То есть, нам не нужно указывать граммы при сравнении или преобразовывать величины. Углеводы даны в граммах и нам нужно сравнить их с 50-ю граммами. Поэтому мы и написали условие «D2<50». Вторым и третьим аргументами мы написали 1 и 0. То есть, если условие «углеводов меньше 50 грамм» выполнится, в ячейке будет записано 1. А если не выполнится — 0. Это (1 и 0) очень удобные обозначения при проверке логических выражений. Мы рекомендуем вам пользоваться именно ими.

На данный момент мы проверили условие только для одной строки. Теперь следует скопировать эту формулу во все нижележащие 999 ячеек. Самый простой (но не самый быстрый и удобный) способ это сделать — воспользоваться специальным **маркером заполнения**. Выделите ячейку I2, куда мы только что записали указанную формулу. Вокруг выделенной ячейки будет нарисована толстая прямоугольная граница, показывающая, что именно эта ячейка выделена. В правом нижнем углу границы увидите чёрный квадратик. Это и есть маркер заполнения. Наведите на него курсор мыши. Форма курсора мыши изменится. Теперь курсор мыши будет иметь форму тонкого черного плюсика. Нажмите левую кнопку мыши и, не отпуская её, начните тянуть мышь вниз. Тяните мышь вниз, ниже видимой части таблицы, «наехав» мышью на нижнюю часть интерфейса электронной таблицы. Видимая часть таблицы начнёт «уезжать» вверх, переходя все

дальше и дальше ко всё более нижним ячейкам таблицы. Таким образом следует «доехать», не отпуская левую кнопку мыши, до последней, 1001-й строки таблицы. Все ячейки, сквозь которые вы «протянули» таким образом ячейку I2 за маркер заполнения, окажутся заполненными формулой «=ЕСЛИ(...<50;1;0)». Прелесть ситуации состоит также в том, что для всех этих ячеек формула будет не та, которая была изначально в ячейке I2 («=ЕСЛИ(D2<50;1;0)»), а измененная в соответствии с текущей строкой. Благодаря технологии автоиндексации адреса, электронная таблица сама изменит номер строки на требуемый. То есть, в строке 3 будет проверяться уже ячейка D3, а в строке 4 — ячейка D4, и т.д.

Неудобство данного метода состоит в том, что при быстром «проматывании» маркером заполнения до необходимой строки, легко «проехать мимо» нужной строки с номером 1001 и случайно заполнить ещё несколько ячеек (несколько десятков ячеек). Тогда следует либо удалить информацию из лишних заполненных ячеек, либо аккуратно учитывать это, применяя результаты (то есть, не учитывать то, что находится в строках ниже 1001-й).

Другой способ заполнить все ячейки от I3 до I1001 формулой, находящейся в ячейке I2, более быстрый, но работающий только в MS Excel.

Выделите ячейку I2. Теперь, при помощи вертикальной полосы прокрутки, переместите просматриваемую часть таблицы так, чтобы на экране была вида ячейка I1001. Для этого потяните мышью вниз ползунок на вертикальной полосе прокрутки до самого нижнего положения.

Теперь, удерживая кнопку [Shift] на клавиатуре, щелкните мышью по ячейке I1001. В результате такого действия будет выделен весь диапазон ячеек от ячейки I2 до ячейки I1001. А именно, все те ячейки,

в которые нужно поместить формулу, которая уже находится в ячейке I2. Заметим, что текущей ячейкой всё ещё является ячейка I2, в которой находится нужная нам формула.

Перейдём в режим редактирования текущей ячейки (I2). Для этого нажмём один раз кнопку [F2] на клавиатуре. Вертикальный моргающий курсор клавиатуры появится в конце формулы ячейки I2. Теперь нажмём на клавиатуре комбинацию клавиш [Ctrl]+[Enter]. В результате этого действия формула из текущей ячейки (I2) заполнится во все выделенные ячейки. Этого мы и добиваемся.

Итак, мы проверили первое условие (углеводы меньше 50 грамм) для каждой строки со 2-й по 1001-ю. Точно так же проверим второе условие — белки меньше 50 грамм — в столбце J. Введём в ячейку J2 формулу «=ЕСЛИ(C2<50;1;0)» и заполним этой формулой, одним из приведённых способов, ячейки от J3 до J1001.

Теперь проверим выполнение общего условия — что одновременно углеводов меньше 50 грамм и белков меньше 50 грамм. Для этого просто перемножим в каждой строке полученные в столбцах I и J цифры, т. е. запишем в ячейку K2 формулу «=I2*J2» и заполним ею ячейки от K3 до K1001. Данный способ даст нам в каждой строке число 1, если и углеводов меньше 50 г, и белков меньше 50 г.

Действительно, чтобы выполнилось условие **И**, необходимо, чтобы оба аргумента были равны 1. Умножение двух чисел, каждое из которых — 0 или 1, даёт 1 только при двух единицах.

Остаётся только посчитать, сколько в столбце K получилось единиц. Для этого достаточно просто сложить все числа в столбце K от ячейки K2 до ячейки K1001. Запишем в требуемую ячейку H2 формулу «=СУММ(K2:K1001)».

Другой способ получить нужные 1 и 0 в каждой строке для проверки сложного условия «углеводы меньше 50 грамм **И** белки меньше 50 грамм» — воспользоваться встроенной логической функцией **И** (AND). В её аргументах следует указывать логические выражения. Результат функции — ИСТИНА, если все логические выражения ИСТИНА, и ЛОЖЬ, если хотя бы одно из них ложно.

Так как при использовании данного способа требуется меньше вспомогательных ячеек, воспользуемся столбцом F, как это предполагает разработчик задания.

Запишем в ячейку F2 формулу, которая сразу положит в ячейку F2 значение 1, если требуемое сложное условие истинно, и 0, если оно ложно. Используем функцию ЕСЛИ, в которой в первом аргументе через функцию И проверим оба нужные нам условия — и про углеводы, и про белки.

То есть, формула будет такой:
«=ЕСЛИ(И(D2<50;C2<50);1;0)».

Эту формулу одним из двух уже описанных выше способов скопируем/заполним в ячейки от F3 до F1001.

Осталось только в требуемой ячейке H2 записать формулу, складывающую все значения диапазона F2:F1001, т. е.: «=СУММ(F2:F1001)».

Нахождение ответа на второй вопрос можно осуществить тем же образом, построив вспомогательный столбец.

Нам теперь необходимо посчитать не количество ячеек, а среднее арифметическое их значений. Поэтому результатом проверки условия ЕСЛИ должны стать не 0 и 1, а те значения, по которым нужно вычислять среднее арифметическое. То есть, значение калорийности в случае истинности условия «содержание жиров менее 1 грамма». А вот при ложности

данного условия очень важно, чтобы значение строки не подсчитывалось функцией «СРЗНАЧ». Функция СРЗНАЧ не учитывает пустые ячейки и текстовые, значит, при ложности условия следует положить в ячейку пустое значение или текстовое. Например, « ».

Получаем формулу для ячейки G2:
«=ЕСЛИ(B2<1;E2;« »)».

Скопируем/заполним эту формулу в ячейки G3:G1001. Теперь достаточно в требуемую ячейку H3 записать формулу, вычисляющую среднее арифметическое значение ячеек G2:G1001:
«=СРЗНАЧ(G2:G1001)».

Другой способ выполнения задания — использовать специальные сложные функции СУММЕСЛИ и СЧЕТЕСЛИ.

Чтобы вычислить среднее арифметическое калорийности ячеек, у которых жиров меньше 1 грамма, нужно сумму калорийностей таких ячеек поделить на количество таких ячеек. Сумму с условием можно вычислить при помощи функции СУММЕСЛИ: «=СУММЕСЛИ(B2:B1001;"<1";E2:E1001)». То есть, проверяем ячейки диапазона B2:B1001 на условие «<1» и складываем те соответствующие им ячейки диапазона E2:E1001, для которых это условие выполнилось.

Количество определяем при помощи функции СЧЕТЕСЛИ: «=СЧЕТЕСЛИ(B2:B1001;"<1")».

Остаётся только поделить результат первой функции на результат второй функции, записав в требуемую ячейку H3 формулу:
«=СУММЕСЛИ(B2:B1001;"<1";E2:E1001)/СЧЕТЕСЛИ(B2:B1001;"<1")».

Способ со вспомогательными ячейками нравится нам больше как более универсальный.

Не для каждой задачи всегда удаётся подобрать такую формулу, которая сразу даст нужный результат.

А при использовании некоторого количества вспомогательных столбцов можно добиться вычисления весьма хитрых формул.

Задача 3. В электронную таблицу занесли информацию о грузоперевозках, совершенных некоторым автопредприятием с 1 по 9 октября. Ниже приведены первые пять строк таблицы.

	A	B	C	D	E	F
1	Дата	Пункт отправления	Пункт назначения	Расстояние	Расход бензина	Масса груза
2	1 октября	Липки	Берёзки	432	63	600
3	1 октября	Орехово	Дубки	121	17	540
4	1 октября	Осинки	Вязово	333	47	990
5	1 октября	Липки	Вязово	384	54	860

Каждая строка таблицы содержит запись об одной перевозке.

В столбце A записана дата перевозки (от «1 октября» до «9 октября»); в столбце B — название населённого пункта отправления перевозки; в столбце C — название населённого пункта назначения перевозки; в столбце D — расстояние, на которое была осуществлена перевозка (в километрах); в столбце E — расход бензина на всю перевозку (в литрах); в столбце F — масса перевезенного груза (в килограммах).

Всего в электронную таблицу были занесены данные по 370 перевозкам в хронологическом порядке.

Выполните задание.

Откройте файл с данной электронной таблицей (возьмите файл из Демоверсии экзамена за 2013 год на сайте ФИПИ). На основании данных, содержащихся в этой таблице, ответьте на два вопроса.

1. На какое суммарное расстояние были произведены перевозки с 1 по 3 октября? Запишите число, обозначающее это суммарное расстояние, в ячейку H2 таблицы.

2. Какова средняя масса груза при автоперевозках, осуществлённых из города Липки? Запишите значение в ячейку H3 таблицы с точностью не менее двух знаков после запятой.

Не забудьте сохранить полученную таблицу.

Решение

Ответ на первый вопрос можно получить несколькими способами. Рассмотрим сначала более универсальный.

Построим вспомогательный столбец I, в котором будем записывать расстояние для тех перевозок, которые были осуществлены с 1 по 3 октября.

Так как по условию задачи перевозки осуществлялись с 1 по 9 октября, будет достаточным проверить, что дата перевозки не позже 3 октября. То есть, проверяемое условие — A2<="3 октября".

Заметим, что дата, используемая в столбце A с датами перевозок, представлена в текстовом формате. Поэтому при сравнении значения ячейки A2 с датой "3 октября" мы используем двойные кавычки, понимая при этом, что сравнение будет происходить в текстовом виде. А именно, будут сравниваться сначала первые символы, при их совпадении — вторые, и т. д. Если бы диапазон дат по условию не ограничивался 9-м октября, пришлось бы писать гораздо более сложное сравнение.

Значение, которое необходимо записать в столбце I, если условие выполнится — это соответствующее расстояние, т. е. D2. Если условие не выполнится, данное расстояние не должно попасть в сумму. Тогда запишем в столбец I ноль. Получаем формулу для ячейки I2: «=ЕСЛИ(A2<="3 октября";D2;0)». Скопируем/заполним эту формулу в ячейки I3:I371.

Теперь посчитаем сумму всех этих ячеек в требуемой ячейке H2: «=СУММ(I2:I371)».

Второй вариант решения задания — воспользоваться сразу формулой, считающей сумму с условием.

Запишем в требуемую ячейку H2 формулу: «=СУММЕСЛИ(A2:A371;"<='3 октября'";D2:D371)».

Заметим, что при указании условия сравнения дату '3 октября' пришлось указать в апострофах (одинарных кавычках), чтобы электронная таблица правильно поняла проверяемое условие, потому что проверяемое условие функции СУММЕСЛИ само должно быть записано в двойных кавычках.

Третий вариант решения. По условию дано, что строки отсортированы по дате перевозки от 1 до 9 октября, все нужные нам строки находятся в первой части таблицы. Поэтому просто найдём, до какой строки продолжаются перевозки с датой 3 октября.

Пролистываем таблицу вниз и находим, что последняя строка с датой «3 октября» — номер 118. Значит, достаточно посчитать сумму расстояний для строк со 2-й по 118-ю. Записываем в требуемую ячейку H2 формулу: «=СУММ(D2:D118)».

Есть ещё более простой и наименеё универсальный способ решения именно этой задачи.

Так как мы знаем, что строки отсортированы по дате и можем найти номер строки (118), которой заканчиваются интересующие нас перевозки, мы можем просто выделить диапазон ячеек от ячейки D2 до ячейки D118 и в нижней части интерфейса электронной таблицы посмотреть на значение суммы выделенных

ячеек. Это значение можно просто вписать руками в требуемую ячейку H2. Такое решение «законно» и засчитывается проверяющими экспертами. Однако, из-за его малой универсальности мы советуем, всё же, освоить универсальное решение и не надеяться, что на экзамене попадётся настолько простая задача.

Найдём ответ на второй вопрос задачи. Для вычисления среднего арифметического значений диапозона ячеек достаточно найти сумму требуемых ячеек и поделить её на количество этих ячеек.

Один из вариантов решения — построить вспомогательные столбцы для нахождения суммы нужных ячеек и для нахождения их количества. Потом сумму по первому из этих столбцов поделить на сумму второго из этих столбцов.

Запишем в ячейку J2 формулу, выбирающую массу груза, если перевозка отправляется из пункта «Липки» или ноль, в противном случае. Для этого в J2 укажем формулу: «=ЕСЛИ(B2="Липки";F2;0)». Скопируем/заполним эту формулу в ячейки J3:J371.

Теперь запишем в ячейку K2 формулу, проверяющую, отправляется ли перевозка из пункта «Липки» и выдающая результат 1, если это условие истинно, и 0 — если условие ложно. Укажем в ячейку K2 формулу: «=ЕСЛИ(B2="Липки";1;0)».

Скопируем/заполним эту формулу в ячейки K3:K371.

Всё готово для итоговых вычислений. Запишем в требуемую ячейку H3 формулу:
«=СУММ(J2:J371)/СУММ(K2:K371)».

Другой (менее универсальный, но более короткий в данном случае) способ решения задачи — воспользоваться функциями СУММЕСЛИ и СЧЕТЕСЛИ. То есть, можно сразу записать в требуемую ячейку H3 формулу: «=СУММЕСЛИ(B2:B371;"Липки";F2:F371)/СЧЕТЕСЛИ (B2:B371;"Липки")».

10 Организация информационной среды

Сетевые технологии

Конспект

Как всем известно, **Интернет** — это всемирная компьютерная сеть. Множество компьютеров по всему миру объединено между собой и могут обмениваться различными данными.

Компьютер, который пытается обратиться к другому компьютеру, называется **клиентом**. Компьютер, к которому происходит обращение, называется **сервером**. Обычно сервером называют также компьютер, предназначенный для предоставления другим компьютерам в сети Интернет различной информации.

Чтобы компьютер мог отправлять и принимать данные в сети Интернет, ему необходимо иметь специальный адрес, который называется **ip-адрес**. Одна из самых распространенных версий ip-адреса — это четыре числа, разделённые точкой. Каждое число может принимать значение от 0 до 255.

В действительности, описываемая версия ip-адреса, так называемая IPv4, состоит из 32 бит (то есть, четырёх байт). При записи ip-адреса в более удобной для человека форме каждый байт записывается в десятичной системе счисления, и эти четыре числа принято разделять точками. Так как каждый байт (8 бит) может принимать значения от 0 до 255, то и получается восприятие ip-адреса как четырёх чисел, разделённых точками, где каждое число принимает значение от 0 до 255.

Чтобы один компьютер мог обратиться к другому в Сети Интернет, первому компьютеру необходимо знать ip-адрес второго компьютера. Однако, если к компьютеру люди обращаются достаточно часто, оказывается неудобным делать это, запоминая его ip-адрес. Людям удобнее запоминать информацию в виде слов.

Для этого удобства придумали ещё один способ адресации компьютеров в сети Интернет. К тем компьютерам, к которым люди будут обращаться часто, прикрепляют специальное символьное название, называемое **доменный адрес** компьютера, — несколько коротких слов (минимум два), разделённых точками. Например, **mail.ru**.

Доменные адреса компьютеров создаются по некоторым правилам. Обычно используется **территориальный способ** или **по принадлежности организации к определённому виду**.

Пример территориального способа создания доменного адреса: **www.sch239.spb.ru**.

Такой адрес нужно разбирать с конца (справа налево):
– **ru** означает, что компьютер находится в России.
– **spb** означает, что он находится в Санкт-Петербурге.
– **sch239** означает название организации, к которой принадлежит компьютер (в данном случае, школа 239).
– **www** означает название компьютера в этой организации, который отвечает на запросы из сети Интернет.

Другой способ выделения доменного адреса, в основном, используется компаниями, расположенными в США (на родине сети Интернет) или компаниями, которые позиционируют себя как международные.

Примеры выделения доменного адреса по принадлежности организации к определённому виду:

.com — коммерческие организации
Например, www.ford.com.

.edu — образовательные организации
Например, www.mit.edu

.gov — правительственные организации.
Например, www.whitehouse.gov

.mil — военные организации
Например, www.pentagon.mil

.org — общественные организации
Например, www.greenpeace.org

.net — организации, занятые сетевыми технологиями. Например, www.nordu.net

Важно понимать, что доменный адрес и ip-адрес — это не заменяющие друг друга способы адресации. Выделение компьютеру доменного адреса не отменяет необходимости ему иметь ip-адрес.

Когда один компьютер пытается обратиться к другому, используя доменный адрес, происходит следующий процесс.

Первый компьютер пытается узнать, какой ip-адрес соответствует требуемому доменному адресу. Для этого он обращается к специальному компьютеру в сети Интернет, называемому **DNS-сервер**.

DNS (domain name service — служба доменных имён) — это и есть та технология, которая позволяет людям использовать доменные имена компьютеров. DNS-сервер хранит информацию о нужном ip-адресе и возвращает его первому компьютеру. И уже по ip-адресу первый компьютер обращается затем к нужному компьютеру.

В действительности, DNS-сервер не всегда непосредственно хранит информацию о требуемом ip-адресе по доменному адресу.

В этом случае он либо возвращает обратно ip-адрес того DNS-сервера, к которому нужно обратиться, следующим за нужным ip-адресом, либо сам ищет нужную информацию у отвечающих за это других DNS-серверов.

На компьютерах, предоставляющих доступ к своим данным в сеть Интернет, обычно располагаются различного рода ресурсы. Чтобы указать, где именно в сети Интернет расположен тот или иной ресурс, используется **URL** (uniform resource locator — определитель местонахождения ресурса). Его можно считать полным адресом ресурса в сети Интернет.

URL состоит из трёх частей: название протокола:// адрес компьютера в сети Интернет/путь к файлу на этом компьютере.

Название протокола — это указание на то, каким способом будет осуществляться передача данных.

Протокол передачи данных — это соглашение между людьми о том, каким способом компьютеры или их части будут обмениваться информацией. Обычно в URL используются протоколы: **HTTP** (hypertext transfer protocol — протокол передачи гипертекста) и **FTP** (file transfer protocol — протокол передачи файлов).

Адрес компьютера в сети Интернет — это доменный или ip-адрес компьютера, на котором располагается нужный ресурс. Например, vasya.com.

Путь к файлу на компьютере — это указание на то, где на компьютере (от некоторой стартовой точки) располагается нужный файл-реурс.

Например, pictures/flower.jpg означает, что на этом компьютере файл flowers.jpg находится в папке pictures.

Общий пример URL для файла:

http://vasya.com/pictures/flowers.jpg.

Если нужный файл находится непосредственно в корне файловой системы, отображаемой компьютером в Интернет, то имя пишется сразу после символа /, отделяющего адрес компьютера от имени файла на компьютере.

Разбор типовых задач

Задача 1. Доступ к файлу rus.doc, находящемуся на сервере obr.org, осуществляется по протоколу https. Фрагменты адреса файла закодированы буквами от А до Ж.

Запишите в таблицу последовательность этих букв, кодирующую адрес указанного файла в сети Интернет.

А) obr.
Б) /
В) org
Г) ://
Д) doc
Е) rus.
Ж) https

Решение

Будем записывать URL указанного файла по порядку, согласно структуре URL.

Так как в данном случае не сказано, что файл находится в каком-то особенном месте сервера, будем считать, что файл расположен непосредственно на сервере (то есть, в корне файловой системы). Это значит, что путь к файлу на сервере состоит только из имени файла.

Таким образом, нужный нам URL состоит из следующих пяти частей:

протокол	://	адрес сервера	/	имя файла

Подставляем по очереди указанные части:

протокол	://	адрес сервера	/	имя файла
https	://	obr.org	/	rus.doc

То есть, все вместе: https://obr.org/rus.doc

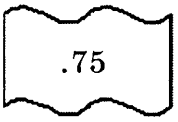
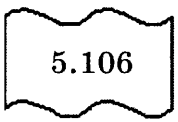
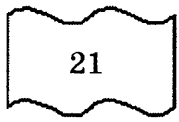
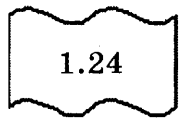
Остаётся только закодировать адрес файла указанными буквами:

https	://	obr	.org	/	rus	.doc
Ж	**Г**	**А**	**В**	**Б**	**Е**	**Д**

Ответ: ЖГАВБЕД

Задача 2. На месте преступления были обнаружены четыре обрывка бумаги. Следствие установило, что на них записаны фрагменты одного IP-адреса. Криминалисты обозначили эти фрагменты буквами А, Б, В и Г. Восстановите IP-адрес.

В ответе укажите последовательность букв, обозначающих фрагменты, в порядке, соответствующем IP-адресу.

			
А	**Б**	**В**	**Г**

Решение

Воспользуемся ограничениями, которые нам известны для ip-адреса. Это четыре числа, разделенные точками, каждое из которых может принимать значения от 0 до 255.

Один из способов решения — перебрать все возможные варианты, составив рядом 4 клочка бумаги. Это всего 24 варианта. Не очень много. Но мы попробуем решить задачу быстрее.

Постараемся найти среди фрагментов такой, про который можно будет сразу что-нибудь сказать исходя из ограничений ip-адреса. Таких фрагментов здесь сразу два: фрагмент «.75» и фрагмент «5.106». Так

как числа в ip-адресе не могут быть больше 255, к этим фрагментам справа нельзя добавить ещё цифры. Действительно, добавление к фрагменту «.75» справа ещё хотя бы одной цифры сделает число после точки не меньше, чем 750 (это больше максимально возможного значения 255). А добавление к фрагменту «5.106» справа ещё хотя бы одной цифры сделает число после точки не меньшим чем 1060 (что также явно больше 255). Значит, к этим фрагментам справа цифр не добавляется. Что может быть в том случае, если это самый правый фрагмент ip-адреса. Либо в случае, когда к этому фрагменту справа прикладывается фрагмент, начинающийся с точки. Оба таких фрагмента не могут быть самыми правыми в ip-адресе. А фрагмент с точкой только один — «.75». Получается, что к фрагменту «5.106» справа возможно добавить фрагмент, начинающийся с точки — «.75», а к фрагменту «.75» справа добавлено ничего уже не может быть. Следовательно, самый правый фрагмент ip-адреса — это «.75». Он непосредственно справа прикладывается к фрагменту «5.106». Получаем правую часть ip-адреса: «5.106.75». Слева к ней должны быть приложены фрагменты «21» и «1.24». Это возможно сделать всего двумя способами (сначала один, затем другой, либо наоборот). Рассмотрим оба варианта:

211.245.106.75

1.24215.106.75

Очевидно, что второй вариант нам не подходит, так как второе число явно больше 255.

Значит, для ответа используем: 211.245.106.75. Вспоминаем, какая часть этого адреса соответствует обозначениям фрагментов А, Б, В, Г:

21	1.24	5.106	.75
В	**Г**	**Б**	**А**

Ответ: ВГБА.

Справочное издание

Денис Михайлович Ушаков

ИНФОРМАТИКА

Новый полный справочник для подготовки к ОГЭ

Редакция «Образовательные проекты»
Ответственный редактор *Н. А. Шармай*

Подписано в печать 22.09.2016.
Формат 84×108 $^{1}/_{32}$. Усл. печ. л. 15,12

(Новый полный справочник для подготовки к ОГЭ)
Тираж 2000 экз. Заказ № 7179
(Самый популярный справочник для подготовки к ОГЭ)
Тираж 3000 экз. Заказ № 7177

Общероссийский классификатор продукции
ОК-005-93, том 2; 953005 — литература учебная

Сертификат соответствия
№ РОСС RU.ME04.H01397 от 29.03.2016 г.

ООО «Издательство АСТ»
129085, г. Москва, Звёздный бульвар, д. 21, стр. 3, комн. 5
Наши электронные адреса: www.ast.ru;
E-mail: stelliferovskiy@ast.ru

Отпечатано с готовых файлов заказчика
в АО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, г. Ульяновск, ул. Гончарова, 14

По вопросам приобретения книг обращаться по адресу:
123317, г. Москва. Пресненская наб., д. 6, стр. 2,
Деловой комплекс «Империя», а/я № 5