ЕДИНЫЙ ГОСУДАРСТВЕННЫЙ ЭКЗАМЕН

О.Б. Богомолова

ИНФОРМАТИКА

ПОЛНЫЙ СПРАВОЧНИК ДЛЯ ПОДГОТОВКИ К ЕГЭ

- Все нужные для ЕГЭ темы школьного курса — «Информация и её кодирование», «Моделирование и компьютерный эксперимент», «Системы счисления», «Основы логики», «Элементы теории алгоритмов», «Программирование», «Архитектура компьютеров и компьютерных сетей», «Технология обработки графической и звуковой информации», «Обработка числовой информации», «Технологии поиска и хранения информации», «Телекоммуникационные технологии»
- Большое количество примеров с решениями
- Полное соответствие современному формату экзамена

СОВРЕМЕННЫЙ ФОРМАТ ЭКЗАМЕНА ПРОВЕРЕНО НА ПРАКТИКЕ

О.Б. БОГОМОЛОВА

ИНФОРМАТИКА

ПОЛНЫЙ СПРАВОЧНИК ДЛЯ ПОДГОТОВКИ К ЕГЭ

АСТ • Астрель
Москва

УДК 373:002
ББК 32.81я721
Б74

Богомолова, Ольга Борисовна

Б74 Информатика : Полный справочник для подготовки к ЕГЭ / О.Б. Богомолова. — Москва: АСТ: Астрель, 2014. — 415, [1] с.

ISBN 978-5-17-080256-2 (ООО «Издательство АСТ») (пер.)
ISBN 978-5-271-46340-2 (ООО «Издательство Астрель») (пер.)
ISBN 978-5-17-080255-5 (ООО «Издательство АСТ») (обл.)
ISBN 978-5-271-46339-6 (ООО «Издательство Астрель») (обл.)

Справочник поможет школьнику освежить в памяти основной теоретический материал по всему курсу информатики за 7–11 классы, ознакомиться с принципами решения типовых задач ЕГЭ, предлагавшихся в последние несколько лет, и подготовиться к экзамену.

Для школьников, учителей информатики и методистов.

УДК 373:002
ББК 32.81я721

Богомолова Ольга Борисовна

ИНФОРМАТИКА

Полный справочник для подготовки к ЭГЭ

Редакция *«Образовательные проекты»*
Ответственный редактор *Н.А. Шармай*
Художественный редактор *Т.Н. Войткевич*
Технический редактор *А.Л. Шелудченко*
Корректор *И.Н. Мокина*
Оригинал-макет подготовлен *ООО «БЕТА-Фрейм»*

Подписано в печать 15.11.2013. Формат $84 \times 108^{1}/_{32}$
Усл. печ. л. 21,84. Тираж 2000 экз. (пер.) Заказ № 9088; 3000 экз. (обл.) Заказ № 9089.

Общероссийский классификатор продукции
ОК-005-93, том 2; 953005 — литература учебная
Сертификат соответствия № РОСС.RU.АЕ51.Н16526 от 26.09.2013

ООО «Издательство АСТ». 129085, г. Москва, Звёздный б-р, д. 21. стр. 3, комн. 5
ООО «Издательство Астрель». 129085, г. Москва, пр-д Ольминского, д. 3а

Отпечатано с готовых файлов заказчика в ОАО «Первая Образцовая типография», филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ». 432980, г. Ульяновск, ул. Гончарова, 14

ISBN 978-5-17-080256-2 (ООО «Издательство АСТ») (пер.)
ISBN 978-5-271-46340-2 (ООО «Издательство Астрель») (пер.)
ISBN 978-5-17-080255-5 (ООО «Издательство АСТ») (обл.)
ISBN 978-5-271-46339-6 (ООО «Издательство Астрель») (обл.)

Содержание

Предисловие

Единый государственный экзамен (ЕГЭ) в настоящее время признан в качестве основной формы объективной оценки качества подготовки школьников, освоивших образовательные программы среднего (полного) общего образования. Результаты ЕГЭ являются результатами одновременно государственной (итоговой) аттестации для образовательных учреждений среднего (полного) общего образования и вступительных экзаменов по соответствующим общеобразовательным предметам — для образовательных учреждений среднего и высшего профессионального образования.

Информатика (прежнее название предмета — «Информатика и информационно-коммуникационные технологии» — действовало до введения в 2012 г. нового Федерального государственного образовательного стандарта) относится к 12 предметам, сдача ЕГЭ по которым производится учащимися на добровольной основе. Однако перечень вузов и ссузов, требующих наличия свидетельства об успешной сдаче ЕГЭ по информатике для поступления на основные специальности, постоянно растет. Возрастает год от года и сложность заданий, предлагаемых на ЕГЭ по информатике.

Поэтому подготовка к ЕГЭ является высоко актуальной задачей как для самих учащихся старшей школы, так и для учителей информатики.

Наилучшей стратегией такой подготовки является, конечно же, системное и целенаправленное формирование основных информационных компетенций школьников, отработка решения разнообразных заданий и выработка навыков работы с основными средствами ИКТ по всем без исключения изучаемым темам курса. Однако нынешние реалии, к сожалению, требуют принимать в расчет и недостаточное количество часов, отпущенных на изучение предмета «Информатика» федеральными учебными планами, и практическое отсутствие задачников-практикумов, поддерживающих не фрагментарное ознакомление с отдельными темами, а плотное прохождение всего курса.

Эти сложности заставляют выстраивать стратегию подготовки к ЕГЭ прежде всего на базе индивидуальной работы

школьников с учебным материалом под руководством и при активной консультационной поддержке со стороны учителя. То есть учитель вначале проводит для класса разбор решения типовых заданий по данной теме, объясняя, почему ту или иную задачу лучше всего решать именно так, а затем решение подобных задач отрабатывается в ходе массивной практической работы с последующим контролем (ручным или автоматизированным) на базе нескольких вариантов заданий, максимально раскрывающих возможные вариации их условий.

Подготовиться к сдаче ЕГЭ на 90–100 баллов — задача достаточно сложная, требующая обоюдной заинтересованности и обоюдного напряжения сил как учащегося, так и учителя, но, как показывает педагогическая практика автора данной книги, вполне выполнимая.

Предлагаемый вашему вниманию справочник — результат многолетней педагогической практики автора. Структура справочника основана на результатах анализа тематики заданий ЕГЭ за последние несколько лет (с 2008 по 2013 год). Книгу можно использовать для самостоятельной (в том числе под контролем со стороны учителя) индивидуальной работы школьника при подготовке к ЕГЭ, для повторения ранее изученных основных теоретических сведений и выработки навыков решения задач.

Каждый раздел справочника по каждой изучаемой теме включает конспект теоретического материала и разбор решений ряда типовых заданий ЕГЭ за 2–3 последних года. Использование справочника рекомендуется в сочетании с дополнительным решением вариантов заданий ЕГЭ, предлагаемых, например, в сборнике «Самое полное издание типовых вариантов заданий ЕГЭ-2014. Информатика» (авт.-составитель Д.М. Ушаков, А.П. Якушкин. — Москва: АСТ: Астрель, 2014).

Раздел 1. Информация. Измерение информации. Кодирование информации

A11, B1 Измерение количества информации

Конспект

Вероятностный подход к измерению количества информации. Информация как снятая неопределённость в знаниях

Для определения количества информации, содержащейся в сообщении о каком-либо объекте или событии, используется **вероятностный подход**. Он основан на следующих соображениях:

- те или иные события имеют некоторую вероятность (возможность произойти или не произойти);
- событие, которое совершается всегда, имеет вероятность, равную 1 (например, восход Солнца); событие, которое не совершается никогда, имеет вероятность, равную 0 (например, восход Солнца на западе); в остальных случаях вероятность совершения события есть дробное число от 0 до 1;
- получая сообщение о совершении (или не совершении) некоторого события, мы получаем некоторое количество информации, которое определяется снятой с её помощью неопределённостью наших знаний об указанном событии:
  - если вероятность совершения события точно равна 1 или 0 (т.е. мы точно знаем, что событие произойдёт (или не произойдёт), то никакой неопределённости в наших знаниях нет, и сообщение о таком событии несёт нулевое количество информации;
  - для равновероятных событий чем больше их количество (т.е. шире возможный выбор вариантов и потому меньше вероятность каждого из них),

тем большее количество информации несёт сообщение о совершившемся конкретном событии;

- количество информации в сообщении о совершении (несовершении) *нескольких* независимых событий равно *сумме* количеств информации, содержащейся в сообщениях о каждом отдельном таком событии.

Формула Хартли

Для N равновероятных возможных событий количество информации, которое несёт сообщение о выборе (совершении) одного конкретного события, определяется **формулой Хартли:**

$$I = \log_2 N,$$

где log — *функция логарифма по основанию 2*, обратная возведению значения основания логарифма в степень, равную I, т.е. из формулы Хартли следует зависимость:

$$N = 2^I.$$

Для облегчения вычислений для значений N, представляющих собой степени числа 2, можно составить таблицу (табл. 1.1):

Таблица 1.1

N	2	4	8	16	32	64	128	256	512	1024
I (бит)	1	2	3	4	5	6	7	8	9	10

Для значений N, не равных степени двойки, при определении количества информации в битах из вышеприведённой таблицы берётся *ближайшее большее значение* N, равное степени 2. Например, для 48 равновозможных событий количество информации, которое содержится в сообщении о совершении конкретного события, принимается равным 6 бит (так как ближайшее большее значение N, равное степени числа 2, равно 64).

«Принцип вилки»

Для приближённого вычисления количества информации при значении N, не равном 2 в некоторой степени, определяются значения количества информации для

двух соседних значений N, составляющих степени 2, и составляется соответствующее двойное неравенство.

Например, пусть нужно оценить количество информации в сообщении о выпадении на верхней грани игрального кубика шести точек. В этом случае $N = 6$. Ближайшими к нему являются значения — степени двойки: $N = 4$ $(2 \cdot 2)$ и $N = 8$ $(2 \cdot 2 \cdot 2)$. Тогда можно составить неравенство:

$$2^2 < 2^I < 2^3.$$

Отсюда искомое количество информации будет больше 2 и меньше 3 битов.

Формула Шеннона. Связь количества информации с понятием вероятностей

Для N событий с различными вероятностями p_1, p_2, ..., p_N количество информации определяется **формулой Шеннона**:

$$I = -\sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i}.$$

Если все эти события равновероятны, т. е. $p_1 = p_2 = ... = p_N = p$, то очевидно, что формула Шеннона преобразуется в формулу Хартли (которая, таким образом, представляет собой частный случай формулы Шеннона).

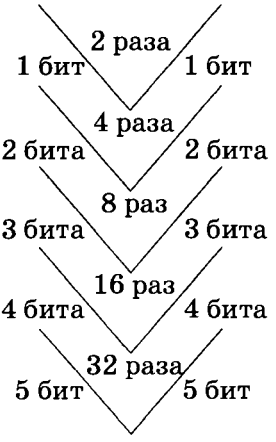
Связь между количеством информации и вероятностью события

Для N равновероятных событий вероятность одного отдельного события $p = 1/N$. С учётом этого формула Хартли может быть преобразована в соотношение:

$$I = \log_2 \frac{1}{p}.$$

В этом случае вычисление количества информации можно производить по табл. 1.1, предварительно вычислив значение N как величину, обратную значению p. Например, для события, вероятность которого (p) составляет 0,018, получается $N = 1/0{,}018 = 55{,}56$, тогда берётся ближайшее большее значение N, кратное 2 $(N = 64)$ и по табл. 1.1 определяется, что $I = 6$ бит.

«Принцип ёлочки»



Сколько информации несёт в себе некоторое сообщение? Известно, что количество информации, равное 1 биту, соответствует снятию неопределённости при помощи ответа «да» или «нет» на один элементарный вопрос, т. е. 1 бит соответствует уменьшению неопределённости в 2 раза. А чему соответствует уменьшение неопределённости, например, в 4 раза? В подобном случае можно задать последовательно два вопроса, на которые даются ответы «да» или «нет». В общем, количество информации в n бит позволяет уменьшить неопределённость в 2^n раз.

Бит. Байт. Производные величины

Принято считать, что минимально возможное количество информации соответствует такому сообщению, получение которого уменьшает неопределённость в 2 раза (пример — сообщение о выпадении на подброшенной монете «орла» из двух равновозможных вариантов — «орёл» и «решка»). Это минимальное количество информации получило название **«бит»** (англ. ***bit*** как сокращение названия ***binary digit*** — двоичная цифра).

В вычислительной технике бит соответствует одному *двоичному разряду*, который может принимать одно из двух возможных значений: 0 или 1. В качестве более крупной величины принят **байт**, соответствующий двоичному числу из 8 разрядов (битов). В оперативной памяти компьютера минимальный объём ячейки памяти, выделяемой для хранения какой-либо величины, как правило, равен одному байту. Ячейки большего размера имеют объём, кратный байту с коэффициентом кратности 2: 2 байта (16 бит), 4 байта (32 бита), 8 байтов (64 бита). Такую «порцию» информации принято называть **машинным словом.**

В теории информации количество информации может быть дробной величиной. В вычислительной технике количество информации может составлять только целое число битов (дробное значение при необходимости округляется в бóльшую сторону).

В вычислительной технике в большинстве практических задач получаемое количество битов округляется в большую сторону до целого количества байтов, хотя в некоторых случаях возможна «потоковая» запись значений, состоящих из количества битов, не кратного 8.

Для обозначения количеств информации, бóльших, чем байт, приняты следующие производные величины:

1 ***килобайт*** **(КБ)** = (2^{10} = 1024) байт;

1 ***Мегабайт*** **(МБ)** = (2^{10} = 1024) килобайт = (2^{20} = = 1048576) байт;

1 ***Гигабайт*** **(ГБ)** = (2^{10} = 1024) Мегабайт = (2^{20} = = 1048576) килобайт = (2^{30} = 1073741824) байт;

1 ***Терабайт*** **(ТБ)** = (2^{10} = 1024) Гигабайт = (2^{20} = = 1048576) Мегабайт = (2^{30} = 1073741824) килобайт = (2^{40} = 1099511627776) байт;

1 ***Петабайт*** **(ПБ)** = (2^{10} = 1024) Терабайт;

1 ***Эксабайт*** **(ЭБ)** = (2^{10} = 1024) Петабайт;

1 ***Зеттабайт*** **(ЗБ)** = (2^{10} = 1024) Эксабайт;

1 ***Йоттабайт*** **(ЙБ)** = (2^{10} = 1024) Зеттабайт.

Внимание! В отличие от одноименных приставок в кратных величинах в математике изменение величин в вычислительной технике происходит на каждом «шаге» вышеуказанной шкалы на 2^{10} = 1024, а не на 10^3 = 1000.

Для избежания этой путаницы были предложены особые, двоичные приставки для производных величин количества информации:

Кибибайт	KiB	2^{10} (1024)
Мебибайт	MiB	2^{20} (1048576)
Гибибайт	GiB	2^{30} (1073741824)
Тебибайт	TiB	2^{40} (1099511627776)
Пебибайт	PiB	2^{50} (1125899906842624)
Эксбибайт	EiB	2^{60} (1152921504606846976)

Алфавитный (алгоритмический) подход к измерению количества информации. Алфавит. Мощность алфавита

В этом случае количество информации в сообщении представляет собой чисто технический параметр (важный с точки зрения хранения или передачи информации) и не зависит от содержания сообщения.

При алфавитном подходе информационное сообщение рассматривается как некоторое количество (K) **знаков (символов, кодов)** из некоторого используемого полного набора, называемого **алфавитом**. Количество (N) знаков в алфавите называется **мощностью** этого алфавита.

В данном конкретном сообщении не обязательно используются **все** знаки алфавита. Мощность алфавита определяется не набором знаков, используемых в конкретном сообщении, а количеством знаков, которые вообще могут быть использованы в сообщениях, кодируемых в соответствии с данным алфавитом.

Алгоритм определения количества информации в сообщении:

1) определяется мощность используемого алфавита N;

2) определяется количество информации, приходящееся в алфавите на один его знак:

- если использование всех знаков равновероятно, то используется формула Хартли (либо её следствие: $N = 2^I$) и табл. 1.1;
- если известны вероятности использования тех или иных знаков (на основе составленной таблицы частоты встречаемости этих знаков), то используется формула Шеннона;

3) вычисленное количество информации (I), приходящееся на один знак, умножается на количество (K) знаков в данном сообщении:

$$I_\Sigma = I \cdot K.$$

Количество различных состояний панели, имеющей M элементов, каждый из которых может находиться в N различных состояниях, равно количеству различных M-разрядных чисел (начиная с нуля) в системе счисления с основанием N и вычисляется как N^M.

Количество всех M-разрядных чисел в системе счисления с основанием N равно N^M. Максимальное значение M-разрядного числа в системе счисления с основанием N равно $(N^M - 1)$.

Если в условии задачи предлагается определить количество сигналов, подаваемых с помощью разного количества элементов (от X до Y), то нужно отдельно вычислить количества возможных сигналов для каждого возможного числа элементов и просуммировать полученные значения.

Разбор типовых задач

Задача 1*[1]. Для регистрации на сайте некоторой страны пользователю требуется придумать пароль. Длина пароля — ровно 11 символов. В качестве символов используются десятичные цифры и 12 различных букв местного алфавита, причём все буквы используются в двух начертаниях: как строчные, так и заглавные (регистр буквы имеет значение!).

Под хранение каждого такого пароля на компьютере отводится минимально возможное и одинаковое целое количество байтов, при этом используется посимвольное кодирование и все символы кодируются одинаковым и минимально возможным количеством битов.

Определите объём памяти, который занимает хранение 60 паролей.

1) 540 байт
2) 600 байт
3) 660 байт
4) 720 байт

[1] Задачи, отмеченные *, взяты из демонстрационных вариантов разных лет, с официального сайта Федерального института педагогических измерений (www.fipi.ru).

Решение

Данная задача решается с использованием *алфавитного подхода к измерению количества информации*.

1. Определяется мощность используемого алфавита. Используются десятичные цифры (10 различных знаков) и 12 букв, каждая из которых может иметь два возможных начертания ($12 \times 2 = 24$ различных знака). Итого мощность используемого алфавита составляет: $10 + 12 \times 2 = 34$ знака.

2. Исходя из известной мощности алфавита определяется количество битов, соответствующее каждому знаку. $N = 34$. Речь идёт о *целом* (не дробном!) количестве битов, минимально достаточном для представления одного знака такого алфавита. Поэтому выбирается ближайшее большее число N, равное степени числа 2: $N = 2^6 = 64$ (значения $2^5 = 32$ недостаточно). Тогда согласно формуле $N = 2^I$ получается: $I = 6$ бит на один знак алфавита.

3. Длина пароля (длина сообщения, кодируемого с использованием рассмотренного алфавита) равна 11 символам ($K = 11$). Тогда согласно формуле $I_\Sigma = I \cdot K$, количество информации в битах, соответствующее всему такому сообщению (паролю), равно $6 \cdot 11 = 66$ битов.

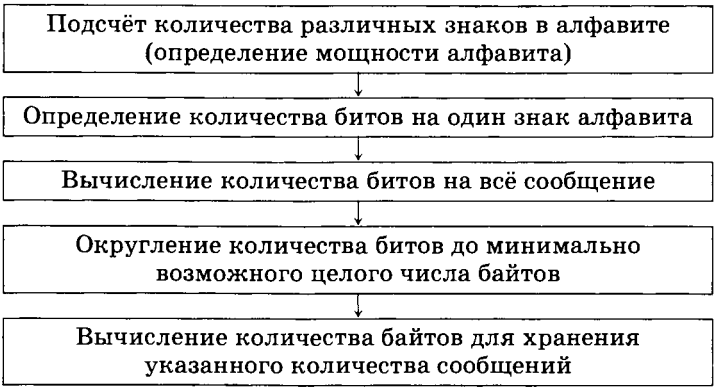
4. По условию задачи, *под хранение каждого такого пароля на компьютере отводится минимально возможное и одинаковое целое количество байтов*. Определяется минимальное количество *целых* байтов, достаточное, чтобы уместить 66 битов. 1 байт равен 8 битам. Выполняем деление количества битов (66) на 8 с округлением результата до целого в большую сторону: $66 / 8 = 8{,}25 \rightarrow 9$ байтов.


Следует не забывать выполнять перевод количества битов в количество байтов!

5. Тогда для хранения 60 паролей потребуется $60 \cdot 9 = 540$ байтов.

Ответ: 540 байтов (вариант ответа №1).

Схема решения задачи



Задача 2*. Автоматическое устройство осуществило перекодировку информационного сообщения на русском языке длиной в 20 символов, первоначально записанного в 2-байтном коде Unicode, в 8-битную кодировку КОИ-8. На сколько бит уменьшилась длина сообщения?

Решение

Информационный объём исходного сообщения (I_1) равен: 20 (символов) × 2 (байта) = 40 байтов = 40 · 8 = = 320 битов.

Информационный объём перекодированного сообщения (I_2) равен: 20 (символов) × 8 (битов) = 160 битов.

Уменьшение информационного объёма («длины») сообщения при его перекодировке составляет:

$\Delta I = I_1 - I_2 = 320 - 160 = 160$ битов.

Ответ: 160 битов.

Следует быть внимательными к условию задачи! В нём может спрашиваться, **на сколько битов** меняется объём сообщения либо **во сколько раз** меняется этот объём. В зависимости от этого нужно не забывать правильно выполнять расчёты и преобразовывать все величины к одной размерности (бит или байт).

Задача 3*.

В некоторой стране автомобильный номер длиной 7 символов составляют из заглавных букв (используются только 22 различные буквы) и десятичных цифр в любом порядке. Каждый такой номер в компьютерной программе записывается минимально возможным и одина-

ковым целым количеством байт (при этом используют посимвольное кодирование и все символы кодируются одинаковым и минимально возможным количеством бит).

Определите объём памяти, отводимый этой программой для записи 50 номеров.

1) 350 байт 2) 300 байт 3) 250 байт 4) 200 байт

Решение

1. Определяется мощность используемого алфавита. Используются десятичные цифры (10 различных знаков) и 22 буквы (только заглавные). Мощность используемого алфавита составляет: 10 + 22 = 32 знака.

2. Исходя из известной мощности алфавита, определяется количество битов, соответствующее каждому знаку. $N = 32$. Ему соответствует значение $2^5 = 32$. Тогда согласно формуле $N = 2^I$ получается: $I = 5$ бит на один знак алфавита.

3. Длина автомобильного номера (длина сообщения, кодируемого с использованием рассмотренного алфавита) равна 7 символам ($K = 7$). Согласно формуле $I_\Sigma = I \cdot K$, количество информации в битах, соответствующее всему такому сообщению (номеру), равна $5 \cdot 7 = 35$ битов.

4. По условию задачи, *под хранение каждого такого номера на компьютере отводится минимально возможное и одинаковое целое количество байтов.* Определяется минимальное количество *целых* байтов, достаточное, чтобы уместить 35 битов. 1 байт равен 8 битам. Выполняется деление количества битов (35) на 8 с округлением результата до целого в большую сторону: 35 / 8 = 4,375 → 5 байтов.

5. Тогда для хранения 50 номеров потребуется $50 \cdot 5 = 250$ байтов.

Ответ: 250 байтов (вариант ответа №3).

Задача 4*. Для передачи сигналов на флоте используются специальные сигнальные флаги, вывешиваемые в одну линию (последовательность важна). Какое количество различных сигналов может передать корабль при помощи четырёх сигнальных флагов, если на ко-

рабле имеются флаги трёх различных видов (флагов каждого вида неограниченное количество).

Решение

1. Имеются флаги трёх возможных видов, — следовательно, это аналог троичной системы счисления.

2. Всего последовательность флагов содержит четыре флага, значит, она является аналогом четырёхзначного троичного числа.

3. Количество различных сигналов, которые можно подавать при помощи такого набора флагов, равно количеству всех возможных значений троичных чисел от 0000 до 2222 включительно, т.е. $3^4 = 81$ сигнал.

Если имеется M элементов, каждый из которых может находиться в N различных состояниях (либо быть одним из N различных видов), то количество состояний такой системы (передаваемых с её помощью сообщений) равно количеству различных M-разрядных чисел (начиная с нуля) в системе счисления с основанием N и вычисляется как N^M.

Ответ: 81 сигнал.

Задача 5*. Некоторое сигнальное устройство за одну секунду передает один из трёх сигналов. Сколько различных сообщений длиной в четыре секунды можно передать при помощи этого устройства?

Решение

1. Возможна передача одного из трёх односекундных сигналов — следовательно, это аналог троичной системы счисления.

2. Длина сообщения составляет 4 секунды, значит, это аналог четырёхзначного троичного числа.

3. Количество различных таких сигналов равно количеству всех возможных значений четырёхзначных троичных чисел (от 0000 до 2222 включительно), т.е. $3^4 = 81$ сообщение.

Количество различных сообщений, включающих в себя M элементов, каждый из которых может иметь N различных состояний, равно количеству различных M-разрядных чисел (начиная с нуля) в системе счисления с основанием N и вычисляется как N^M.

Ответ: 81 сообщение.

A9 Неравномерный двоичный код

Конспект

Неравномерный двоичный код. Условие Фано

Кодирование символов обычно предполагает, что каждому символу всегда сопоставляется одинаковое количество битов (например, в кодовой таблице ASCII каждому символу сопоставляется один байт, хранящий порядковый номер того или иного символа в этой таблице). Такой способ кодирования прост и удобен, однако очевидно, что он является не самым оптимальным. Для значительной части символов используются не все биты отведенных под них байтов (часть старших битов — нулевые), а при наличии в тексте только части символов, предусмотренных в таблице ASCII (например, если текст содержит только прописные русские буквы), приходится все равно использовать 8-битный код.

Более компактным является *неравномерный двоичный код* (особенно если при его построении исходить из частоты встречаемости различных символов и присваивать наиболее часто используемым знакам самые короткие коды, как это сделано в *методе Хаффмана*). При этом количество битов, отводимых для кодирования символов, в целом зависит от количества используемых в конкретном случае различных символов (от *мощности алфавита*), а коды, соответствующие разным символам, могут иметь различную длину в битах.

Главное при таком кодировании — обеспечить возможность *однозначного декодирования* записанной с помощью этих кодов строки (поочередного, слева направо, выделения и распознавания из сплошной последовательности нулей и единиц кодов отдельных букв). Для этого коды символам необходимо назначать в соответствии с *условиями Фано*.

Прямое условие Фано. **Неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с началом (префиксом) какого-либо другого, более длинного кода.**

A	B	C
10	11	001

D: 00
недопустимо:

C	001
D	00

Код D совпадает с началом кода C

A	B	C
10	11	00

D: 11
недопустимо:

B	11
D	11

Код D совпадает с кодом B

A	B	C
100	110	010

D: 00
допустимо:

Код D не совпадает ни с одним другим кодом и с началом никакого другого кода

Обратное условие Фано. **Неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с окончанием (постфиксом) какого-либо другого, более длинного кода.**

A	B	C
10	11	001

D: 01
недопустимо:

C	001
D	01

Код D совпадает с концом кода C

A	B	C
10	11	00

D: 11
недопустимо:

B	11
D	11

Код D совпадает с кодом B

A	B	C
100	110	010

D: 01
допустимо:

Код D не совпадает ни с одним другим кодом и с началом никакого другого кода

Для однозначности декодирования последовательности кодов достаточно выполнения *хотя бы одного* из двух вышеуказанных условий Фано:

— при выполнении прямого условия Фано последовательность кодов однозначно декодируется с начала;

— при выполнении обратного условия Фано последовательность кодов однозначно декодируется с конца.

Выбрать, какое из двух правил Фано используется при решении конкретной задачи, можно, проанализировав коды в условии задачи (без учёта кода, проверяемого в вариантах ответа): если для исходных кодов выполняется прямое правило Фано, то его и нужно использовать при решении, и наоборот.

Вместе с тем нужно помнить, что правила Фано — это достаточное, но не необходимое условие однозначного декодирования: если не выполняется ни прямое, ни обратное правило Фано, конкретная двоичная последовательность может оказаться такой, что она декодируется однозначно (так как остальные возможные варианты до конца декодирования довести не удаётся). В подобном случае необходимо пытаться строить дерево декодирования в обоих направлениях.

 Рекомендуется начинать решение задач такого типа с анализа выполнимости правил Фано для исходных кодов, указанных в условии задачи (т.е. без учета искомого кода в вариантах ответов). В зависимости от того, какое из двух правил Фано выполняется для исходных кодов, при дальнейшем решении задачи производится сравнение более короткого кода с началом (при выполнении прямого правила Фано) или с концом (при выполнении обратного правила Фано) более длинного кода.

 Если для заданной последовательности кодов выполняется прямое правило Фано, то её декодирование необходимо вести с начала (слева направо).
Если для заданной последовательности кодов выполняется обратное правило Фано, то её декодирование необходимо вести с конца (справа налево).

 При сравнении пары кодов удобно подписывать более короткий код под более длинным, выравнивая эти записи по левому краю — для прямого правила Фано либо по правому краю — для обратного правила Фано.

Разбор типовых задач

Задача 1*. Для кодирования некоторой последовательности, состоящей из букв А, Б, В, Г и Д, решили использовать неравномерный двоичный код, позволяющий однозначно декодировать двоичную последовательность, появляющуюся на приёмной стороне канала связи. Использовали код: А–1, Б–000, В–001, Г–011. Укажите, каким кодовым словом должна быть закодирована буква Д. Длина этого кодового слова должна

быть наименьшей из всех возможных. Код должен удовлетворять свойству однозначного декодирования.

1) 00 2) 01 3) 11 4) 010

Решение

Проверяемый код буквы Д	Существующие коды букв А, Б, В и Г				Вывод
	А	Б	В	Г	
	1	000	001	011	
00	00 1 Совпадения нет	000 00 **Совпадение есть!**	001 00 **Совпадение есть!**	011 00 Совпадения нет	Код 00 для буквы Д непригоден
01	01 1 Совпадения нет	000 01 Совпадения нет	001 01 Совпадения нет	011 01 **Совпадение есть!**	Код 01 для буквы Д непригоден
11	11 1 **Совпадение есть!**	000 11 Совпадения нет	001 11 Совпадения нет	011 11 Совпадения нет	Код 11 для буквы Д непригоден
010	010 1 Совпадения нет	000 010 Коды не равны	001 010 Коды не равны	011 010 Коды не равны	**Код 010 для буквы Д пригоден**

В итоге, допустимым для буквы Д является код 010. Это единственный возможный вариант, поэтому условие о том, что кодовое слово должно иметь минимально возможную длину, в данном случае не требуется.

Ответ: код 010 (вариант ответа №4).

Задача 2*. Для передачи по каналу связи сообщения, состоящего только из символов А, Б, В и Г, используется неравномерный (по длине) код: А–00, Б–11, В–010, Г–011. Через канал связи передаётся сообщение: ГБВАВГ. Закодируйте сообщение данным кодом. Полученную двоичную последовательность переведите в шестнадцатеричную систему счисления. Какой вид будет иметь это сообщение?

1) 71013 2) DBCACD 3) 7A13 4) 31A7

Решение

1. Кодируется заданное сообщение, заменяя каждую его букву соответствующим кодом:

Сообщение:	Г	Б	В	А	В	Г
Коды:	011	11	010	00	010	011

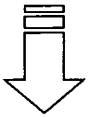
Тогда сообщению соответствует строка цифр: **0111101000010011.**

2. Рассматривая полученную строку как двоичное число, необходимо преобразить его в шестнадцатеричную систему счисления (метод преобразования по тетрадам):

 Таблица соответствия между шестнадцатеричными цифрами и тетрадами двоичных цифр:

Двоичная триада	0000	0001	0010	0011	0100	0101	0110	0111
Шестнадцатеричное значение	0	1	2	3	4	5	6	7
Двоичная триада	1000	1001	1010	1011	1100	1101	1110	1111
Шестнадцатеричное значение	8	9	A	B	C	D	E	F

0111 1010 0001 0011



7 A 1 3

Ответ: 7A13 (вариант ответа №3).

Задача 3*. Для кодирования некоторой последовательности, состоящей из букв А, Б, В, Г и Д, решили использовать неравномерный двоичный код, позволяющий однозначно декодировать двоичную последовательность, появляющуюся на приёмной стороне канала связи. Использовали код: А–110, Б–100, В–01, Г–11. Укажите, каким кодовым словом должна быть закодирована бук-

ва Д. Длина этого кодового слова должна быть наименьшей из всех возможных. Код должен удовлетворять свойству однозначного декодирования.

1) 10 2) 0 3) 00 4) 010

Решение

Для исходных кодов: А–110, Б–100, В–01, Г–11 не выполняется прямое правило Фано, так как код буквы Г (11) совпадает с началом кода буквы А (110). В этом случае нужно проверить, выполняется ли для этих кодов обратное правило Фано:

— коды букв А и Б не равны; коды букв В и Г также не равны;

— код буквы В (01) не совпадает с окончанием ни кода буквы А (110), ни буквы Б (100);

— код буквы Г (11) также не совпадает с окончанием ни кода буквы А (110), ни буквы Б (100).

Следовательно, для данного набора кодов выполняется обратное правило Фано.

Проверяемый код буквы Д	**Существующие коды букв А, Б, В и Г**				**Вывод**
	А	**Б**	**В**	**Г**	
	110	**100**	**01**	**11**	
10	110 10 **Совпадение есть!**	100 10 Совпадения нет	01 10 Коды не равны	11 10 Коды не равны	Код 10 для буквы Д непригоден
0	110 0 **Совпадение есть!**	100 0 **Совпадение есть!**	01 0 Совпадения нет	11 0 Совпадения нет	Код 0 для буквы Д непригоден
00	110 00 Совпадения нет	100 00 **Совпадение есть!**	01 00 Коды не равны	11 00 Коды не равны	Код 00 для буквы Д непригоден
010	110 010 Коды не равны	100 010 Коды не равны	010 01 Совпадения нет	010 11 Совпадения нет	**Код 010 для буквы Д пригоден**

Таким образом, допустимым для буквы Д является код 010. Это единственный возможный вариант, поэтому условие о том, что кодовое слово должно иметь минимально возможную длину, в данном случае не требуется.

Ответ: код 010 (вариант ответа №4).

B10 Передача информации по коммуникационным каналам

Конспект

Передача информации

Передача информации — это информационный процесс, при котором производится перемещение информации через пространство и/или через время от одного субъекта (*источника информации*) к другому субъекту (*приемнику информации*). При этом информация передается в форме *документа* (записи на некотором *физическом носителе*) либо в форме *сообщения* (последовательности *сигналов* по *каналу связи*).

В процессе передачи информации возможны *помехи*: случайные искажения физических характеристик канала связи, которые являются носителем информации, либо повреждения физического носителя. Наличие таких помех вынуждает применять различные способы борьбы с ними, в том числе многократное резервирование документов либо многократную пересылку сообщений, применение специальных методов контроля и коррекции ошибок (контрольная сумма, код Хемминга и пр.).

Измерение скорости передачи информации

Скорость передачи информации по каналу связи (обычно рассматривается только передача сообщений[1])

[1] В случае с записью документов на физическом носителе, в принципе, тоже можно вычислить скорость «передачи информации» исходя из скорости перемещения носителя и его объёма, однако обычно такие расчёты выполняются и приводятся только как курьёз.

вычисляется как количество информации, переданной за одну секунду. Базовой единицей при этом является *«бит в секунду»* (бит/с, bits per second, bps); может также использоваться размерность *«байт в секунду»* и производные от нее величины (кб/с, Мб/с и пр.).

Для измерения скорости передачи информации также применяется единица, называемая *«бод»* (baud). Однако в бодах измеряется не собственно скорость передачи информации, а скорость изменения информационного параметра, являющегося носителем передаваемой информации. В частном случае (при синхронной двоичной передаче) скорость в бодах может быть равна скорости в битах в секунду. Однако, например, в современных модемах при одном изменении уровня несущего сигнала может передаваться больше одного бита информации (например, 4 бита), тогда скорости 2400 бод соответствует скорость передачи информации 9600 бит/с.

Не следует путать эти две величины: бод и бит/с!

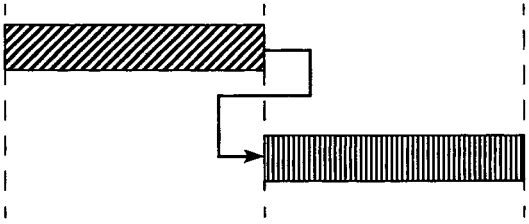
Диаграммы процессов (сетевые диаграммы, диаграммы Ганта)

В некоторых задачах, связанных с процессами передачи информации (особенно в случаях, когда один процесс начинается по истечении заданного времени после начала другого) можно существенно облегчить их решение благодаря его наглядному представлению с помощью *диаграмм процессов*. Такие диаграммы также называют *диаграммами Ганта* — по имени их изобретателя, американского инженера, механика и специалиста по менеджменту Генри Лоуренса Ганта.

Типичная диаграмма Ганта представляет собой отрезки или прямоугольные полоски, размещённые вдоль горизонтальной шкалы времени, где каждый отрезок соответствует отдельной задаче (подзадаче) или процессу. Начало, конец и длина каждого такого отрезка соответствуют началу, концу и длительности соответствующего процесса, а сами такие отрезки обычно располагаются друг за другом со сдвигом по вертикали.

В современных диаграммах Ганта, построенных при помощи специальных программ — *систем управления проектами*, кроме временны́х зависимостей, также

отображаются зависимости (связи) между задачами. Например, самым распространённым типом такой зависимости является связь «Окончание — Начало», когда очередная задача начинается после окончания предыдущей:



Для некоторых задач удобно рисовать подобную диаграмму, изображающую два процесса (или более) и размечая на ней временны́е отметки их начал и окончаний. Применение диаграммы Ганта для решения задач будет рассмотрено ниже.

Разбор типовых задач

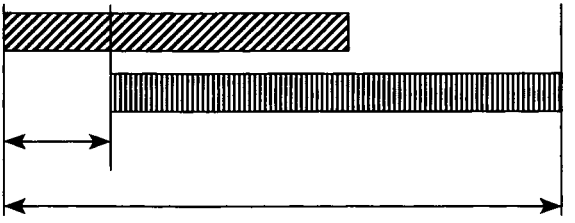
Задача 1*. У Кати есть доступ в Интернет по высокоскоростному одностороннему радиоканалу, обеспечивающему скорость получения информации 2^{20} бит в секунду. У Сергея нет скоростного доступа в Интернет, но есть возможность получать информацию от Кати по телефонному каналу со средней скоростью 2^{13} бит в секунду. Сергей договорился с Катей, что она скачает для него данные объёмом 9 Мбайт по высокоскоростному каналу и ретранслирует их Сергею по низкоскоростному каналу.

Компьютер Кати может начать ретрансляцию данных не раньше, чем им будут получены первые 1024 Кбайт этих данных. Каков минимально возможный промежуток времени (в секундах) с момента начала скачивания Катей данных до полного их получения Сергеем?

Решение

Создаётся набросок сетевой диаграммы, соответствующей условию задачи. Рассматриваются два процесса передачи, осуществляемые с разной скоростью, из

которых один процесс начинается спустя заданное время после начала другого. Общий вид диаграммы имеет вид:



Процесс 1: компьютер Кати скачивает файл объёмом в 9 Мб ($= 9 \cdot 2^{20}$ байт $= 9 \cdot 2^{23}$ бит) со скоростью 2^{20} бит/с.

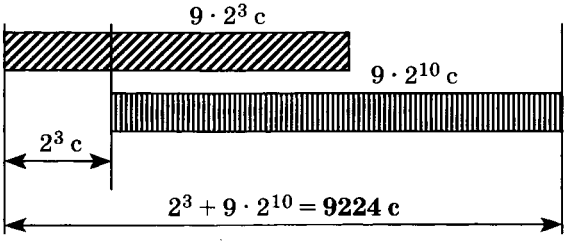
Длительность процесса 1: $9 \cdot 2^{23} / 2^{20} = 9 \cdot 2^3$ с.

Процесс 2: компьютер Сергея скачивает файл объёмом в 9 Мб ($=9 \cdot 2^{23}$ бит) со скоростью 2^{13} бит/с.

Длительность процесса 2: $9 \cdot 2^{23} / 2^{13} = 9 \cdot 2^{10}$ с.

Начало процесса 2 — через время, равное времени скачивания со скоростью 2^{20} бит/с информации объёмом 1024 кб ($= 1024 \cdot 2^{10}$ байт $= 2^{10} \cdot 2^{10}$ байт $= 2^{20}$ байт $= 2^{23}$ бит), т.е. через $2^{23} / 2^{20} = 2^3$ с.

Сетевая диаграмма с надписанными результатами расчётов:



Благодаря сетевой диаграмме нетрудно определить, какие величины нужно суммировать для вычисления общей длительности процесса (т.е. времени, прошедшего с момента начала скачивания Катей данных до полного их получения Сергеем).

Ответ: 9224 с.

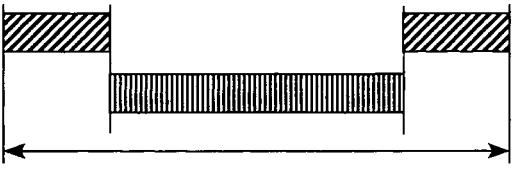
Следует не забывать приводить все величины, указанные в условии задачи, к одной размерности! Например, если скорость передачи информации задана в битах в секунду, то все значения объёмов информации нужно преобразовать в биты.

Задача 2. Лена скачивает дистрибутив ОС Linux с зарубежного сайта-репозитория, пользуясь односторонним каналом цифровой передачи данных через телевизионное эфирное вещание, обеспечивающим приём информации со скоростью 4 Мбит/с. При этом информация передаётся фрагментами по 10 Мбайт. Для начала передачи каждого фрагмента компьютер Лены должен отправить на сервер сообщение-запрос объёмом 32 кбайт, а после получения фрагмента подтвердить его безошибочный приём отдельным сообщением объёмом 16 кбайт. Для отправки таких сообщений Лена пользуется радиомодемом GPRS, который обеспечивает скорость передачи информации 128 кбит/с. Определить минимально возможное время, за которое Лена сможет скачать файл дистрибутива объёмом 350 Мбайт.

Решение

Принцип решения данной задачи аналогичен предыдущей.

Общий вид сетевой диаграммы:



Процесс 1: передача сообщения-запроса объёмом 32 Кбайт ($= 2^5 \cdot 2^{10}$ байт $= 2^5 \cdot 2^{13}$ бит) через радиомодем GPRS со скоростью 128 Кбит/с ($= 2^7 \cdot 2^{10}$ бит/с).

Длительность процесса 1: $2^5 \cdot 2^{13} / 2^7 \cdot 2^{10} = 2^{18} / 2^{17} = 2$ с.

Процесс 2: приём очередного фрагмента файла объёмом 10 Мбайт ($= 10 \cdot 2^{20}$ байт $= 10 \cdot 2^{23}$ бит) через телевизионное эфирное вещание со скоростью 4 Мбит/с ($= 2^2 \cdot 2^{20}$ бит/с).

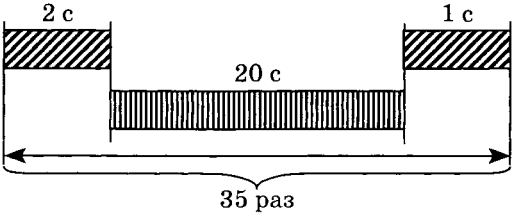
Длительность процесса 2: $10 \cdot 2^{23} / 2^2 \cdot 2^{20} = 10 \cdot 2^{23} / 2^{22} = 20$ с.

Процесс 3: передача сообщения-подтверждения объёмом 16 Кбайт ($= 2^4 \cdot 2^{10}$ байт $= 2^4 \cdot 2^{13}$ бит) через радиомодем GPRS со скоростью 128 Кбит/с ($= 2^7 \cdot 2^{10}$ бит/с).

Длительность процесса 1: $2^4 \cdot 2^{13} / 2^7 \cdot 2^{10} = 2^{17} / 2^{17} = 1$ с.

Файл объёмом 350 Мбайт принимается порциями по 10 Мбайт, следовательно, весь процесс приёма информации состоит из 35 элементарных процессов, рассмотренных выше (передача команд + приём очередного фрагмента).

Сетевая диаграмма с надписанными результатами расчётов:



Итого для приёма всего файла потребуется $35 \cdot 23 = 805$ с.

Ответ: 805 с.

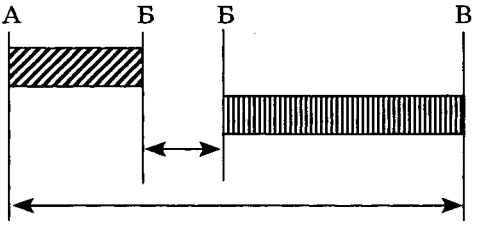
Лучше разделить описываемый процесс передачи информации на отдельные повторяющиеся этапы, чтобы вначале определить длительность отдельного такого этапа, а затем вычислить общую длительность процесса.

Задача 3. Между городами Ачинск (А) и Веденеево (В) нет прямого канала связи. Данные из Ачинска в Веденеево передаются через промежуточный шлюз в городе Беляево (Б) с помощью каналов связи А-Б и Б-В. Скорость передачи данных по каналу Б-В в 3 раза ниже, чем скорость передачи данных по каналу А-Б. Передача данных из Б в В начинается через 8 секунд после того, как закончился приём этих данных в Б. Сколько времени (в секундах) потребуется для передачи из А в В пакета данных, если для передачи этого пакета из А в Б требуется 45 сек?

В ответе укажите только число, слово «секунд» или букву «с» добавлять не нужно.

Решение

Общий вид сетевой диаграммы:



Процесс 1: передача данных по каналу А–Б со скоростью 3 скорости передачи данных по каналу Б–В.

Длительность процесса 1: 45 с

Процесс 2: передача данных по каналу Б-В со скоростью v (неизвестная величина).

Начало процесса 2 — через 8 с после завершения передачи информации по каналу А–Б.

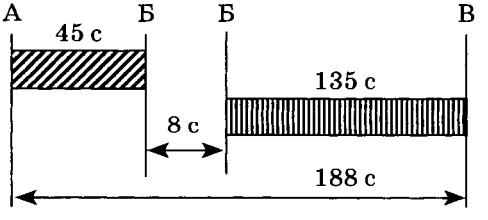
1. Объём передаваемых данных X определяется из условия, что длительность передачи данных (45 с) по каналу А–Б достигается при скорости передачи $3v$:

$$X = 45 \cdot v \cdot 3 = 135v.$$

2. Тогда длительность передачи данных по каналу Б–В определяется уравнением:

$$T_2 = X / v = 135 \text{ с}.$$

Сетевая диаграмма с надписанными результатами расчётов:



Общая длительность процесса: 45 + 8 + 135 = 188 с.

Ответ: 188 с.

Задача 4. Документ объёмом 8 Мбайт можно передать с одного компьютера на другой двумя способами:

А) сжать архиватором, передать архив по каналу связи, распаковать;

Б) передать по каналу связи без использования архиватора.

Какой способ быстрее и насколько, если:

- средняя скорость передачи данных по каналу связи составляет 2^{15} бит в секунду;
- объём сжатого архиватором документа равен 40% от исходного;
- время, требуемое на сжатие документа — 6 секунд, на распаковку — 3 секунды?

В ответе напишите букву А, если способ А быстрее, или Б, если быстрее способ Б. Сразу после буквы напишите, на сколько секунд один способ быстрее другого.

Например, если способ Б быстрее способа А на 23 секунды, в ответе нужно написать Б23.

Решение (вариант 1)

В данной задаче не требуется строить сетевую диаграмму, — только расчёты нужно выполнить дважды, для обоих предложенных вариантов (А и Б).

Вариант А:

Предполагается, что исходный массив информации сначала архивируется (6 секунд). Затем он в уже упакованном виде (40% от исходного объёма) передаётся по каналу связи с заданной средней скоростью, а после этого распаковывается (3 секунды).

Записывается соответствующая цепочка вычислений:

$$6 + (0{,}4 \cdot 8 \cdot 2^{20} \cdot 2^3 / 2^{15}) + 3$$

мегабайты — 8, 2^{20}

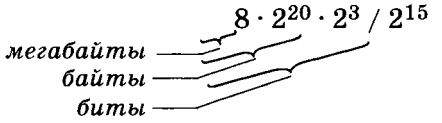
байты — 2^3

биты — 2^{15}

После некоторого количества арифметических операций вычисляется длительность всего процесса в секундах.

Вариант Б:

Аналогичным образом записываются вычисления времени передачи неупакованного массива информации по каналу связи:



Выполнив вычисления и сравнив результат с полученным для варианта А, определяется ответ к задаче.

Решение (вариант 2)

Более экономичный способ решения задачи — вычисления выполняются только один раз.

Сравнивая предлагаемые варианты (А и Б) получается, что 40% от исходного объёма массива информации приходится передавать по каналу связи *в обоих случаях*. Различие же состоит в следующем.

В варианте А, кроме этих 40% объёма файла, по каналу связи больше ничего передавать не требуется, но зато имеются «накладные расходы» времени на упаковку/распаковку в количестве 6 + 3 = 9 секунд.

В варианте Б этих дополнительных расходов времени нет, но зато приходится передавать по каналу связи оставшиеся 60% объёма файла.

Получается, что для решения задачи достаточно сравнить между собой затраты времени именно на эти различающиеся в вариантах А и Б операции:

9 секунд на упаковку/распаковку	VS	передача 60% объёма файла

Другими словами, достаточно вычислить время передачи по каналу связи с заданной скоростью 60% исходного объёма информации и вычесть из этого времени 9 секунд:

- если результат отрицателен, то быстрее способ Б;
- если результат положителен, то быстрее способ А;
- если результат равен нулю, то оба способа равнозначны по скорости;

- абсолютное значение (модуль) разности — это и есть искомое время, на которое один способ быстрее другого.

  В данном случае:

$$0{,}6 \cdot 8 \cdot 2^{20} \cdot 2^3 / 2^{15} - 9 = 0{,}6 \cdot 8 \cdot 2^{23} / 2^{15} - 9 =$$
$$= 4{,}8 \cdot 2^8 - 9 = 4{,}8 \cdot 256 - 9 = 1219{,}8 \text{ с.}$$

Ответ: **А1219,8** (способ А быстрее способа Б на 1219,8 секунды).

Рекомендуется большие числа преобразовывать в произведения некоторой константы (не кратной 2) на соответствующую степень двойки. Это облегчает выполнение операций умножения и деления (благодаря возможности сокращения двоек) и уменьшает вероятность ошибок в вычислениях.

Задача 5. Одной из технологий, лежащих в основе функционирования сети Интернет, является пакетная передача данных. Каждый файл перед его пересылкой разбивается на отдельные фрагменты (кадры). Каждый кадр пересылается отдельно, после приёма проверяется на наличие ошибок и в зависимости от результатов проверки принимающий компьютер отправляет передающему запрос на повторную передачу ошибочного кадра либо подтверждение о безошибочном приёме кадра. После безошибочного приёма всех кадров они вновь соединяются в единый файл.

В используемом варианте сетевого протокола каждый кадр содержит в себе 512 байт информации из передаваемого файла плюс 128 байт служебной информации (IP-адреса отправителя и получателя, контрольная сумма и т.д.). Последний передаваемый кадр может быть неполным и содержать менее 512 байт информации из передаваемого файла при том же объёме добавляемой служебной информации. Подтверждение о безошибочном приёме кадра имеет объём 32 байта (служебная информация). Канал передачи информации является настолько надежным, что все кадры передаются без ошибок.

Требуется передать файл объёмом 4000 байт. Канал связи является асинхронным: в направлении от источника к получателю файла достигается скорость 10 кбит/с, а в обратном направлении — скорость 2 кбит/с.

Определить минимальное время, за которое может быть передан весь файл.

Решение

1. Файл разбивается на фрагменты по 512 байт (последний фрагмент может быть неполным):

4000 = 7 * 512 + 416.

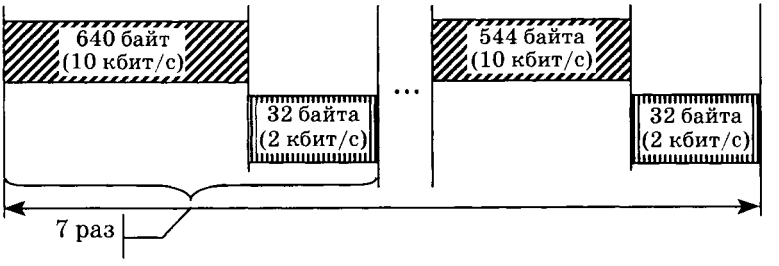
Итого имеются 8 кадров, из которых 8-й неполный.

2. Объём полного кадра равен: 512 + 128 = 640 байт. Объём последнего (неполного) кадра равен:

416 + 128 = 544 байта.

3. Весь процесс состоит из 7 повторений элементарных процессов: передача полного кадра со скоростью 10 кбит/с + обратная передача подтверждения (32 байта) со скоростью 2 кбит/с и из одного элементарного процесса: передача неполного кадра со скоростью 10 кбит/с + обратная передача подтверждения (32 байта) со скоростью 2 кбит/с.

Общий вид сетевой диаграммы:



Вычисления:

$$7 \cdot \left(\frac{640 \cdot 2^3}{10 \cdot 2^{10}} + \frac{32 \cdot 2^3}{2 \cdot 2^{10}} \right) + \left(\frac{544 \cdot 2^3}{10 \cdot 2^{10}} + \frac{32 \cdot 2^3}{2 \cdot 2^{10}} \right) = 4{,}925 \text{ с.}$$

Ответ: 4,925 с.

Раздел 2. **Моделирование и компьютерный эксперимент**

A2, B9 **Задачи на графах**

Конспект

Граф — это один из способов графического представления информации, отражающий количество объектов изучаемой системы и взаимосвязи между ними.

Объекты, отраженные в графе, представлены в нём как **вершины (узлы)** графа, а связи между ними — как **дуги (рёбра)**. Таким образом, граф представляет собой совокупность непустого множества вершин и множества связей между вершинами.

Количество вершин графа называют его **порядком**. Количество рёбер называют **размером** графа.

Рёбрам графа могут быть сопоставлены числовые значения, которые называют **весами** рёбер. Например, вес ребра в графе, обозначающем дорожную сеть, может представлять собой длину соответствующей дороги между вершинами графа, обозначающими населённые пункты. Граф, рёбрам которого назначены значения весов, называют **взвешенным**.

Две вершины называют **концевыми вершинами (концами)** ребра, которое их соединяет. При этом говорят, что ребро **инцидентно** каждой из соединяемых им вершин, и наоборот, каждая концевая вершина называется **инцидентной** соединяющему их ребру. Две концевые вершины одного и того же ребра называют **соседними**.

Рёбра, имеющие общую концевую вершину, называют **смежными**.

Рёбра, инцидентные одной и той же паре вершин (т.е. соединяющие одну и ту же пару вершин) называют **кратными**, или **параллельными**. Граф с кратными рёбрами называют **мультиграфом**.

Ребро, концами которого является одна и та же вершина, называется **петлёй**. Граф, содержащий петли (и кратные рёбра), называют **псевдографом**.

Степенью вершины называют количество инцидентных ей (исходящих из неё) рёбер, при этом петли, замкнутые на эту вершину, входят в подсчёт дважды.

Вершина называется **изолированной**, если она не является концом ни для одного ребра. Вершина называется **висячей (листом)**, если она является концом ровно одного ребра.

Пустой граф — граф, состоящий из произвольного количества изолированных вершин (т.е. не имеющий рёбер).

Полный граф — граф, не имеющий петель и кратных рёбер, в котором каждая пара вершин соединена ребром.

Путь (цепь) в графе — конечная последовательность вершин, каждая из которых (кроме последней) соединена со следующей вершиной ребром. **Циклом** называют путь, в котором первая и последняя вершины совпадают. Путь (или цикл) называют **простым**, если рёбра в нём не повторяются. Простой путь (цикл) называют **элементарным**, если вершины в нём не повторяются.

Длиной пути (или цикла) называют количество составляющих его рёбер.

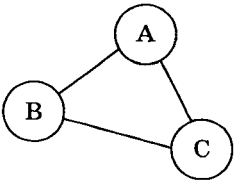
Связный граф — граф, в котором для любых двух вершин существует связывающий их путь.

Сильно связный граф — ориентированный граф, в котором существует маршрут из любой вершины в любую другую.

Неориентированный граф

В неориентированном графе связи между любыми парами концевых вершин являются двунаправленными, т.е. эти концевые вершины «равноправны» по отношению к этой связи.

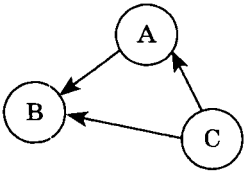
Пример неориентированного графа:



Ориентированный граф (орграф)

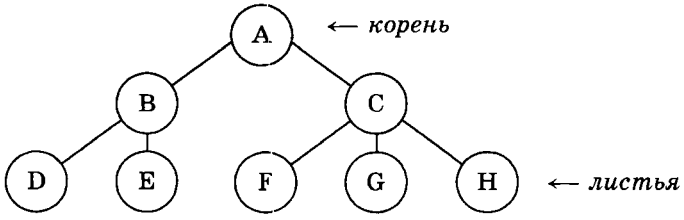
В ориентированном графе связи между концевыми вершинами являются направленными. Рёбра ориентированного графа называют **дугами**. Пути в ориентированном графе называют **ориентированными путями (маршрутами)**. Замкнутый путь (цикл) в ориентированном графе называют **контуром**. Ориентированный граф, в котором каждая пара концевых вершин связана только одной дугой, называют **направленным** (в отличие от него, в простом ориентированном графе какие-то вершины могут быть соединены двумя дугами, имеющими противоположные направления). Полный направленный граф называют **турниром**. Ориентированный граф, полученный из исходного путём смены направлений рёбер на противоположные, называют **обратным**.

Пример ориентированного графа (является направленным, турниром):



Дерево — граф, в котором существует один-единственный путь между любой парой вершин и не имеется ни одного цикла. **Ориентированное (направленное) дерево** — орграф, в котором существует один-единственный маршрут между любой парой вершин и не имеется ни одного контура. Одна из вершин дерева (его **корень**) не имеет входящих в неё дуг, а все остальные вершины имеют ровно одну входящую дугу. При этом вершины, не имеющие исходящих из них дуг, называются **листьями**.

Пример дерева:

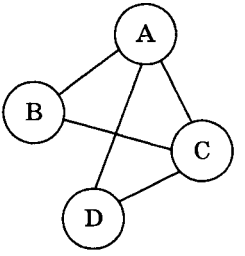


Двоичное дерево — ориентированное дерево, в котором для каждой вершины количество исходящих из неё дуг не превосходит двух.

Способы представления графов

1. **Графический способ** — изображение графа.

Пример:



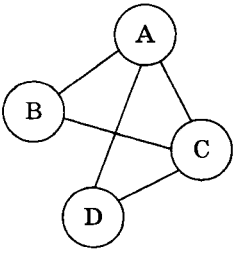
2. **Список рёбер** — перечисление всех рёбер графа как пар обозначений связываемых этими рёбрами вершин.

Пример: {A,B}, {A,D}, {A,C}, {B,C}, {C,D}

3. **Матрица смежности** — квадратная симметричная таблица (матрица), в которой и столбцы, и строки соответствуют вершинам графа, а в ячейках на их пересечении записываются числа, обозначающие наличие или отсутствие связей между соответствующими парами вершин (обычно — количество связей между вершинами).

В простейшем случае, когда граф не имеет кратных рёбер и петель, матрица смежности содержит единицы для ячеек, соответствующих парам вершин, связанных ребром, и нули — для несвязанных вершин.

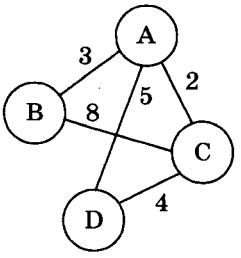
Пример:



	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

Для взвешенного графа возможен вариант матрицы смежности, где в ячейках записываются веса рёбер или нули (либо ячейки оставляются пустыми).

Пример:



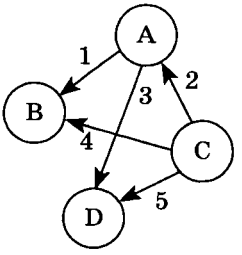
	A	B	C	D
A		3	2	5
B	3		8	
C	2	8		4
D	5		4	

4. **Матрица инцидентности** — таблица, столбцы которой соответствуют вершинам, а строки — рёбрам. При этом в ячейках на их пересечении записываются числа:

— для неориентированного графа — число 1, если данная вершина инцидентна данному ребру, или 0 — в противном случае;

— для ориентированного графа — число −1, если данная дуга исходит из данной вершины, число 1, если данная дуга входит в данную вершину, число 2, если дуга представляет собой петлю, или 0 — в противном случае.

Пример:



	A	B	C	D
1	−1	1	0	0
2	1	0	−1	0
3	−1	0	0	1
4	0	1	−1	0
5	0	0	−1	1

Разбор типовых задач

Задача 1*. Между населёнными пунктами A, B, C, D, E, F построены дороги, протяжённость которых приведена в таблице. (Отсутствие числа в таблице означает, что прямой дороги между пунктами нет.)

	A	B	C	D	E	F
A		2	4			
B	2		1		7	
C	4	1		3	4	
D			3		3	
E		7	4	3		2
F					2	

Определите длину кратчайшего пути между пунктами A и F (при условии, что передвигаться можно только по построенным дорогам).

1) 9 2) 10 3) 11 4) 12

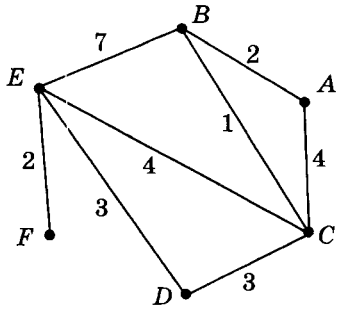
Решение

Это типичная задача на построение графов по заданной матрице смежности. Вершинами искомого графа (карты городов) являются названия городов, обозначенные буквами от *A* до *F*, а рёбра определяются наличием в таблице чисел, указывающих веса этих рёбер.

Построение такого графа — первый шаг решения задачи. Для этого достаточно разметить точки *A*, *B*, *C*, *D*, *E*, *F* и соединить линиями те из них, для которых на пересечении строки и столбца таблицы имеется непустая ячейка. Число, находящееся в этой ячейке, нужно записать над соответствующим ребром.

Поскольку граф в данном случае не является ориентированным (в условии не указано, что можно двигаться по дорогам только в одном направлении, значит, возможно и встречное движение), его матрица смежности зеркально симметрична относительно главной диагонали (выделенной серым фоном ячеек). Поэтому, при рисовании графа достаточно просмотреть, например, только ячейки над главной диагональю.

Для данной таблицы (матрицы смежности) получается граф следующего вида:



Остаётся перебрать все возможные пути от вершины *A* к вершине *F*. При этом обратить внимание, что любой такой путь заканчивается одинаково — отрезком *EF*:

ABEF — длина пути: 2 + 7 + 2 = 11;

ABCEF — длина пути: 2 + 1 + 4 + 2 = 9;

ACEF — длина пути: 4 + 4+ 2 = 10;

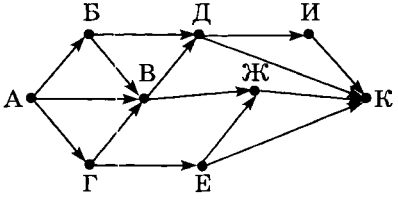
ABCDEF — длина пути: 2 + 1 + 3 + 3 + 2 = 11;

ACDEF — длина пути: 4 + 3 + 3 + 2 = 12.

Из них самый короткий — путь *ABCEF* длиной 9 единиц.

Ответ: 9 (вариант ответа №1).

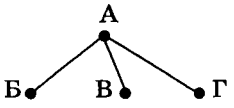
Задача 2*. На рисунке — схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?



Решение

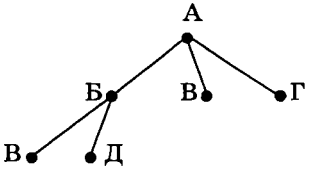
Предложенная схема дорог представляет собой ориентированный граф. Нужно проследить *все* возможные пути от вершины А до вершины К. Чтобы при этом не пропустить ни одного возможного варианта, строится второй граф — *дерево вариантов*. (Такой граф чем-то похож на деревья ходов игроков, которые требуется строить при решении задач С3, посвящённых определению выигрышных ходов в играх «в камешки».)

В самом начале нашего дерева вычерчивается «корневая» вершина А и от неё — три ветви к вершинам Б, В и Г:

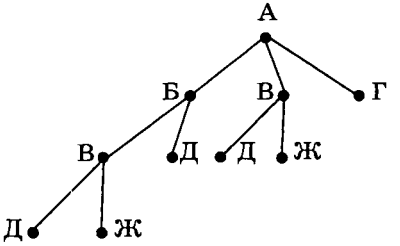


Эта же операция — выяснение, к скольким другим вершинам ведут рёбра, и вычерчивание соответствующих ветвей дерева — повторяется для каждой из получившейся концевых вершин (Б, В и Г), а затем — для новых концевых вершин, которые будут получаться на каждом новом шаге. При этом какие-то обозначения (буквы) могут повторяться в разных ветвях строящегося дерева. Построение каждой ветви прекращается, если очередной вершиной окажется вершина с буквой К — конечный (по условию задачи) пункт путешествия. Тогда эта вершина для наглядности помечается другим цветом, жирностью шрифта, обводкой и т.п.

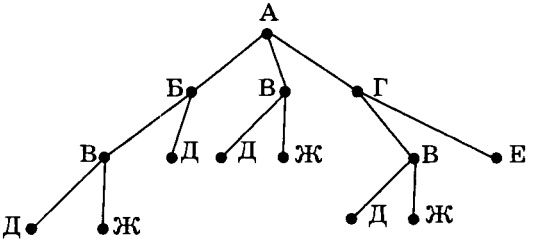
1. От вершины Б можно проехать к вершинам В и Д:



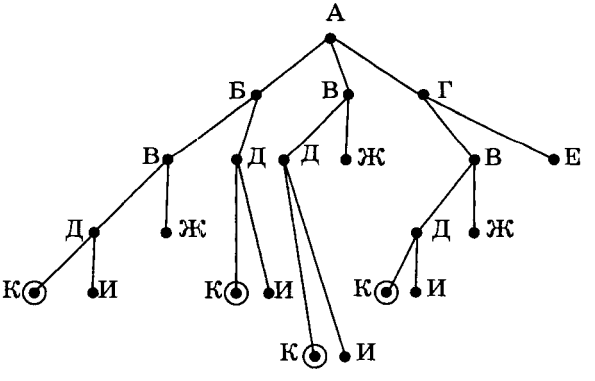
2. От вершины В можно проехать к вершинам Д и Ж, — построение ветвей повторяется дважды для обеих точек В:



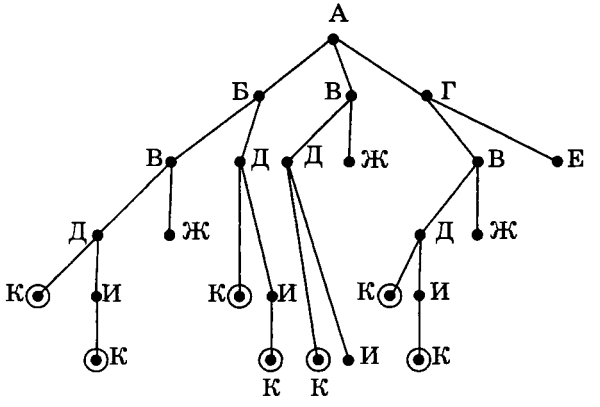
3. От вершины Г можно проехать к вершинам В и Е, при этом для точки В сразу копируются ранее построенные ветви:



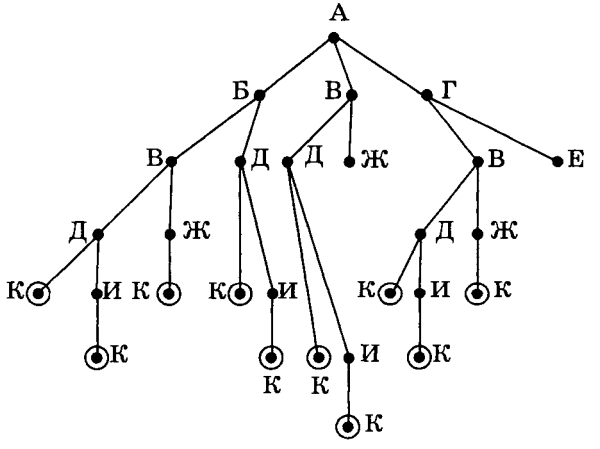
4. От вершины Д можно проехать к вершинам И и К, где К — конечная вершина:



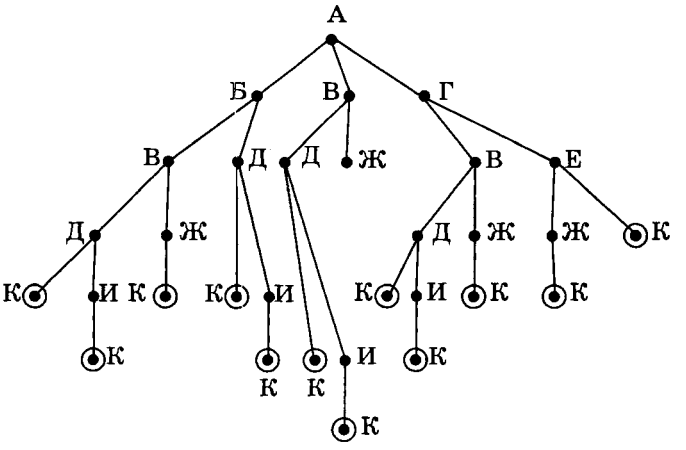
5. От вершины И можно проехать только к вершине К, т.е. достаточно вычертить только одну ветвь:



6. От вершины Ж тоже можно проехать только к вершине К:



7. Наконец, от вершины Е можно проехать к вершинам Ж и К, причём, от вершины Ж есть путь только к вершине К:



Все ветви построенного дерева завершаются конечной вершиной К, следовательно, оно содержит все возможные маршруты от А к К.

Количество возможных вариантов пути от А к К равно количеству «концевых» вершин К и равно 13 штук.

Ответ: 13 возможных вариантов пути.

Построение дерева вариантов — это универсальный метод решения задач с перебором возможных вариантов (действий, решений, ходов и пр.), обладающий высокой наглядностью и существенно уменьшающий риск пропуска возможных вариантов.

Задача 3*. Таблица стоимости перевозок устроена следующим образом: числа, стоящие на пересечениях строк и столбцов таблиц, означают стоимость проезда между соответствующими соседними станциями. Если пересечение строки и столбца пусто, то станции не являются соседними.

Стоимость проезда по маршруту складывается из стоимостей проезда между соответствующими соседними станциями.

Укажите таблицу, для которой выполняется условие: «Минимальная стоимость проезда по маршруту из E в B не более 5».

1)

	A	B	C	D	E
A		1	3		6
B	1			3	
C	3			4	
D		3	4		3
E	6			3	

3)

	A	B	C	D	E
A		3	4		7
B	3			4	
C	4				
D		4			1
E	7			1	

2)

	A	B	C	D	E
A		2	4		6
B	2			4	
C	4			2	
D		4	2		
E	6				

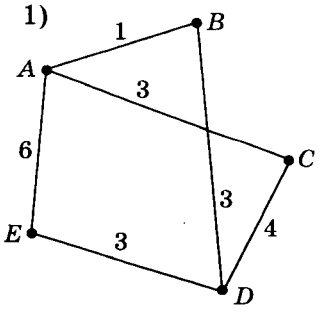
4)

	A	B	C	D	E
A		4	2		7
B	4			3	
C	2			6	
D		3	6		3
E	6			3	

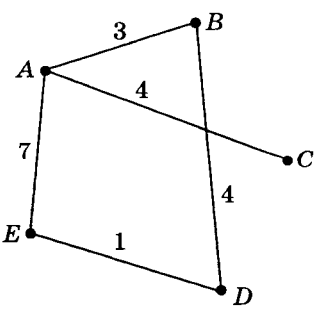
Решение

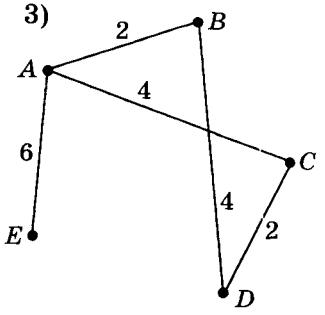
1. Поскольку графическое представление информации более наглядно, по заданным таблицам строятся соответствующие графы:

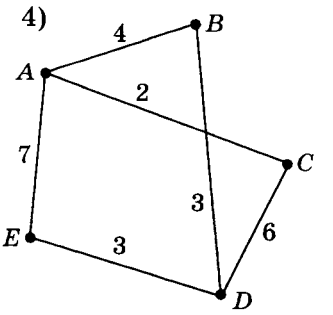
1)



2)







2. Для каждого из четырёх полученных графов перебираются все возможные пути из E в B, подсчитывая суммарную стоимость проезда по каждому маршруту:

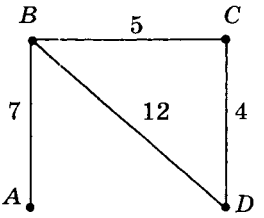
1) E-A-B (6 + 1 = 7); E-D-B (3 + 3 = 6);
   E-D-C-A-B (3 + 4 + 3 +1 = 11);
   E-A-C-D-B (6 + 3 + 4 +3 = 16);
2) E-A-B (7 + 3 = 10); E-D-B (1 + 4 = 5);
3) E-A-B (6 + 2 = 8); E-A-C-D-B (6 + 4 + 2 + 4 = 16);
4) E-A-B (7 + 4 = 11); E-D-B (3 + 3 = 6);
   E-A-C-D-B (7 + 2 + 6 + 3 = 18);
   E-D-C-A-B (3 + 6 + 2 + 4 = 15).

3. Определяется, для какого графа соблюдается условие «минимальная стоимость проезда по маршруту из E в B не больше 5»: очевидно, этому условию соответствует только вариант №2.

Ответ: вариант ответа №2.

Поскольку матрица смежности симметрична относительно её главной диагонали, достаточно анализировать только одну её часть (например, над главной диагональю).

Задача 4. Четыре города связаны дорогами, длина которых указана на схеме:



Следует определить, какие города наиболее удалены друг от друга (если считать по расстояниям между ними по дорогам). В качестве ответа указать *кратчайшее* расстояние между этими городами.

Решение (вариант 1)

Данное условие на первый взгляд кажется противоречивым: надо определить *наиболее удаленные* друг от друга города, но указать *кратчайшее* расстояние между ними. Однако это условие означает, что между каждой парой городов может быть *несколько* возможных путей проезда. Требуется определить из этих возможных маршрутов для каждой пары городов самые короткие, а затем сравнить их между собой и определить, какой из них имеет наибольшую длину.

То есть нужно найти самый длинный маршрут среди самых коротких.

1. Определяются все возможные маршруты для каждой пары городов и самые короткие среди них:

- ***AB*** — **7** (единственный, он же — самый короткий);
- ***AC***: *ABC* – (7 + 5) = 12; *ABDC* – (7 + 12 + 4) = 23 (самый короткий — **12**);
- ***AD***: *ABD* – (7 + 12) = 19; *ABCD* – (7 + 5 + 4) = 16 (самый короткий — **16**);
- ***BC***: *BC* – 5; *BDC* – (12 + 4) = 16 (самый короткий — **5**);
- ***BD***: *BD* – 12; *BCD* – (5 + 4) = 9 (самый короткий — **9**);
- ***CD***: *CD* – 4; *CBD* – (5 + 12) = 17 (самый короткий — **4**).

Не всегда прямой путь — самый короткий!

2. Среди найденных кратчайших путей между каждой парой городов определяется наибольший:

AB	AC	AD	BC	BD	CD
7	12	**16**	5	9	4

Решение (вариант 2)

1. По заданному графу строится таблица. При этом если между парой городов возможно несколько путей, то в соответствующую ячейку таблицы записываются

все возможные значения длины пути. Поскольку таблица симметрична, достаточно заполнить её верхнюю часть.

	A	B	C	D
A		7	$ABC = 12$; $ABDC = 23$	$ABD = 19$; $ABCD = 16$
B			$BC = 5$; $BDC = 16$	$BD = 12$; $BCD = 9$
C				$CD = 4$; $CBD = 17$
D				

2. В каждой ячейке оставляется наименьшее из значений:

	A	B	C	D
A		7	12	**16**
B			5	9
C				4
D				

3. Выбирается наибольшее значение из числа находящихся в ячейках таблицы (отмечено жирным шрифтом).

Ответ: 16.

Раздел 3. **Системы счисления**

A1, B7 Двоичная, восьмеричная, шестнадцатеричная системы счисления. Арифметика в указанных системах счисления

Конспект

Системы счисления

Система счисления — знаковая система, позволяющая по определённым правилам записывать числа при помощи символов некоторого алфавита (*цифр*).

Позиционные системы счисления: количественные значения цифр зависят от их *позиций* (*разрядов*) в числе, что позволяет при помощи небольшого набора цифр записывать практически любые по величине числа.

Непозиционные системы счисления: значение числа получается путём суммирования (и вычитания) количественных значений цифр, не зависящих от их местоположения в числе. Пример: римская система счисления.

Римская цифра	M	D	C	L	X	V	I
Значение	1000	500	100	50	10	5	1

При расшифровке римской записи числа:

— если меньшая по значению цифра располагается *слева* от большей, то значение меньшей цифры *вычитается* из значения большей;

— если меньшая по значению цифра располагается *справа* от большей, то значение меньшей цифры *прибавляется* к значению большей;

— в числе рекомендуется вначале выделить группы цифр, в которых меньшая цифра расположена левее большей, и вести расшифровку числа в несколько этапов. Пример:

Римская запись	Десятичное число
CDXXXVII	[CD] + [XXX] + [VII] = = [[D] – [C]] + [[X] + [X] + [X]] + [[V] + [I] + [I]] = = (500 – 100) + (10 + 10 + 10) + (5 + 1 + 1) = = 400 + 30 + 7 = 437

Основание позиционной системы счисления:

— определяет изменение количественного значения («во сколько раз») при изменении положения цифры в числе на один разряд правее/левее;

— равно количеству цифр в алфавите системы счисления.

Теоретически возможно любое значение основания системы счисления, начиная с 2. Для систем счисления с основанием p, меньшим 10, в качестве знаков алфавита системы счисления используются десятичные цифры от 0 до $(p - 1)$; для систем с основанием p, большим 10, используются все 10 десятичных цифр плюс дополнительные символы (обычно — латинские заглавные буквы, начиная с «A»). На практике системы счисления с основанием больше 16 практически не используются (за исключением измерения времени и градусной меры углов, основанных на системе счисления с основанием 60).

Обычно при записи числа значение основания системы счисления записывается в виде нижнего индекса после последней цифры числа.

Примеры наиболее часто используемых систем счисления:

Система счисления	Основание (p)	Алфавит системы счисления	Пример записи числа
Двоичная	2	0, 1	101101_2
Восьмеричная	8	0, 1, 2, 3, 4, 5, 6, 7	12345_8
Десятичная	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	1234_{10}
Шестнадцатеричная	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (=10), B (=11), C (=12), D (=13), E (=14), F (=15)	$F4D9_{16}$

Формы записи чисел в различных системах счисления

Свёрнутая («обычная») форма записи числа — привычная запись числа как последовательности цифр, стоящих на своих разрядах.

Развёрнутая форма записи числа — запись числа в виде суммы произведений его цифр на основание системы счисления в степени, равной значению разряда той или иной цифры числа (для целого числа нумерация разрядов ведётся с нуля справа налево; для дробного числа нумерация разрядов ведётся от десятичной запятой влево по возрастанию, а вправо — по убыванию, при этом разряду единиц присваивается нулевой номер).

Форма (схема) Горнера — преобразованная запись развёрнутой формы, при которой за счёт использования скобок удается избавиться от возведения основания счисления в степени. Схема Горнера предполагает рекуррентные вычисления.

Примеры: а) для целых чисел:

Формы записи	Примеры чисел			
	двоичное	восьмеричное	десятичное	шестнадцатеричное
Свёрнутая	101101	12345	1234	F4D9
Развёрнутая	$1\cdot2^5+0\cdot2^4+1\cdot2^3+1\cdot2^2+0\cdot2^1+1\cdot2^0$	$1\cdot8^4+2\cdot8^3+3\cdot8^2+4\cdot8^1+5\cdot8^0$	$1\cdot10^3+2\cdot10^2+3\cdot10^1+4\cdot10^0$	$F\cdot16^3+4\cdot16^2+D\cdot16^1+9\cdot16^0=(15)\cdot16^3+4\cdot16^2+(13)\times16^1+9\cdot16^0$
Схема Горнера	$((((1\cdot2+0)\times2+1)\cdot2+1)\times2+0)\cdot2+1$	$(((1\cdot8+2)\times8+3)\cdot8+4)\cdot8+5$	$((1\cdot10+2)\times10+3)\times10+4$	$((F\cdot16+4)\cdot16+D)\cdot16+9$

б) для дробных чисел:

Формы записи	Примеры чисел	
	двоичное	десятичное
Свёрнутая	1001,11	123,45
Развёрнутая	$1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0+1\cdot2^{-1}+1\cdot2^{-2}$	$1\cdot10^2+2\cdot10^1+3\cdot10^0+4\cdot10^{-1}+5\cdot10^{-2}$

Перевод числа из недесятичной системы счисления в десятичную осуществляется путём выполнения вычислений по развернутой записи исходного числа.

Примеры:

— перевести в десятичную систему счисления число $101110{,}101_2$:

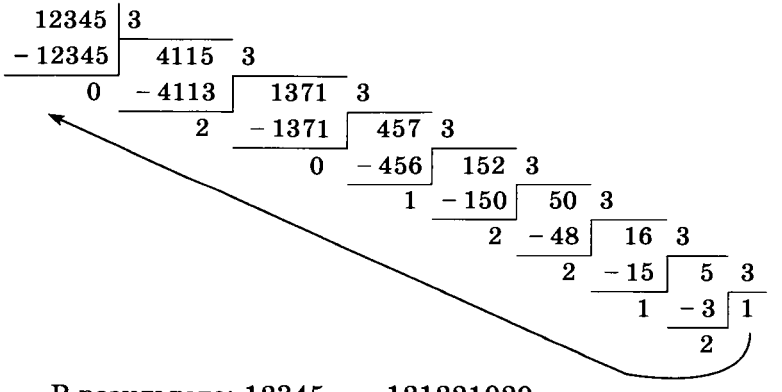
$101110{,}101_2 = 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 1\cdot32 + 0\cdot16 + 1\cdot8 + 1\cdot4 + 1\cdot2 + 0\cdot1 + 1\cdot0{,}5 + 0\cdot0{,}25 + 1\cdot0{,}125 = 32 + 8 + 4 + 2 + 0{,}5 + 0{,}125 = 46{,}625_{10}$;

— перевести в десятичную систему счисления число $F4D9{,}7_{16}$:

$F4D9{,}7_{16} = F\cdot16^3 + 4\cdot16^2 + D\cdot16^1 + 9\cdot16^0 + 7\cdot16^{-1} = (15)\cdot16^3 + 4\cdot16^2 + (13)\cdot16^1 + 9\cdot16^0 + 7\cdot16^{-1} = 15\cdot4096 + 4\cdot256 + 13\cdot16 + 9\cdot1 + 7\cdot 0{,}0625 = 61440 + 1024 + 208 + 9 + 0{,}4375 = 62681{,}4375.$

Перевод целого десятичного числа в недесятичную систему счисления выполняется путём последовательного деления числа с остатком на основание системы счисления с последующей записью полученного результата и остатков на каждом шаге деления в порядке, обратном порядку их получения. Деление производится до тех пор, пока полученный на очередном шаге результат не будет меньше основания системы счисления.

Пример: требуется перевести число 12345_{10} в троичную систему счисления:



В результате: $12345_{10} = 121221020_3$.

Перевод десятичной дроби в недесятичную систему счисления выполняется путём последовательного умножения числа на основание системы счисления с отбрасыванием получаемых целых частей на каждом шаге умножения и последующей записью полученных значений целых частей по порядку их получения. Умножение производится до получения значения с нулевой дробной частью либо до достижения необходимой точности представления дроби (необходимого количества значащих цифр после запятой).

Пример: требуется перевести число $0{,}123_{10}$ в пятеричную систему счисления с точностью до 5 значащих цифр после запятой:

0,123	×	5	=	0	,625
0,625	×	5	=	3	,125
0,125	×	5	=	0	,625
0,625	×	5	=	3	,125
0,125	×	5	=	0	,625
Получено 5 значащих цифр – деление прекращаем.					

В результате: $0{,}123_{10} \approx 0{,}03030_5 = 0{,}(03)_5$.

Представление десятичной дроби в недесятичной системе счисления, как правило, является приближенным (за исключением случаев, когда последовательность делений завершается получением нулевой дробной части либо за счёт представления дроби как периодической).

Перевод вещественного десятичного числа в недесятичную систему счисления выполняется в два этапа:

1) отдельно осуществляется перевод *целой части* числа путём последовательности *делений* на основание системы счисления;
2) отдельно выполняется перевод *дробной части* числа путём последовательности *умножений* на основание системы счисления.

Запись целой части числа в искомой системе счисления дополняется справа запятой и записью дробной части в искомой системе счисления.

Пример: требуется перевести число $15{,}12_{10}$ в пятеричную систему счисления с точностью до 5 значащих цифр после запятой:

1) $15_{10} = 30_5$;

2) $0{,}12_{10} = 0{,}03_5$.

В результате: $15{,}12_{10} = 30{,}03_5$.

Перевод чисел между недесятичными системами счисления обычно удобнее всего производить через десятичную систему счисления:

1) перевести исходное число в десятичную систему счисления;

2) перевести полученное десятичное число в требуемую систему счисления.

Перевод чисел между системами счисления с кратными основаниями. Если основания исходной и конечной системы кратны друг другу, то перевод чисел между этими системами счисления можно выполнять по упрощённой схеме.

1. ***Перевод двоичного числа в восьмеричную систему счисления*** производится по *триадам* цифр:

— исходное двоичное число разбивается на группы по три цифры («триады») справа налево; при необходимости крайняя слева группа цифр дополняется незначащими нулями слева;

— каждая триада двоичных цифр заменяется соответствующим ей восьмеричным значением согласно таблице:

Двоичная триада	000	001	010	011	100	101	110	111
Восьмеричное значение	0	1	2	3	4	5	6	7

Пример: требуется перевести число 1011010_2 в восьмеричную систему счисления:

1011010

⇓

001 011 010

⇓

1 3 2

В результате: $1011010_2 = 132_8$.

2. ***Перевод восьмеричного числа в двоичную систему счисления*** также производится по *триадам* цифр:

— исходное восьмеричное число разбивается на отдельные цифры;

— каждая восьмеричная цифра заменяется соответствующей ей триадой двоичных цифр по таблице (см. выше);

— искомое двоичное число составляется из полученных триад; незначащие нули слева отбрасываются.

Пример: требуется перевести число 12345_8 в двоичную систему счисления:

$$1\ 2\ 3\ 4\ 5$$
$$\Downarrow$$
$$001\ 010\ 011\ 100\ 101$$
$$\Downarrow$$
$$1010011100101$$

В результате: $12345_8 = 1010011100101_2$.

3. ***Перевод двоичного числа в шестнадцатеричную систему счисления*** производится по *тетрадам* цифр:

— исходное двоичное число разбивается на группы по четыре цифры («тетрады») справа налево; при необходимости крайняя слева группа цифр дополняется незначащими нулями слева;

— каждая тетрада двоичных цифр заменяется соответствующим ей шестнадцатеричным значением согласно таблице:

Двоичная триада	0000	0001	0010	0011	0100	0101	0110	0111
Шестнадцатеричное начение	0	1	2	3	4	5	6	7
Двоичная триада	1000	1001	1010	1011	1100	1101	1110	1111
Шестнадцатеричное значение	8	9	A	B	C	D	E	F

Пример: требуется перевести число 1011010_2 в шестнадцатеричную систему счисления:

1011010
⇓
0101 1010
⇓
5 A

В результате: $1011010_2 = 5A_{16}$.

4. ***Перевод шестнадцатеричного числа в двоичную систему счисления*** также производится по *тетрадам* цифр:

— исходное шестнадцатеричное число разбивается на отдельные цифры;
— каждая шестнадцатеричная цифра заменяется соответствующей ей тетрадой двоичных цифр по таблице (см. выше);
— искомое двоичное число составляется из полученных тетрад; незначащие нули слева отбрасываются.

Пример: требуется перевести число $1ADA_{16}$ в двоичную систему счисления:

1 A D A
⇓
0001 1010 1101 1010
⇓
1101011011010

В результате: $1ADA_{16} = 1101011011010_2$.

При преобразовании чисел по триадам и тетрадам не забывайте дополнять триады (тетрады) незначащими нулями слева до трёх (четырёх) знаков, а также отбрасывать незначащие нули после завершения преобразования.

Таблицы степеней

Существенную помощь при вычислениях, связанных с переводом чисел в десятичную систему счисления, могут оказать таблицы значений степеней оснований системы счисления. В качестве примера приведена такая таблица для основания системы счисления, равного 2 (рекомендуется выучить её наизусть). Аналогично можно составить подобные таблицы и для других оснований систем счисления.

n	0	1	2	3	4	5	6	7
2^n	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
Значение	1	2	4	8	16	32	64	128

n	8	9	10	20	30
2^n	2^8	2^9	2^{10}	2^{20}	2^{30}
Значение	256	516	1024	1048576	1073741824

Арифметика в недесятичных системах счисления (на примере двоичной арифметики)

Правила арифметических вычислений в общем едины для любой позиционной системы счисления. Необходима только внимательность при отслеживании переносов между разрядами, возникающих при непривычном недесятичном значении основания системы счисления.

1. ***Сложение одноразрядных двоичных чисел:***

+	**0**	**1**
0	0	1
1	1	$1^{(1)}$

Запись $^{(1)}$ обозначает перенос единицы в старший разряд.

2. ***Умножение одноразрядных двоичных чисел:***

×	**0**	**1**
0	0	0
1	0	1

3. ***Сложение многоразрядных двоичных чисел*** выполняется в столбик.

Пример: требуется вычислить значение выражения $111101_2 + 1001_2$:

```
 111   1      ← переносы
+111101
   1001
-------
1000110
```

В результате: $111101_2 + 1001_2 = 1000110_2$.

4. *Вычитание многоразрядных двоичных чисел* также выполняется в столбик, при этом возможен заём 1 из старшего разряда.

Пример: требуется вычислить значение выражения $110101_2 - 1001_2$:

```
  1          ← заём
 110101
-  1001
 ------
 101100
```

В результате: $110101_2 - 1001_2 = 101100_2$.

5. *Умножение многоразрядных двоичных чисел* выполняется в столбик аналогично умножению десятичных чисел. Однако при этом для двоичной системы счисления можно воспользоваться следующими простыми правилами:

— всегда умножать большее число на меньшее;

— умножение заменяется сложением копий множимого числа, записанных друг под другом со сдвигом каждый раз на одну позицию влево для каждого единичного разряда множителя (для нулевых разрядов множителя копия множимого в искомой сумме пропускается).

Пример: требуется вычислить значение выражения $110101_2 \times 1101_2$:

```
    110101
  ×   1101
 --------
    110101
+   110101
+ 110101          ← для 2-го разряда (нулевого) копия
 ----------         множимого пропущена
 1010110001
```

В результате: $110101_2 \times 1101_2 = 1010110001_2$.

6. *Деление многоразрядных двоичных чисел* выполняется аналогично делению в столбик десятичных чисел. При этом на каждом шаге деления выполняется последовательное вычитание делителя из очередного делимого до получения остатка, равного 0 или 1, а подсчитанное количество вычитаний записывается как очередное значение частного.

Пример: требуется вычислить значение выражения $1001011_2 / 101_2$:

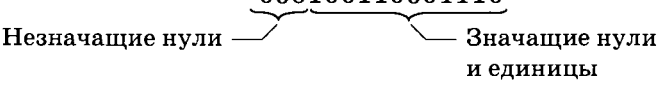
1001011	101
− 1001011	1111
0	

В результате: $1001011_2 / 101_2 = 1111_2$.

Арифметические операции в других системах счисления выполняются аналогично.

Для упрощения расчётов и исключения возможных ошибок при выполнении арифметических операций в недесятичных системах счисления рекомендуется вначале перевести все числа-операнды в десятичную систему счисления, выполнить расчёты в ней, а затем выполнить перевод результата в искомую систему счисления.

Незначащими в записи числа являются только нули, стоящие слева от числа (которые не влияют на числовое значение и могут быть отброшены). Все остальные нули и единицы в записи числа являются *значащими*!



Разбор типовых задач

Задача 1*. Сколько единиц в двоичной записи числа 1025?

1) 1 2) 2 3) 10 4) 11

Решение

1. Число 1025_{10} переводится в двоичную систему счисления: $1025_{10} = 10000000001_2$.

2. Количество единиц подсчитывается в двоичной записи числа: 2 единицы.

Ответ: 2 (вариант ответа №2).

Задача 2*. Запись числа 67_{10} в системе счисления с основанием n оканчивается на 1 и содержит 4 цифры. Чему равно основание этой системы счисления n?

Решение

Решение подобных задач основано на развёрнутой форме записи числа в системе счисления с основанием n при его переводе в десятичную систему:

$$d_{10} = a_m \cdot n^m + a_{m-1} \cdot n^{m-1} + ... + a_2 \cdot n^2 + a_1 \cdot n^1 + a_0.$$

Ещё одна форма подобной записи — *формула Горнера*:

$$d_{10} = a_m \cdot n^m + a_{m-1} \cdot n^{m-1} + ... + a_2 \cdot n^2 + a_1 \cdot n^1 + a_0 =$$
$$= (a_m \cdot n^{m-1} + a_{m-1} \cdot n^{m-2} + ... + a_2 \cdot n^1 + a_1) \cdot n + a_0 =$$
$$= ((a_m \cdot n^{m-2} + a_{m-1} \cdot n^{m-3} + ... + a_2) \cdot n + a_1) \cdot n + a_0 =$$
$$= ... = (((...(a_m \cdot n + a_{m-1}) \cdot n + ... + a_2) \cdot n + a_1) \cdot n + a_0.$$

Требуется только первый шаг преобразования «канонической» записи в формулу Горнера:

$$d_{10} = a_m \cdot n^m + a_{m-1} \cdot n^{m-1} + ... + a_2 \cdot n^2 + a_1 \cdot n^1 + a_0 =$$
$$= (a_m \cdot n^{m-1} + a_{m-1} \cdot n^{m-2} + ... + a_2 \cdot n^1 + a_1) \cdot n + a_0.$$

В этой записи: d — исходное десятичное число, a_0 — цифра, которой должна заканчиваться запись числа в искомой системе счисления, n — основание системы счисления, x — некоторый сомножитель при n:

$$d_{10} = x \cdot n + a_0.$$

Отсюда выражается искомое значение n:

$$n = \frac{d_{10} - a_0}{x}.$$

При этом неизвестную величину x можно рассматривать как некоторый параметр, на который наложено условие: x является натуральным числом (т.е. целым и положительным). Другое очевидное условие — получаемое значение n тоже должно быть натуральным числом, причём его значение должно быть больше, чем единственная указанная в условии задачи цифра записи числа в этой системе счисления.

Последовательно перебираются разные значения x, начиная с 1 и далее по возрастанию, и проверяется получаемый результат (n) на соответствие указанным вы-

ше условиям. Очевидно, что эти значения должны быть *кратны* получаемой разности $d_{10} - a_0$.

В данной задаче дано исходное десятичное число 67 и указано, что его запись должна заканчиваться цифрой 1. Следовательно:

$n = \frac{d_{10} - a_0}{x} = \frac{67 - 1}{x} = \frac{66}{x}$, т.е. требуется найти значения x, кратные 66 (те, на которые число 66 делится нацело). Это значения 1, 2, 3, 6, 11, 22 и 33 (значение $x = 66$ не подходит, так как n должно быть больше 1).

Остаётся проверить, при каких найденных значениях x исходное число 67 будет в системе с полученным основанием n иметь 4 цифры:

- $n = 66$: $67_{10} = 11_{66}$ — не годится;
- $n = 33$: $67_{10} = 21_{33}$ — не годится;
- $n = 22$: $67_{10} = 31_{22}$ — не годится;
- $n = 11$: $67_{10} = 61_{11}$ — не годится;
- $n = 6$: $67_{10} = 151_6$ — не годится;
- $n = 3$: $67_{10} = 2111_3$ — годится;
- $n = 2$: $67_{10} = 1000011_2$ — не годится.

Таким образом, условию задачи удовлетворяет только значение $n = 3$.

Ответ: 3.

Задача 3*. Дано $A = A7_{16}$, $B = 251_8$. Какое из чисел C, записанных в двоичной системе, отвечает условию $A < C < B$?

1) 10101100_2
2) 10101010_2
3) 10101011_2
4) 10101000_2

Решение

1. Используя правила перевода восьмеричных и шестнадцатеричных чисел в двоичную систему счисления по триадам и тетрадам, все заданные числа переводятся в одну и ту же систему счисления — двоичную (в которой представлены варианты ответов):

- $A = A7_{16} \Rightarrow (A_{16} = 1010_2)\,(7_{16} = 0111_2) \Rightarrow 10100111_2$;
- $B = 251_8 \Rightarrow (2_8 = 010_2)\,(5_8 = 101_2)\,(1_8 = 001_2) \Rightarrow$
  $\Rightarrow 010101001_2 \Rightarrow 10101001_2$.

2. Записываются полученные двоичные значения в заданный «шаблон» неравенства:

$\underline{1010}0111_2 < \quad < \underline{1010}1001_2$.

3. Первые четыре бита в этих числах одинаковы (выделено подчеркиванием), а последние четыре бита соответствуют в левом (меньшем) числе 0111, а в правом (большем) — 1001. Следовательно, среднее число неравенства (C) должно:

- начинаться с тех же четырёх битов 1010;
- заканчиваться четвёркой битов 1000 (единственной возможной между 0111 и 1001).

Следовательно, искомое число равно 10101000_2.

Ответ: $C = 10101000_2$ (вариант ответа № 4).

При преобразовании чисел по триадам и тетрадам следует не забывать дополнять триады (тетрады) незначащими нулями слева до трёх (четырёх) знаков, а также отбрасывать незначащие нули после завершения преобразования.

Задача 4*. В системе счисления с некоторым основанием десятичное число 18 записывается в виде 30. Укажите это основание.

Решение

Решение подобных задач основано на развёрнутой форме записи числа в системе счисления с основанием n при его переводе в десятичную систему:

$$d_{10} = a_m \cdot n^m + a_{m-1} \cdot n^{m-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n^1 + a_0.$$

Неизвестное основание системы счисления обозначается буквой n и записывается уравнение на основе имеющихся у нас данных:

$$18 = 3n + 0.$$

Решая это уравнение, получается, что: $3n = 18 \Rightarrow$ $\Rightarrow n = 6$ (шестеричная система счисления).

Ответ: 6.

Задача 5*. Дано $A = 9D_{16}$, $B = 237_8$. Какое из чисел C, записанных в двоичной системе, отвечает условию $A < C < B$?

1) 10011010_2
2) 10011110_2
3) 10011111_2
4) 11011110_2

Решение

1. Используя правила перевода восьмеричных и шестнадцатеричных чисел в двоичную систему счисления по триадам и тетрадам, все заданные числа переводятся в одну и ту же систему счисления — двоичную (в которой представлены варианты ответов):

- $A = 9D_{16} \Rightarrow (9_{16} = 1001_2)\,(D_{16} = 1101_2) \Rightarrow 10011101_2$;
- $B = 237_8 \Rightarrow (2_8 = 010_2)\,(3_8 = 011_2)\,(7_8 = 111_2) \Rightarrow$ $\Rightarrow 010011111_2 \Rightarrow 10011111_2$.

2. Полученные двоичные значения записываются в заданный «шаблон» неравенства:

$\underline{10011101}_2 < \quad < \underline{10011111}_2$.

3. Первые шесть битов в этих числах одинаковы (выделено подчёркиванием), а последние два бита соответствуют в левом (меньшем) числе 01, а в правом (большем) — 11. Следовательно, среднее число неравенства (C) должно:

- начинаться с тех же шести битов 100111;
- заканчиваться парой битов 10 (единственной возможной между 01 и 11).

Следовательно, искомое число равно 10011110_2.

Ответ: $C = 10011110_2$ (вариант ответа №2).

Задача 6*. Вычислите сумму чисел X и Y, если $X = 110111_2$, $Y = 135_8$. Результат представьте в двоичном виде.

1) 11010100
2) 10100100
3) 10010011
4) 10010100

Решение

$X = 110111_2 = 55_{10}$;

$Y = 135_8 = 93_{10}$;

$X + Y = 55 + 93 = 148$;

$148_{10} = 10010100_2$.

Ответ: 10010100_2 (вариант ответа №4).

Задача 7*. В системе счисления с некоторым основанием десятичное число 49 записывается в виде 100. Укажите это основание.

Решение

Исходя из развёрнутой формы записи числа в системе счисления с основанием n при его переводе в десятичную систему:

$$d_{10} = a_m \cdot n^m + a_{m-1} \cdot n^{m-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n^1 + a_0.$$

и, обозначив неизвестное основание системы счисления буквой n, записывается уравнение на основе имеющихся данных:

$$49 = 1 \cdot n^2 + 0 \cdot n + 0.$$

В результате получается простое квадратное уравнение:

$$n^2 - 49 = 0, \text{ откуда } n^2 = 49 \Rightarrow n_1 = -7,\ n_2 = 7.$$

Очевидно, что основание системы счисления может быть только натуральным числом, поэтому первый (отрицательный) корень уравнения не подходит. Следовательно, ответом является число 7 — семеричная система счисления.

Ответ: 7.

Задача 8. Имеется десятичное число 1104. В некоторой системе счисления это число записывается тремя единицами и тремя нулями. Найти основание этой системы счисления.

Решение (вариант 1)

Основная сложность данной задачи — в том, что известны только количества нулей и единиц в искомой записи числа, но не известны их позиции в этой записи. Единственное, что можно сказать точно, — искомая запись состоит из 6 цифр, из которых три — единицы, а три — нули. Поэтому решение начинается с полной шестизначной записи числа 1104_{10} в системе с искомым основанием x:

$$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0,$$

где a_5, a_4, a_3, a_2, a_1, a_0 — цифры записи числа в системе счисления с искомым основанием x.

Первая цифра a_5 ***точно*** равна 1 (иначе запись числа не была бы шестизначной). Расположение же двух дру-

гих единиц неизвестно, поэтому требуется проверять все возможные варианты:

a_5	a_4	a_3	a_2	a_1	a_0
1	1	1	0	0	0
1	1	0	1	0	0
1	1	0	0	1	0
1	1	0	0	0	1
1	0	1	1	0	0
1	0	1	0	1	0
1	0	1	0	0	1
1	0	0	1	1	0
1	0	0	1	0	1
1	0	0	0	1	1

 Положение первой единицы определено однозначно ($a_5 = 1$). Остальные возможные варианты расположения двух единиц определяются.

Сначала предполагается, что обе единицы — это a_4 и a_3. Затем «закрепляется» положение первой единицы (a_4) и перебираются все возможные положения второй единицы. Затем меняется положение первой единицы на следующее по порядку a и перебираются все возможные положения второй единицы и т.д.

На практике может не портебоваться перебор всех решений.

1) Пусть $(a_5, a_4, a_3, a_2, a_1, a_0) = (111000)$. Тогда исходная запись

$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$,

вырождается в запись: $1104_{10} = x^5 + x^4 + x^3$.

Перебираются возможные значения x, начиная с минимально возможного (2):

- $x = 2$: $2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$ (< 1104); равенства нет, значит $x = 2$ — не подходит;
- $x = 3$: $3^5 + 3^4 + 3^3 = 243 + 81 + 27 = 351$ (< 1104); $x = 3$ — не подходит;

- $x = 4$: $4^5 + 4^4 + 4^3 = 1024 + 256 + 64 = 1344$ (> 1104); $x = 4$ — не подходит, при этом очевидно, что для бóльших значений x будут тоже получаться значения больше требуемого 1104. Значит, данный вариант расположения нулей и единиц в записи числа непригоден при ***любом*** значении x (или, что то же самое, уравнение $1104_{10} = x^5 + x^4 + x^3$ не имеет решения в натуральных числах больше 1).

2) Пусть $(a_5, a_4, a_3, a_2, a_1, a_0) = (110100)$. Тогда исходная запись

$$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0,$$

вырождается в запись: $1104_{10} = x^5 + x^4 + x^2$.

Перебираются возможные значения x, начиная с минимально возможного (2):

- $x = 2$: $2^5 + 2^4 + 2^2 = 32 + 16 + 4 = 52$ (< 1104); равенства нет, значит $x = 2$ — не подходит;
- $x = 3$: $3^5 + 3^4 + 3^2 = 243 + 81 + 9 = 333$ (< 1104); $x = 3$ — не подходит;
- $x = 4$: $4^5 + 4^4 + 4^2 = 1024 + 256 + 8 = 1288$ (> 1104); $x = 4$ — не подходит, при этом очевидно, что для бóльших значений x будут тоже получаться значения больше требуемого 1104. Значит, данный вариант непригоден при любом значении x.

3) Пусть $(a_5, a_4, a_3, a_2, a_1, a_0) = (110010)$. Тогда исходная запись

$$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0,$$

вырождается в запись: $1104_{10} = x^5 + x^4 + x$.

Перебираются возможные значения x, начиная с минимально возможного (2):

- $x = 2$: $2^5 + 2^4 + 2 = 32 + 16 + 2 = 50$ (< 1104); $x = 2$ — не подходит;
- $x = 3$: $3^5 + 3^4 + 3 = 243 + 81 + 3 = 327$ (< 1104); $x = 3$ — не подходит;
- $x = 4$: $4^5 + 4^4 + 4 = 1024 + 256 + 4 = 1284$ (> 1104); $x = 4$ — не подходит, и для бóльших значений x получаются значения больше требуемого 1104.

Значит, данный вариант непригоден при любом значении x.

4) Пусть $(a_5, a_4, a_3, a_2, a_1, a_0) = (110001)$. Тогда исходная запись

$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$,

вырождается в запись: $1104_{10} = x^5 + x^4 + 1$.

Перебираются возможные значения x, начиная с минимально возможного (2):

- $x = 2$: $2^5 + 2^4 + 1 = 32 + 16 + 1 = 49$ (< 1104); $x = 2$ — не подходит;
- $x = 3$: $3^5 + 3^4 + 1 = 243 + 81 + 1 = 325$ (< 1104); $x = 3$ — не подходит;
- $x = 4$: $4^5 + 4^4 + 1 = 1024 + 256 + 1 = 1281$ (> 1104); $x = 4$ — не подходит, и для бóльших значений x получаются значения больше требуемого 1104. Значит, данный вариант непригоден при любом значении x.

5) Пусть $(a_5, a_4, a_3, a_2, a_1, a_0) = (101100)$. Тогда исходная запись

$1104_{10} = a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$,

вырождается в запись: $1104_{10} = x^5 + x^3 + x^2$.

Перебираются возможные значения x, начиная с минимально возможного (2):

- $x = 2$: $2^5 + 2^3 + 2^2 = 32 + 8 + 4 = 44$ (< 1104); $x = 2$ — не подходит;
- $x = 3$: $3^5 + 3^3 + 3^2 = 243 + 27 + 9 = 279$ (< 1104); $x = 3$ — не подходит;
- $x = 4$: $4^5 + 4^3 + 4^2 = 1024 + 64 + 16 = 1104$.

Итак, для данного варианта расположения трёх единиц и трёх нулей в записи числа и для основания системы счисления, равного 4, получается требуемое десятичное значение 1104. Следовательно, искомое основание системы счисления равно 4.

Решение (вариант 2)

Данную задачу возможно решить «в лоб», без каких-либо логических рассуждений. Достаточно просто преобразовывать исходное десятичное число в двоич-

ную, троичную и т.д. систему счисления, в полученном результате подсчитывать количества единиц и нулей и проверять эти количества на соответствие заданному в задаче условию (три единицы, три нуля и больше никаких других цифр), пока для какого-то основания системы не будет найдено такое соответствие. Единственный недостаток такого способа решения — деление (для пересчёта из десятичной системы счисления в искомую) выполняется сложнее и чревато большим количеством ошибок, чем умножение (при вычислении десятичного числа по записи в некоторой системе счисления по первому способу).

Итак:

- основание системы счисления 2:
  $1104_{10} = 10001010000_2$ — имеется три единицы, но количество нулей слишком велико;
- основание системы счисления 3:
  $1104_{10} = 1111220_3$ — имеется четыре единицы, только один нуль и «лишние» двойки;
- основание системы счисления 4:
  $1104_{10} = 101100_4$ — имеется ровно три единицы и ровно три нуля при отсутствии других цифр, следовательно, подходит этот вариант.

Ответ: 4.

Задачи на кодирование, решаемые с применением недесятичных систем счисления

Конспект

Система счисления — знаковая система, позволяющая по определенным правилам записывать числа при помощи символов некоторого алфавита.

Как правило, символами алфавита системы счисления являются десятичные цифры. Однако это — лишь результат общепринятой договорённости; формально такими символами могут быть любые знаки, в том числе произвольные буквы (пример — использование ла-

тинских букв от A до F в качестве цифр шестнадцатеричной системы счисления).

Примеры наиболее часто используемых систем счисления:

Система счисления	Основание (p)	Алфавит системы счисления	Пример записи числа
Двоичная	2	0, 1	101101_2
Восьмеричная	8	0, 1, 2, 3, 4, 5, 6, 7	12345_8
Десятичная	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	1234_{10}
Шестнадцатеричная	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (=10), B (=11), C (=12), D (=13), E (=14), F (=15)	$F4D9_{16}$

Во многих задачах, в условиях которых не говорится о необходимости преобразования чисел в другую систему счисления или о выполнении арифметических операций в различных системах счисления, можно найти аналогию с той или иной позиционной системой счисления и этим облегчить решение задачи.

Для решения приведённых ниже задач необходимо уметь выполнять перевод чисел из десятичной системы счисления в троичную и обратно.

Перевод числа из троичной системы счисления в десятичную осуществляется путём выполнения вычислений по развёрнутой записи исходного числа.

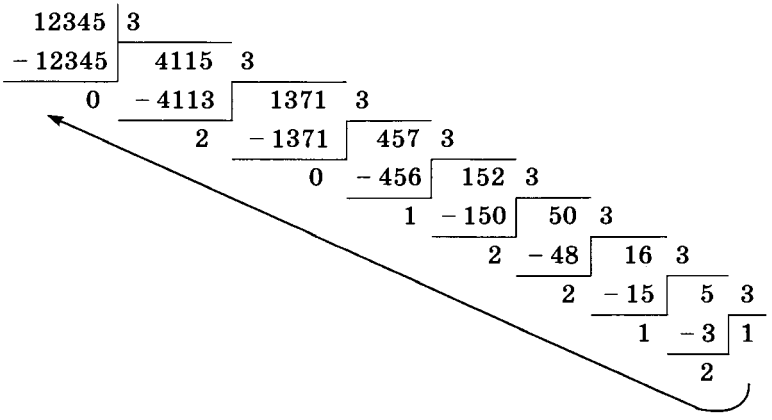
Пример. Требуется перевести в десятичную систему счисления число 12021_3:

$$12021_3 = 1 \cdot 3^4 + 2 \cdot 3^3 + 0 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 =$$
$$= 1 \cdot 81 + 2 \cdot 27 + 0 \cdot 9 + 2 \cdot 3 + 1 \cdot 1 =$$
$$= 81 + 54 + 6 + 1 = 142_{10}.$$

Перевод целого десятичного числа в троичную систему счисления выполняется путём последовательного деления с остатком числа на основание системы счисления (3) с последующей записью полученного результата и остатков на каждом шаге деления в порядке, обрат-

ном порядку их получения. Деление производится до тех пор, пока полученный на очередном шаге результат не будет меньше основания системы счисления.

Пример: требуется перевести число 12345_{10} в троичную систему счисления:



В результате: $12345_{10} = 121221020_3$.

Разбор типовых задач

Задача 1*. Все 5-буквенные слова, составленные из букв А, О, У, записаны в алфавитном порядке.

Вот начало списка:

1. ААААА
2. ААААО
3. ААААУ
4. АААОА

……

Запишите слово, которое стоит на **240-м месте** от начала списка.

Решение

Поскольку слова начинаются с букв А и их список начинается с ААААА, а далее идут слова ААААО, ААААУ, АААОА и т.д., буквы А, О, У сопоставляются цифрам: А – 0, О – 1, У – 2. В результате исходная задача сводится к следующей:

Все 5-значные числа, составленные из цифр 0, 1, 2, записаны в порядке возрастания. Вот начало списка:

1. 00000
2. 00001
3. 00002
4. 00010
.....

Какое число стоит на 240-м месте в списке?

Чтобы определить число, стоящее в списке на 240-й позиции, нужно учесть «рассогласование» между значениями чисел и их порядковыми номерами в списке:

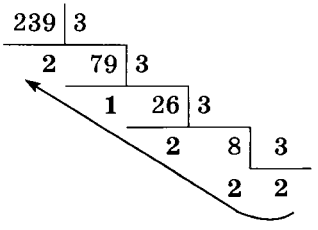
- последовательность чисел начинается с значения $00000_3 = 0_{10}$;
- нумерация чисел в списке начинается с единицы.

Тогда для определения числа, стоящего в 240-й позиции, составляется следующее соответствие:

Порядковый номер	Значение числа
1	0
240	(240 – 1)

Очевидно, что на 240-м месте в списке стоит десятичное число 239 (так как значение числа на 1 меньше, чем порядковый номер).

Пятиразрядная запись этого числа в троичной системе счисления определяется:



В результате, на 240-м месте в списке находится пятиразрядное троичное число **22212**.

Преобразуя это число к записи слова, кодируемого буквами А, О, У по правилу соответствия: А–0, О–1, У–2, получается слово: **УУУОУ**.

Ответ: **УУУОУ**.

Задача 2. Все 5-буквенные слова составлены из букв К, О, Т. Вот начало списка:

1. ТТТОК
2. ТТТКТ
3. ТТТКО
4. ТТТКК
5. ТТОТТ

……

На каком месте списка находится слово КОТОК?

Решение

В приведённом списке после слова ТТТОК идёт слово ТТТКТ, потом ТТТКО, затем — ТТТКК и ТТОТТ. Тогда сопоставив буквы К, О, Т цифрам: Т–0, О–1, К–2, условие задачи переписывается в виде:

Все 5-значные числа составлены из цифр 0, 1, 2. Вот начало списка:

1. 00012
2. 00020
3. 00021
4. 00022
5. 00100

……

На каком месте списка находится число 21012?

Число 21012 записано в троичной системе счисления. После преобразования его в десятичное:

$$21012_3 = 2 \cdot 3^4 + 1 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0 =$$
$$= 2 \cdot 81 + 27 + 3 + 2 = 192.$$

Чтобы определить номер позиции в списке для числа 192, учитывается «рассогласование» между значениями чисел и их порядковыми номерами в списке:

- последовательность чисел начинается с $12_3 = 5$;
- нумерация чисел в списке начинается с единицы.

Для определения номера позиции числа 192 составляется соответствие:

Порядковый номер	**Значение числа**
1 (= 5 – 4)	5
(192 – 4)	192

Очевидно, что десятичное число 192 будет стоять в списке на 188-м месте (так как значение числа на 4 больше, чем порядковый номер).

Ответ: на 188-м месте.

Задача 3*. Строки (цепочки латинских букв) создаются по следующему правилу.

Первая строка состоит из одного символа — латинской буквы «A». Каждая из последующих цепочек создается такими действиями: в очередную строку сначала записывается буква, чей порядковый номер в алфавите соответствует номеру строки (на i-м шаге пишется i-я буква алфавита), а затем к ней слева дважды подряд приписывается предыдущая строка.

Вот первые 4 строки, созданные по этому правилу:

(1) A

(2) AAB

(3) AABAABC

(4) AABAABCAABAABCD

Латинский алфавит (для справки):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Запишите шесть символов подряд, стоящие в седьмой строке со 117-го по 122-е место (считая слева направо).

Решение

Определяется формула для вычисления длины итоговой цепочки, выписывая длины первых цепочек, приведённых в условии:

№ цепочки	Цепочка	Длина цепочки
1	A	1
2	AAB	3
3	AABAABC	7
4	AABAABCAABAABCD	15

Искомая формула:

$$<\textit{длина цепочки}> = 2^i - 1$$

(прибавление единицы в показателе степени двойки не требуется, так как нумерация цепочек производится не с нуля, а с единицы).

Длина итоговой, 7-й цепочки будет равна $2^7 - 1 = 127$ символов.

Теперь цепочка «расплетается» по шагам назад.

1. Согласно правилам, указанным в условии задачи, итоговая, 7-я цепочка имеет формат:

(6)	(6)	G

При этом, согласно выведенной формуле, 6-я цепочка имеет длину $2^6 - 1 = 63$ символа. В таблицу, обозначающую формат рассматриваемой строки-цепочки, добавляются обозначения соответствующих номеров позиций символов[1]:

(6)	(6)	G
1–63	64–126	127

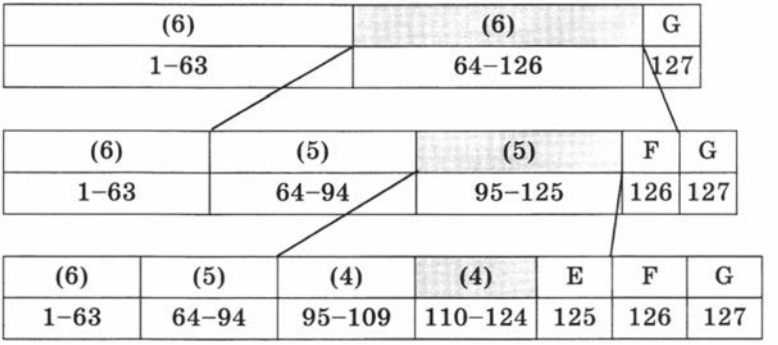
2. Искомые символы располагаются во второй по счёту подстроке (6). Продолжается «расплетание» цепочек ещё на один шаг назад, «раскрывая» эту вторую цепочку (6) и оставляя неизменными остальные части таблицы. При этом учитывается, что длина предыдущей, 5-й цепочки равна $2^5 - 1 = 31$ символу.

(6)	(6)	G
1–63	64–126	127

(6)	(5)	(5)	F	G
1–63	64–94	95–125	126	127

[1] Следует помнить, что отсчёт номеров позиций (как и отсчёт дней по календарю) ведётся по следующим правилам: если предыдущая подцепочка начинается с символа с порядковым номером n и имеет длину d, то номер символа, которым заканчивается эта подцепочка, вычисляется по формуле $(n + d - 1)$, а следующая подцепочка будет начинаться с символа под номером $(n + d)$.

3. Искомые символы находятся во второй «раскрытой» цепочке (5). Вычисляется длина цепочки (4): $2^4 - 1 = 15$ символов.

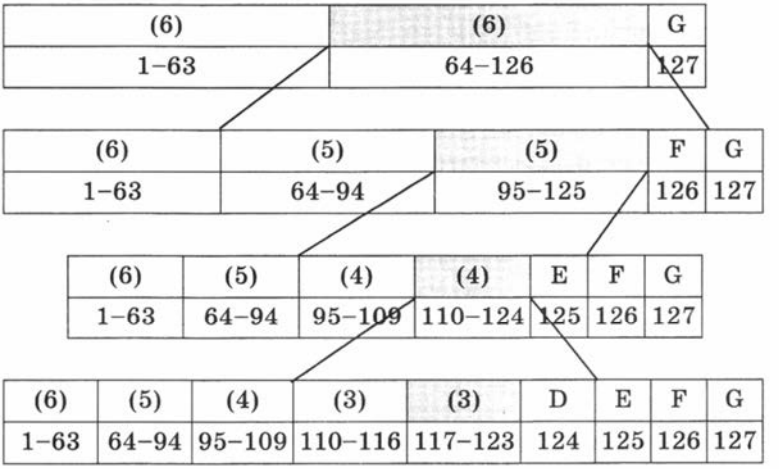


(6)	(6)	G
1–63	64–126	127

(6)	(5)	(5)	F	G
1–63	64–94	95–125	126	127

(6)	(5)	(4)	(4)	E	F	G
1–63	64–94	95–109	110–124	125	126	127

4. Искомые символы находятся во второй «раскрытой» цепочке (4). Вычисляется длина цепочки (3): $2^3 - 1 = 7$ символов).



(6)	(6)	G
1–63	64–126	127

(6)	(5)	(5)	F	G
1–63	64–94	95–125	126	127

(6)	(5)	(4)	(4)	E	F	G
1–63	64–94	95–109	110–124	125	126	127

(6)	(5)	(4)	(3)	(3)	D	E	F	G
1–63	64–94	95–109	110–116	117–123	124	125	126	127

5. Получается, что все нужные символы находятся в одной подцепочке (вторая (3)), которую совсем нетрудно выписать полностью, «подсмотрев» её в условии задачи: AABAABC.

A	A	A	A	A	B	C
117	118	119	120	121	122	123

Искомые символы, записанные на позициях 117–122, — это AABAAB.

Если бы искомые символы оказались бы распределены между несколькими подцепочками, то потребовалось бы или «расплести» эти подцепочки, или сразу их выписать и найти искомое.

Ответ: искомые символы **AABAAB**.

Задача 4*. Строки (цепочки символов латинских букв) создаются по следующему правилу.

Первая строка состоит из одного символа — латинской буквы «A». Каждая из последующих цепочек создается такими действиями: в очередную строку сначала записывается буква, чей порядковый номер в алфавите соответствует номеру строки (на i-м шаге пишется i-я буква алфавита), а затем к ней слева дважды подряд приписывается предыдущая строка.

Вот первые 4 строки, созданные по этому правилу:

(1) A

(2) AAB

(3) AABAABC

(4) AABAABCAABAABCD

Латинский алфавит (для справки):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Имеется задание: «Определить символ, стоящий в n-й строке на позиции $2^{n-1} - 5$, считая от левого края цепочки». Выполните это задание для $n = 8$.

Решение

При $n = 8$ задание выглядит следующим образом: «Определить символ, стоящий в **8**-й строке на позиции $2^7 - 5 = \mathbf{123}$, считая от левого края цепочки».

Тогда, аналогично предыдущей задаче:

$$<\textit{длина цепочки}> = 2^i - 1$$

(прибавление единицы в показателе степени двойки здесь не требуется, так как нумерация цепочек производится не с нуля, а с единицы.)

Длина итоговой, 8-й цепочки равна $2^8 - 1 = 255$ символов.

1. Согласно правилам, указанным в условии задачи, итоговая, 8-я цепочка имеет формат:

(7)	(7)	H

2. Согласно выведенной формуле 7-я цепочка имеет длину $2^7 - 1 = 127$ символов. Это означает, что искомый символ находится почти в конце первой из двух составляющих 7-х цепочек.

Далее, легко заметить, что правила составления цепочек определяют, что каждая n-я цепочка всегда завершается последовательностью из n букв алфавита начиная с A. Тогда седьмая цепочка завершается последовательностью из 7 букв: ABCDEFG. Причём искомый символ — один из них.

Пронумеровав символы, стоящие в конце рассматриваемой цепочки, получается:

...	A	B	C	D	E	F	G
...	121	122	123	124	125	126	127

Отсюда легко определить, что на 123-м месте находится символ C.

Ответ: C.

Раздел 4. **Основы логики**

(A3, A10) Таблицы истинности. Законы алгебры логики. Задачи, решаемые с использованием таблиц истинности

Конспект

В *алгебре логики* изучаются логические операции, производимые над *высказываниями*. Такие высказывания могут быть истинными или ложными. Применяя к простым высказываниям *логические операции*, можно строить *составные высказывания*.

Основные логические операции

- *Отрицание* (инверсия, логическое НЕ)

Смысл операции: результат меняется на противоположный (вместо истины — ложь, вместо лжи — истина).

Обозначение: ¬.

Таблица истинности:

A	¬A
0	1
1	0

- *Логическое сложение* (дизъюнкция, логическое ИЛИ)

Смысл операции: результат — истина, если хотя бы один операнд — истина (операндом называется то значение или та переменная, над которым (которой) осуществляется операция).

Обозначения: ∨ или +.

Таблица истинности:

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

- *Логическое умножение* (конъюнкция, логическое И)

Смысл операции: результат — истина, если оба операнда — истина.

Обозначения: $\wedge$ или &.

Таблица истинности:

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

- *Исключающее ИЛИ* (сложение по модулю 2, строгая дизъюнкция)

Смысл операции: результат — истина, если операнды различны.

Обозначения: $\oplus$ или $\neq$.

Таблица истинности:

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

- *Следование* (импликация)

Смысл операции: из лжи может следовать что угодно, а из истины— только истина.

Обозначение: $\rightarrow$.

Таблица истинности:

A	B	A $\rightarrow$ B
0	0	1
0	1	1
1	0	0
1	1	1

- *Равносильность* (эквиваленция)

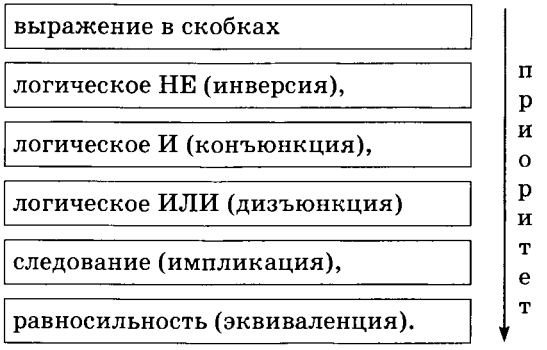
Смысл операции: результат — истина, если операнды одинаковы.

Обозначения: ≡ или ↔.

Таблица истинности:

A	B	A ↔ B
0	0	1
0	1	0
1	0	0
1	1	1

Если в логическом выражении используется несколько логических операций, то их порядок определяется *приоритетами логических операций:*



Операцию «импликация» можно выразить через «ИЛИ» и «НЕ»:

$$A \to B = \neg A \vee B.$$

Операцию «эквиваленция» также можно выразить через «ИЛИ» и «НЕ»:

$$A \leftrightarrow B = \neg A \wedge \neg B \vee A \wedge B.$$

Основные законы алгебры логики

Законы коммутативности	$A \vee B = B \vee A$, $A \wedge B = B \wedge A$
Законы ассоциативности	$(A \vee B) \vee C = A \vee (B \vee C)$, $(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Окончание табл.

Законы дистрибутивности	$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$, $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
Закон непротиворечия (высказывание не может быть одновременно истинным и ложным)	$A \wedge \neg A = 0$
Закон исключения третьего (либо высказывание, либо его отрицание должно быть истинным)	$A \vee \neg A = 1$
Закон двойного отрицания	$\neg(\neg A) = A$
Законы де Моргана	$\neg(A \vee B) = \neg A \wedge \neg B$, $\neg(A \wedge B) = \neg A \vee \neg B$
Законы рефлексивности (идемпотенции)	$A \vee A = A$, $A \wedge A = A$
Свойства логических констант 1 и 0	$A \wedge 0 = 0$, $A \wedge 1 = A$, $A \vee 0 = A$, $A \vee 1 = 1$
Законы поглощения	$A \vee (A \wedge B) = A$, $A \wedge (A \vee B) = A$, $A \vee (\neg A \wedge B) = A \vee B$

Связь алгебры логики с программированием

В большинстве существующих языков программирования предусмотрены функции для реализации основных логических операций («НЕ» — `NOT`, «И» — `AND`, `&`, «ИЛИ» — `OR`, `|`, «Исключающее ИЛИ» — `XOR`, `^`), позволяющих обрабатывать данные логического типа (`Boolean`). При этом обычно считается, что значение «ложь» (`FALSE`) связано с целочисленным значением 0, а значение «истина» (`TRUE`) — с целочисленным значением 1 (или любым ненулевым).

В программировании подобные логические операции широко используются вместе с операторами сравнения для формирования логических условий (в командах ветвления, циклах с пост- или предусловием и пр.), а также собственно для обработки логических данных. При этом синтаксис многих языков программирования

допускает сложные конструкции из оператора присваивания и операций сравнения, например в языке БЕЙСИК является корректной запись:

```
X = (A = B)
```

Она означает, что выполняется сравнение значений переменных A и B и при их равенстве (для текстовых данных — совпадении) переменной X присваивается логическое значение «Истина» (иначе — «Ложь»).

Кроме операций над логическими данными, в некоторых языках программирования (например, Java или C) также предусмотрены *побитовые логические операции*, при которых целочисленные операнды рассматриваются как двоичные числа, выбранная логическая операция производится для каждой соответствующей по номеру позиции в числе пары битов, а результат представляет собой также целое число.

Применение алгебры логики при решении задач

Законы алгебры логики могут применяться при решении различных задач, связанных с принадлежностью точки заданному интервалу либо области, при определении выполнения или невыполнения заданного правила и т.д.

Примеры.

1. При решении задач типа «Какое из приведённых имён удовлетворяет логическому условию: $\neg$*(последняя буква гласная* $\rightarrow$*первая буква согласная)* $\wedge$ *вторая буква согласная*» достаточно для каждого из трёх упомянутых в условии правил предусмотреть логическую переменную, значение которой равно 1 «Истина», если данное правило выполнено для конкретного имени, либо 0 («Ложь»), если данное правило не выполнено для конкретного имени, и далее вести обработку согласно правилам алгебры логики значений этих переменных.

2. При решении задач типа «*Для какого из указанных значений X истинно высказывание:* $\neg((X > 2) \rightarrow (X > 3))$» аналогичным образом выделяются логические переменные, соответствующие каждой из приведённых в условии операций сравнения для тех или

иных граничных и внутриинтервальных значений числовой переменной X, после чего полученные значения этих логических переменных обрабатываются согласно правилам алгебры логики.

3. При решении задач об определении принадлежности точек интервалу на числовой прямой либо области плоскости, заданной системой неравенств (задачи группы С1) логические операции используются при формировании требуемого условия, определяющего нужную область.

Правила, которыми нужно руководствоваться при решении логических задач с интервалами:

1) при обмене местами левой и правой частей неравенства его знак меняется на противоположный;

2) если неравенство является ложным, то эквивалентное ему истинное неравенство не только имеет противоположный знак, но и становится из строгого нестрогим и наоборот; то же самое происходит при замене истинного неравенства эквивалентным ему ложным;

3) соединение компонентов логического выражения операцией «И» соответствует *пересечению* интервалов их истинности (интервалов значений, при которых эти компоненты истинны); соединение компонентов логического выражения операцией «ИЛИ» соответствует *объединению* интервалов их истинности;

4) интервал ложности представляет собой всю часть числовой прямой, кроме интервала истинности этого выражения, — производится *вычитание* интервала истинности из числовой прямой; аналогично определяется интервал истинности по интервалу ложности.

Кроме того, при решении задач с интервалами надо внимательно читать текст условия: если в вопросе фигурирует, например, «наибольшее *натуральное* число X» или «наибольшее *целое положительное* число X», то это означает добавление дополнительного условия: $X > 0$.

Разбор типовых задач

Задача 1*. Дан фрагмент таблицы истинности выражения F:

X	Y	Z	F
0	0	0	0
0	0	1	0
1	1	1	1

Каким выражением может быть F?

1) $X \wedge Y \wedge Z$

2) $\neg X \vee \neg Y \vee Z$

3) $X \vee Y \vee Z$

4) $\neg X \wedge \neg Y \wedge \neg Z$

Решение

Обычно в таких задачах даётся только фрагмент таблицы истинности, поэтому пытаться решать задачу «в лоб», выводя соответствующее таблице логическое выражение и сравнивая его с вариантами ответов, бессмысленно. Лучше всего просто проверять предлагаемые ответы один за другим, поочерёдно подставляя в соответствующее логическое выражение значения переменных из каждой строки таблицы и проверяя, получается ли в результате указанное в той же строке таблицы требуемое значение F.

 Если для какой-то из строк таблицы получается неправильный результат, то можно прервать проверку данного варианта ответа и сразу перейти к следующему варианту.

Проверяется первый вариант ответа: $X \wedge Y \wedge Z$, учитывая, что операция И даёт значение 1 только когда все значения переменных равны 1. (Серым цветом отмечаются строки таблицы, в которых результат вычисления выражения не совпадает с заданным значением F.)

X	Y	Z	X ∧ Y ∧ Z	F
0	0	0	0	0
0	0	1	0	0
1	1	1	1	1

Таким образом, правильный ответ может быть найден практически сразу, но при наличии времени рекомендуется проверить и остальные варианты.

Проверяется второй вариант ответа: $\neg X \vee \neg Y \vee Z$, учитывая, что операция ИЛИ даёт значение 1, если значение хотя бы одной переменной равно 1 (для чего значения переменных, записанных с отрицанием, должны быть нулевыми).

X	Y	Z	$\neg X \vee \neg Y \vee Z$	F
0	0	0	1	0
0	0	1	1	0
1	1	1	1	1

Проверяется третий вариант ответа: $X \vee Y \vee Z$.

X	Y	Z	$X \vee Y \vee Z$	F
0	0	0	0	0
0	0	1	1	0
1	1	1	1	1

Проверяется четвёртый вариант ответа: $\neg X \wedge \neg Y \wedge \neg Z$.

X	Y	Z	$\neg X \wedge \neg Y \wedge \neg Z$	F
0	0	0	1	0
0	0	1	0	0
1	1	1	0	1

Таким образом, правильным является первый вариант ответа.

Ответ: вариант ответа №1.

Задача 2*. Символом F обозначено одно из указанных ниже логических выражений от трёх аргументов: X, Y, Z.

Дан фрагмент таблицы истинности выражения F:

X	Y	Z	F
0	1	1	0
1	1	1	1
0	0	1	1

Какое выражение соответствует F?

1) $X \wedge \neg Y \wedge \neg Z$
2) $\neg X \wedge \neg Y \wedge Z$
3) $\neg X \vee \neg Y \vee Z$
4) $X \vee \neg Y \vee \neg Z$

Решение

Проверяется первый вариант ответа: $X \wedge \neg Y \wedge \neg Z$, учитывая, что операция И даёт значение 1, только когда все значения переменных равны 1.

X	Y	Z	X ∧ ¬Y ∧ ¬Z	F
0	1	1	0	0
1	1	1	0	1
0	0	1		1

Проверяется второй вариант ответа: $\neg X \wedge \neg Y \wedge Z$.

X	Y	Z	¬X ∧ ¬Y ∧ Z	F
0	1	1	0	0
1	1	1	0	1
0	0	1		1

Проверяется третий вариант ответа: $\neg X \vee \neg Y \vee Z$, учитывая, что операция ИЛИ даёт значение 1, если значение хотя бы одной переменной равно 1.

X	Y	Z	¬X ∨ ¬Y ∨ Z	F
0	1	1	1	0
1	1	1		1
0	0	1		1

Проверяется четвёртый вариант ответа: $X \vee \neg Y \vee \neg Z$.

X	Y	Z	X ∨ ¬Y ∨ ¬Z	F
0	1	1	0	0
1	1	1	1	1
0	0	1	1	1

Таким образом, правильным является четвёртый вариант ответа.

Ответ: вариант ответа №4.

Задача 3*. Какое из приведённых имён удовлетворяет логическому условию: (первая буква согласная →вто-

рая буква согласная) ∧ (предпоследняя буква гласная → последняя буква гласная)?

1) КРИСТИНА
2) МАКСИМ
3) СТЕПАН
4) МАРИЯ

Решение

Решение таких задач не составляет особых трудностей. Единственная «хитрость» состоит в том, что нужно сначала преобразовать запись логического выражения в более привычную таблицу двоичных значений 0 и 1. Строки этой таблицы соответствуют вариантам ответов.

Составляется таблица. (*Для справки:* операция И даёт значение 1, только когда все значения переменных равны 1, а операция следования (→) даёт результат 0 только в одном случае — когда из 1 следует 0.)

Имя	X_1: первая буква согласная	X_2: вторая буква согласная	X_3: предпоследняя буква гласная	X_4: последняя буква гласная	X_5: $X_1 \to X_2$	X_6: $X_3 \to X_4$	Результат: $X_5 \to X_6$
КРИСТИНА	1	1	0	1	1	1	1
МАКСИМ	1	0	1	0	0	0	0
СТЕПАН	1	1	1	0	1	0	0
МАРИЯ	1	0	1	1	0	1	0

В таблице серым цветом выделена строка, соответствующая правильному ответу (первому).

Обнаружив, что условие выполняется для какого-то варианта, остальные варианты ответа можно не проверять.

Ответ: КРИСТИНА (вариант ответа №1).

Задача 4*. Какое из приведённых имён удовлетворяет логическому условию:

¬(последняя буква гласная → первая буква согласная) ∧ ∧ вторая буква согласная

1) ИРИНА
2) АРТЁМ
3) СТЕПАН
4) МАРИЯ

Решение

Составляется таблица. (*Для справки:* операция И даёт значение 1, только когда все значения переменных равны 1, а операция следования ($\rightarrow$) даёт результат 0 только в одном случае — когда из 1 следует 0.)

Имя	X_1: последняя буква гласная	X_2: первая буква согласная	X_3: вторая буква согласная	X_4: $X_1 \rightarrow X_2$	X_5: $\neg X_4$	Результат: $X_5 \rightarrow X_3$
ИРИНА	1	0	1	0	1	1
АРТЁМ	0	0	1	1	0	0
СТЕПАН	0	1	1	1	0	0
МАРИЯ	1	1	0	1	0	0

В таблице выделена строка, соответствующая правильному ответу (первому).

Ответ: ИРИНА (вариант ответа №1).

Задача 5*. Укажите, какое логическое выражение равносильно выражению $A \vee \neg(\neg B \vee \neg C)$:

1) $\neg A \vee B \vee \neg C$
2) $A \vee (B \wedge C)$
3) $A \vee B \vee C$
4) $A \vee \neg B \vee \neg C$

Решение

Преобразуется исходное выражение, используя законы алгебры логики:

$A \vee \neg(\neg B \vee \neg C) =$ {закон де Моргана} $= A \vee (B \wedge C)$

(наличие скобок несущественно, так как операция «И» имеет более высокий приоритет, чем операция «ИЛИ»).

Полученное выражение совпадает с вариантом ответа №2.

Ответ: $A \vee (B \wedge C)$ (вариант ответа №2).

Задача 6*. Какое логическое выражение равносильно выражению $\neg(\neg A \vee \neg B) \wedge C$:

1) $\neg A \vee B \vee \neg C$
2) $A \wedge B \wedge C$
3) $(A \vee B) \wedge C$
4) $(\neg A \wedge \neg B) \vee \neg C$

Решение

Преобразуется исходное выражение, используя законы алгебры логики:

$\neg(\neg A \vee \neg B) \wedge C =$ {закон де Моргана} $= A \wedge B \wedge C.$

Полученное выражение совпадает с вариантом ответа №2.

Ответ: $A \wedge B \wedge C$ (вариант ответа №2).

Задача 7*. Каково наибольшее целое число X, при котором истинно высказывание

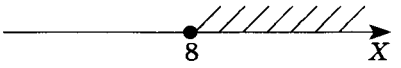
$$(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))?$$

Решение

В данной задаче требуется полностью решить задачу, а не проверять четыре предложенных варианта ответа, и добавлено ещё одно дополнительное условие: искомое число должно быть *целым* и *наибольшим*. Такое условие означает, что в результате решения логического уравнения необходимо получить диапазон значений X, который распространяется от требуемого наибольшего значения в сторону уменьшения. Решать же задачу лучше не табличным способом, а при помощи графической схемы, которая представляет собой пересекающиеся интервалы на числовой прямой.

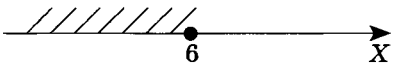
Способ 1

1. Строится интервал для условия $50 < X \cdot X$. Данное условие преобразовывается в эквивалентное условие $X \cdot X \geqslant 50$; кроме того, речь идёт только о целых числах: $X \geqslant 8$.



2. Строится интервал для условия $50 > (X + 1) \cdot (X + 1)$, которое преобразовывается в эквивалентное:

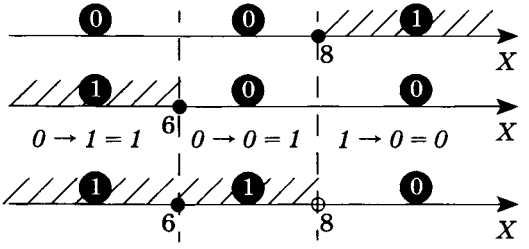
$(X + 1) \cdot (X + 1) \leqslant 50$, откуда $X \leqslant 6$.



3. Теперь следует интерпретировать логическую операцию следования ($\rightarrow$) для построенных интервалов.

Интервалы располагаются один под другим. Заштрихованные значения соответствуют логической единице, а не заштрихованные — нулю.

Операция следования в данном случае «направлена сверху вниз». Она даёт значение 0, только когда из 1 следует 0, а в остальных случаях получается нужное значение 1 (истина). Поэтому, разбив числовую ось на соответствующие интервалы, нетрудно построить требуемый «результирующий» интервал.



При этом важно учитывать, что в исходных интервалах граничные точки принадлежат соответствующим интервалам и потому на эти точки полностью распространяется заданная логическая операция. Следовательно, в результирующем интервале граничная точка 8 (для которой выполняется равенство $1 \to 0 = 0$) уже не будет входить в результирующий «единичный» интервал. Наибольшее целое значение X, удовлетворяющее условию задачи, равно 7.

Способ 2

1. Наоборот, сначала рассматривается, в каких ситуациях истинна операция следования. Результат её выполнения является ложным в единственном случае — когда из 1 следует 0, и истинным в остальных случаях. Тогда исходное выражение можно представить как три возможных варианта систем неравенств:

$$(50 < X \cdot X) \to (50 > (X + 1) \cdot (X + 1))$$

$$\begin{cases} 50 < X \cdot X; & (1) \\ 50 > (X + 1) \cdot (X + 1); & (1) \end{cases} \qquad \begin{cases} 50 < X \cdot X; & (0) \\ 50 > (X + 1) \cdot (X + 1); & (0) \end{cases}$$

$$\begin{cases} 50 < X \cdot X; & (0) \\ 50 > (X + 1) \cdot (X + 1); & (1) \end{cases}$$

Для поиска решений этих неравенств нужно в тех случаях, когда неравенство ложно (логическое значение 0), изменить знак неравенства на противоположный, причём строгое неравенство преобразуется в нестрогое и наоборот:

$$\begin{cases} 50 < X \cdot X, \\ 50 > (X+1) \cdot (X+1); \end{cases} \qquad \begin{cases} 50 \geqslant X \cdot X, \\ 50 > (X+1) \cdot (X+1); \end{cases}$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$\begin{cases} X \cdot X > 50, \\ (X+1) \cdot (X+1) < 50; \end{cases} \qquad \begin{cases} X \cdot X \leqslant 50, \\ (X+1) \cdot (X+1) < 50; \end{cases}$$

$$\begin{cases} 50 \geqslant X \cdot X, \\ 50 \leqslant (X+1) \cdot (X+1); \end{cases}$$

$$\Downarrow$$

$$\begin{cases} X \cdot X \leqslant 50, \\ (X+1) \cdot (X+1) \geqslant 50; \end{cases}$$

Значение $(X + 1) \cdot (X + 1)$ заведомо больше, чем значение $X \cdot X$. Поэтому первая система неравенств не имеет решения.

Решение (в целых числах) второй системы неравенств — интервал значений $(-\infty, 6]$. Он определяется по второму неравенству этой системы:
$(X + 1) \cdot (X + 1) < 50 \Rightarrow X + 1 < \sqrt{50} \Rightarrow X + 1 \leqslant 7 \Rightarrow X \leqslant 6$.

Решение (в целых числах) третьей системы неравенств — пересечение интервалов значений $(-\infty, 7]$ и $[7, +\infty)$, которое равно числу 7.

Из найденных двух чисел — решений систем уравнений — требуется наибольшее. Оно равно 7.

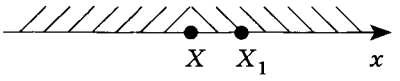
Способ 3

Операция следования даёт результат «Истина» в трёх случаях из четырёх возможных, а результат «Ложь» — только в одном случае из четырёх, когда из 1 («Истина») следует 0 («Ложь»). Поэтому удобно решать предложенную задачу в «обратной» формулировке:

Каково *наименьшее* целое число, при котором *ложно* высказывание $(50 < X \cdot X) \rightarrow (50 > (X + 1) \cdot (X + 1))$?

Прежде всего доказывается правомерность такой замены формулировки задачи.

Предполагается, что искомое значение X в первоначальной задаче найдено. Раз в исходной задаче оно названо наибольшим целым числом, то это означает, что заданное выражение истинно на диапазоне значений $(-\infty, X]$. Тогда на оставшейся части числовой оси, т.е. в диапазоне $(X, +\infty)$, данное выражение будет ложно. Определив наименьшее целое значение X_1, при котором заданное выражение ложно, легко получается искомое наибольшее X, при котором это выражение истинно: $X = X_1 - 1$.



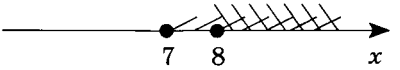
Новая задача решается:

1) Выражение $(50 < X \cdot X)$ $(50 > (X + 1) \cdot (X + 1))$ ложно, когда

$$\begin{cases} 50 < X \cdot X \text{ — } \textit{истинно;} \\ 50 > (X + 1) \cdot (X + 1) \text{ — } \textit{ложно.} \end{cases}$$

2) Первой операции сравнения соответствует диапазон целых чисел $[8, +\infty)$. Второй операции сравнения (учитывая, что она должна быть ложной, т.е. истинно сравнение $50 \leqslant (X + 1) \cdot (X + 1)$) соответствует диапазон целых чисел $[7, +\infty)$.

3) Графическое представление решения «модифицированной» задачи:



4) То, что оба условия сравнения объединены в систему, означает, что оба они должны выполняться одновременно. Следовательно, решением этой системы уравнений является *пересечение* построенных интервалов (пересечение множества составляющих их целых чисел). Этот *интервал ложности* операции следования — $[8, +\infty)$.

5) Наименьшее целое значение (наше X_1), при котором заданное выражение ложно, равно 8.

Искомое значение X в первоначальной задаче (наибольшее целое число, при котором исходное логическое выражение истинно) на 1 меньше найденного нами значения X_1.

Следовательно для исходной задачи ответ — число 7.

Ответ: 7

B15 Решение логических уравнений. Решение систем логических уравнений

Конспект

Теоретический материал по данной теме подробно представлен в разделе 4 «Основы логики».

Полезно запомнить следующее **правило**: если известно количество решений уравнения $F(x_1, x_2, \ldots x_n) = 1$, то количество возможных решений «противоположного» уравнения $F(x_1, x_2, \ldots x_n) = 0$ равно разности количества всех возможных комбинаций значений переменных $x_1, x_2, \ldots x_n$ (которое равно 2^n) и количества решений уравнения $F(x_1, x_2, \ldots x_n) = 1$ (и, соответственно, наоборот):

Кол-во решений $F(x_1, x_2, \ldots x_n) = 1$	Кол-во всех возможных комбинаций $x_1, x_2, \ldots x_n$	Кол-во решений $F(x_1, x_2, \ldots x_n) = 0$
X_1		X_0
=	2^n	=
$2^n - X_0$		$2^n - X_1$

Это правило легко доказать, рассмотрев полную таблицу истинности логической функции $F(x_1, x_2, \ldots x_n)$: если исключить из нее строки, соответствующие значению $F = 1$, то останутся строки, соответствующие значению $F = 0$, и наоборот.

Разбор типовых задач

Задача 1*. Сколько различных решений имеет уравнение

$$((J \to K) \to (M \wedge N \wedge L)) \wedge ((J \wedge \neg K) \to \\ \to \neg(M \wedge N \wedge L)) \wedge (M \to J) = 1,$$

где J, K, L, M, N — логические переменные?

В ответе **не нужно** перечислять все различные наборы значений J, K, L, M и N, при которых выполнено данное равенство. В качестве ответа нужно указать количество таких наборов.

Решение.

Самая последняя по порядку выполнения логическая операция здесь — операция И. Её результат равен 1 только в одном-единственном случае — когда все операнды равны 1. Следовательно, взамен исходного уравнения получается система уравнений:

$$\begin{cases} (J \to K) \to (M \wedge N \wedge L) = 1; \\ (J \wedge \neg K) \to \neg(M \wedge N \wedge L) = 1; \\ M \to J = 1. \end{cases}$$

Сначала рассматривается третье уравнение. Из него однозначно определяется, что значение переменной J обязательно должно быть равно 1, а значение M может быть любым (2 возможных варианта значений переменных).

Операция следования даёт 0 в случае, когда из 1 следует 0, и 1 — во всех остальных случаях. Анализируются вторые операнды в двух первых уравнениях (во втором уравнении тот же самый компонент, что и в первом уравнении, взят с отрицанием).

Рассматривается каждый из возможных вариантов.

1. Пусть верно первое уравнение: $(J \to K) \to \\ \to (M \wedge N \wedge L) = 1$. Тогда обязательное условие его выполнения — $M \wedge N \wedge L = 1$. Это возможно только в одном случае — когда все три эти переменные равны 1. А вот значение $J \to K$ может быть любым: 0 или 1.

Однако если $M \wedge N \wedge L = 1$, то второе уравнение будет истинным только в одном-единственном случае: когда $J \wedge \neg K = 0$: ведь $\neg(M \wedge N \wedge L)$ тогда равно 0, а операция следования при нулевом втором операнде даёт единицу, только если нулю равен и первый операнд. Причём J, как выяснилось ранее, может быть равно только 1. Значит, чтобы $J \wedge \neg K$ было равно 0, нужно, чтобы $\neg K$ было равно 0, т.е. чтобы K было равно 1. Итого по результатам анализа этой ситуации получается, что в данном случае возможен только один вариант значений переменных: $J = K = M = N = L = 1$.

2. Пусть теперь верно второе уравнение: $(J \wedge \neg K) \rightarrow \rightarrow \neg(M \wedge N \wedge L) = 1$. Тогда для его выполнения нужно, чтобы $\neg(M \wedge N \wedge L)$ было равно 1, т.е. $M \wedge N \wedge L = 0$. Это возможно, если хотя бы одна переменная имеет значение 0. Переменных три, следовательно, всех возможных комбинаций их значений будет $2^3 = 8$. Из них одна комбинация (когда все три переменных равны 1) не подходит, следовательно, пригодных вариантов остается 7. Однако если $M \wedge N \wedge L = 0$, то первое уравнение системы выполняется только в одном случае: если $J \rightarrow K = 0$. А это возможно только в одном случае: когда $J = 1$, а $K = 0$ (что не противоречит выводам для третьего уравнения системы). Значит, в этом случае возможно 7 вариантов значений переменных, при которых исходное уравнение верно.

Итого, для обоих рассмотренных случаев возможных вариантов значений переменных будет $7 + 1 = 8$.

Ответ: 8

Подобные логические рассуждения могут быть по силам не всем учащимся, да и хорошо знакомые с логикой школьники могут запутаться и допустить ошибку. Поэтому для таких задач на ЕГЭ лучше придерживаться следующего правила: если остаётся достаточно времени, то лучше решать такую задачу традиционным способом — составляя полную таблицу истинности и подсчитывая в ней количество «правильных» строк. Если время в дефиците, то можно попытаться сократить «рутинную» работу, поискав требуемую цепь логических рассуждений.

Задача 2*. Сколько различных решений имеет уравнение

$$J \wedge \neg K \wedge L \wedge \neg M \wedge (N \vee \neg N) = 0,$$

где J, K, L, M, N — логические переменные?

В ответе **не нужно** перечислять все различные наборы значений J, K, L, M и N, при которых выполнено данное равенство. В качестве ответа вам нужно указать количество таких наборов.

Решение.

Во-первых, комбинация $(N \vee \neg N)$ ***всегда*** равна 1, а это означает, что переменная N принципиально может быть любой и никак не определяет выполнение уравнения. Поэтому переменная N временно исключается из рассмотрения.

Во-вторых, операция И даёт в результате 0, если хотя бы один операнд равен 0.

В-третьих, «активных» переменных-операндов (значения которых могут влиять на результат выполнения логического выражения) — четыре.

Сколько не повторяющихся комбинаций можно составить из n элементов, каждый из которых может иметь значение 0 или 1? Очевидно[1], их будет 2^n. Соответственно, четыре переменные могут дать $2^4 = 16$ возможных комбинаций. Для решения годятся все, кроме одной — когда все эти операнды равны 1. Значит, пригодных комбинаций будет 15.

Переменная N может принимать любое из двух значений (0 или 1), не влияя на результат вычисления логического выражения. Следовательно, с учётом N возможно $2 \cdot 15 = 30$ возможных комбинаций пяти «участ-

[1] Ответить на этот вопрос очень просто, если вспомнить, что такими неповторяющимися комбинациями являются значения битов двоичного числа из соответствующего количества разрядов (в данном случае 4) при полном переборе числовых значений от 0 до максимально возможного значения (от 0000 до 1111). Количество всех возможных неповторяющихся комбинаций значений четырёх двоичных переменных равно $1111_2 + 1 = 16$ (где добавляемая ещё одна комбинация — та самая из одних нулей).

вующих» в уравнении логических переменных, при которых приведённое в условии логическое уравнение верно.

Ответ: 30.

При подготовке к экзамену для получения правильного ответа и проверки своего решения можно составить компьютерную программу, содержащую некоторое количество (по числу используемых в логическом уравнении переменных) вложенных циклов, в которых соответствующая цикловая переменная меняется от 0 до 1. Например[1]:

```
program z2005_b2;
  var k,l,m,n : boolean;
      x : byte;
  begin
    x := 0;
    for k := false to true do
      for l := false to true do
        for m := false to true do
          for n := false to true do
              if (k and l and m) or (not(l) and
                 not(m) and n) then x := x + 1;
    writeln ('кол-во решений:',x);
     end.
```

Задача 3*. Сколько различных решений имеет система уравнений

$$\begin{cases} ((x_1 \equiv x_2) \vee (x_3 \equiv x_4)) \wedge (\neg(x_1 \equiv x_2) \vee \neg(x_3 \equiv x_4)) = 1, \\ ((x_3 \equiv x_4) \vee (x_5 \equiv x_6)) \wedge (\neg(x_3 \equiv x_4) \vee \neg(x_5 \equiv x_6)) = 1, \\ \ldots \\ ((x_7 \equiv x_8) \vee (x_9 \equiv x_{10})) \wedge (\neg(x_7 \equiv x_8) \vee \neg(x_9 \equiv x_{10})) = 1, \end{cases}$$

где x_1, x_2, ..., x_{10} — логические переменные?

[1] На Паскале вместо значений 0 и 1 нужно использовать логические (тип `boolean`) значения `true` и `false`, однако в операторе цикла `for` эти значения можно использовать так же, как константы 1 и 0 соответственно, так как логический тип тоже является перечислимым. Кроме того, поскольку нужно проверять равенство заданного логического выражения значению 1 (т.е. `true`), в условном операторе нет необходимости записывать операцию сравнения: ведь ветвь `then` выполняется только в этом случае.

В ответе не нужно перечислять все различные наборы значений x_1, x_2, ..., x_{10}, при которых выполнена данная система равенств. В качестве ответа вам нужно указать количество таких наборов.

Решение

1) Анализируется первое уравнение:

$((x_1 \equiv x_2) \vee (x_3 \equiv x_4)) \wedge (\neg(x_1 \equiv x_2) \vee \neg(x_3 \equiv x_4)) = 1.$

Последняя выполняемая операция здесь — **И**, поэтому:

$((x_1 \equiv x_2) \vee (x_3 \equiv x_4)) = 1$ и $(\neg(x_1 \equiv x_2) \vee \neg(x_3 \equiv x_4)) = 1.$

Следует обратить внимание: в обеих частях записаны одни и те же тождества, только в первом случае они записаны «как есть», а во втором — с отрицаниями. Тогда, если $(x_1 \equiv x_2) = 1$ и $(x_3 \equiv x_4) = 1$, то первая запись будет истинной, но тогда $\neg(x_1 \equiv x_2)$ и $\neg(x_3 \equiv x_4)$ оба будут ложными, и вторая запись ложна. И наоборот, при $(x_1 \equiv x_2) = 0$ и $(x_3 \equiv x_4) = 0$ первая запись будет ложной, а вторая (с отрицаниями) — истинной. Не подходит ни тот, ни другой вариант. «Спасает положение» то, что тождества в обеих записях соединены операцией **ИЛИ**, т.е. оба раза достаточно, чтобы единице было равно хотя бы одно из этих тождеств.

Вывод: чтобы первое уравнение системы было равно 1, нужно, чтобы либо $(x_1 \equiv x_2) = 1$ и $(x_3 \equiv x_4) = 0$, либо, наоборот, $(x_1 \equiv x_2) = 0$ и $(x_3 \equiv x_4) = 1$.

Первое из этих «либо» даёт такие варианты значений переменных, когда x_1 и x_2 одинаковы, а x_3 и x_4 различны:

(0, 0, 1, 0)
(0, 0, 0, 1)
(1, 1, 1, 0)
(1, 1, 0, 1)

Второе «либо», аналогично, даёт варианты, в которых, наоборот, x_1 и x_2 различны, а x_3 и x_4 одинаковы:

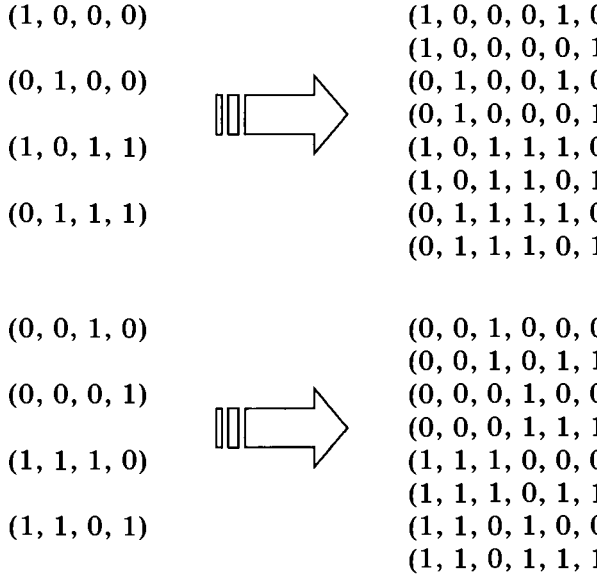
(1, 0, 0, 0)
(0, 1, 0, 0)
(1, 0, 1, 1)
(0, 1, 1, 1)

Всего — 8 вариантов.

2) Добавляется в анализ второе уравнение:

$((x_3 \equiv x_4) \vee (x_5 \equiv x_6)) \wedge (\neg(x_3 \equiv x_4) \vee \neg(x_5 \equiv x_6)) = 1.$

Рассуждая аналогично и учитывая, что для x_3 и x_4 возможные варианты «унаследованы» от предыдущего уравнения, получается, что в вариантах значений x_5, x_6, добавленных этим вторым уравнением, для одинаковых значений x_3 и x_4 должны быть разными значения x_5 и x_6, а для различных значений x_3 и x_4 — одинаковые значения x_5 и x_6:



Итого из 8 предыдущих вариантов благодаря второму уравнению получается 16 (вдвое больше).

3) Очевидно, такая тенденция сохранится и дальше, ведь уравнения системы — типовые. Значит, добавление в рассмотрение третьего уравнения, пропущенного в записи системы и использующего переменные x_5, x_6, x_7, x_8, снова удвоит количество вариантов значений переменных: из 16 их получится 32.

Аналогично, последнее, четвёртое уравнение системы (переменные x_7, x_8, x_9, x_{10}) снова удваивает коли-

чество вариантов, «унаследованное» от предыдущего уравнения. В итоге для всей системы уравнений получается 64 возможных варианта значений переменных $x_1 - x_{10}$.

Ответ: 64 варианта значений переменных.

Задача 4. Сколько существует различных наборов значений логических переменных x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5, которые удовлетворяют всем перечисленным ниже условиям?

$(x_1 \to x_2) \wedge (x_2 \to x_3) \wedge (x_3 \to x_4) \wedge (x_4 \to x_5) = 1;$

$(y_1 \to y_2) \wedge (y_2 \to y_3) \wedge (y_3 \to y_4) \wedge (y_4 \to y_5) = 1;$

$y_5 \to x_5 = 1$

В ответе не нужно перечислять все различные наборы значений переменных x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5, при которых выполнена данная система равенств. В качестве ответа вам нужно указать количество таких наборов.

Решение

Как и всегда при решении задач с системами логических уравнений, нужно сначала проанализировать каждое уравнение в отдельности. При этом, первое и второе уравнения заданной системы практически идентичны (с точностью до имён переменных — «игреки» вместо «иксов»), и это существенно облегчает работу.

Анализируя первое уравнение:

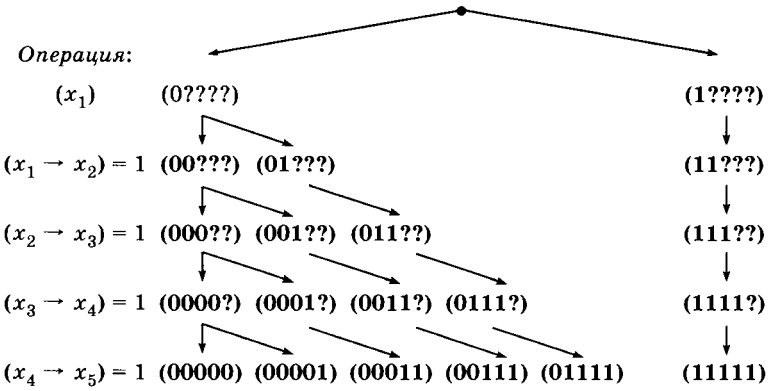
$(x_1 \to x_2) \wedge (x_2 \to x_3) \wedge (x_3 \to x_4) \wedge (x_4 \to x_5) = 1.$

Таблица истинности логической операции следования: единственная ситуация, при которой её результат равен нулю, — когда из единицы следует нуль, а во всех других случаях эта операция возвращает единицу:

a	b	$a \to b$
0	0	1
0	1	1
1	0	0
1	1	1

Кроме того, поскольку все отдельные операции следования в первом уравнении соединены операцией И, для выполнения заданного в нём равенства требуется, чтобы *все* операции следования давали в результате единицу.

Чтобы найти все возможные комбинации значений переменных, задействованных в первом уравнении, удобнее всего выполнить построение дерева решений: это позволит не запутаться и не пропустить какие-то варианты. При построении дерева на каждом его очередном шаге анализируется очередная пара переменных и для каждой имеющейся ветви определяются дальнейшие варианты ветвления. Слева указываются логические операции следования, которые и анализируются на соответствующих шагах (уровнях дерева). Ключевым моментом при построении дерева является уже отмеченный выше факт, что для получения единичного результата из нуля может следовать любое значение второй переменной, а из единицы — *только* единица.



Основную идею при построении данного дерева можно условно выразить фразой: «*размножаются только нули*». То есть если имеется начало набора значений пяти переменных, которое на данный момент завершается нулём, то продолжить его можно как нулём, так и единицей (в дереве имеется ветвление), но если текущая

последовательность заканчивается единицей, то продолжать её можно *только* единицей, и в дереве не будет никакого ветвления, а только продолжение уже существующей ветви.

Полный набор возможных значений переменных, удовлетворяющих первому уравнению, тогда содержится в самой нижней строке построенного дерева (в его «листьях»): $(x_1x_2x_3x_4x_5)$ = (00000), (00001), (00011), (00111), (01111), (11111).

Второе уравнение по структуре полностью совпадает с первым. Поэтому анализировать его нет необходимости, и можно сразу записать набор возможных для него значений переменных: $(y_1y_2y_3y_4y_5)$ = (00000), (00001), (00011), (00111), (01111), (11111).

Если бы в условии задачи присутствовали только рассмотренные два уравнения, то, поскольку в них нет общих переменных, решением этой системы уравнений были бы *все* возможные попарные сочетания найденных наборов значений «иксов» и «игреков». Именно третье уравнение, в котором одной логической операцией связаны один из «иксов» и один из «игреков», является *«ключом»*, определяющим выбор: какие из найденных комбинаций наборов значений $(x_1x_2x_3x_4x_5)$ и $(y_1y_2y_3y_4y_5)$ подходят, а какие — нет.

Запись этого третьего уравнения:

$y_5 \rightarrow x_5 = 1$.

Согласно ему, из всех найденных пар наборов значений «иксов» и «игреков» для решения подходят только такие, в которых значения указанных переменных соответствуют истинности заданной логической операции, т.е.:

- когда в наборе значений $(y_1y_2y_3y_4y_5)$ пятая цифра равна нулю, в пару с ним годятся *любые* наборы значений $(x_1x_2x_3x_4x_5)$, поскольку, что бы в них ни стояло в пятой позиции (0 или 1), результат операции $y_5 \rightarrow x_5 = 1$ в любом случае будет равен 1 (см. таблицу истинности для этой операции);

- когда в наборе значений $(y_1y_2y_3y_4y_5)$ пятая цифра равна единице, в пару с ним годятся *только такие* наборы значений $(x_1x_2x_3x_4x_5)$, в которых пятая цифра равна 1.

Удобнее и нагляднее всего расписать все получаемые комбинации значений x и y в виде таблицы (*«матрицы решений»*). Анализируемые цифры в ней выделены подчеркиванием.

$(y_1y_2y_3y_4y_5)$	$(x_1x_2x_3x_4x_5)$						*Кол-во вариантов (пар)*
	(00000)	**(00001)**	**(00011)**	**(00111)**	**(01111)**	**(11111)**	
(00000)	+	+	+	+	+	+	6
(00001)	–	+	+	+	+	+	5
(00011)	–	+	+	+	+	+	5
(00111)	–	+	+	+	+	+	5
(01111)	–	+	+	+	+	+	5
(11111)	–	+	+	+	+	+	5
Всего возможных вариантов (пар) наборов значений $(x_1x_2x_3x_4x_5)$ ***и*** $(y_1y_2y_3y_4y_5)$**:**							**31**

Ответ: заданная система уравнений имеет 31 решение.

Таким образом, в данной задаче система логических уравнений состоит из двух чётко обособленных частей: первые два уравнения представляют собой «генераторы наборов значений переменных», а последнее уравнение — это «селектор», «фильтр», осуществляющий отбор из всех возможных попарных комбинаций наборов «иксов» и «игреков».

Задача 5*. Сколько существует различных наборов значений логических переменных x_1, x_2, ..., x_9, x_{10}, которые удовлетворяют всем перечисленным ниже условиям?

$$(x_1 \wedge \neg x_2) \vee (x_3 \wedge \neg x_4) = 0,$$
$$(x_3 \wedge \neg x_4) \vee (x_5 \wedge \neg x_6) = 0,$$
$$(x_5 \wedge \neg x_6) \vee (x_7 \wedge \neg x_8) = 0,$$
$$(x_7 \wedge \neg x_8) \vee (x_9 \wedge \neg x_{10}) = 0.$$

В ответе не нужно перечислять все различные наборы значений x_1, x_2, ..., x_9, x_{10}, при которых выполнена данная система равенств. В качестве ответа вам нужно указать количество таких наборов.

Решение

1. Анализ начинается с последнего уравнения:
$(x_7 \wedge \neg x_8) \vee (x_9 \wedge \neg x_{10}) = 0$.

Это уравнение состоит из двух операндов (скобок), соединённых логической операцией ИЛИ. Результат выполнения операции ИЛИ равен нулю в единственном случае — если оба операнда равны нулю. Тогда:

$$\begin{cases}(x_7 \wedge \neg x_8) = 0,\\ (x_9 \wedge \neg x_{10}) = 0.\end{cases}$$

Анализируются все возможные варианты значений переменных x_7, x_8, x_9, x_{10}, удовлетворяющие этим условиям, и составляется таблица. При этом учитывается, что логическая операция И даёт нулевой результат в трёх случаях — когда хотя бы одна переменная равна нулю или обе переменные равны нулю, и для каждого такого случая для первой пары переменных возможно по три случая для второй пары переменных. Кроме того, нужно учитывать, что переменные x_8 и x_{10} записаны с отрицаниями, поэтому в таблицу нужно записывать противоположные значения.

Таблица 1.1

$(x_7 \wedge \neg x_8)$	$(x_9 \wedge \neg x_{10})$
0	0

Таблица 1.2

x_7	x_8	x_9	x_{10}
0	1 ($\neg$0)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)
0	0 ($\neg$1)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)
1	1 ($\neg$0)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)

2. Аналогично анализируется предпоследнее уравнение: $(x_5 \wedge \neg x_6) \vee (x_7 \wedge \neg x_8) = 0$.

Это уравнение также состоит из двух операндов, соединённых логической операцией ИЛИ, которая даёт нулевой результат, если оба операнда равны нулю:

$$\begin{cases}(x_5 \wedge \neg x_6) = 0,\\ (x_7 \wedge \neg x_8) = 0.\end{cases}$$

Так же, как для последнего уравнения, записываются в таблицу все возможные варианты значений переменных x_5, x_6, x_7, x_8, удовлетворяющие этим условиям. Поскольку структура обоих рассмотренных уравнений одинакова, заполнение таблицы будет точно таким же.

Таблица 2.1

$(x_5 \wedge \neg x_6)$	$(x_7 \wedge \neg x_8)$
0	0

Таблица 2.2

x_5	x_6	x_7	x_8
0	1 ($\neg$0)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)
0	0 ($\neg$1)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)
1	1 ($\neg$0)	0	1 ($\neg$0)
		0	0 ($\neg$1)
		1	1 ($\neg$0)

Для всех четырёх уравнений структура и заполнение таблицы возможных значений четырёх используемых в каждом уравнении переменных одна и та же. Смысл решения задачи проявляется во взаимосвязи между этими таблицами, отражающими взаимосвязь между уравнениями.

Следует обратить внимание, что переменные x_7 и x_8 используются в обоих рассмотренных уравнениях. Поэтому для каждого варианта (набора значений) переменных x_7 и x_8, приведённого в таблице 2.2, нужно определить количество наборов переменных x_7, x_8, x_9, x_{10}, имеющихся в таблице 1.2.

Таблица 2.2 (дополненная)

x_5	x_6	x_7	x_8	Кол-во вариантов с учётом x_9, x_{10}
0	1 (¬0)	0	1 (¬0)	3 ❶
		0	0 (¬1)	3 ❷
		1	1 (¬0)	3 ❸
0	0 (¬1)	0	1 (¬0)	3 ❶
		0	0 (¬1)	3 ❷
		1	1 (¬0)	3 ❸
1	1 (¬0)	0	1 (¬0)	3 ❶
		0	0 (¬1)	3 ❷
		1	1 (¬0)	3 ❸

Таблица 1.2

x_7	x_8	x_9	x_{10}
0 ❶	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
0 ❷	0 (¬1)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
1 ❸	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)

3. Аналогично анализируется второе по счёту уравнение: $(x_3 \wedge \neg x_4) \vee (x_5 \wedge \neg x_6) = 0$.

Таблица 3.1

$(x_3 \wedge \neg x_4)$	$(x_5 \wedge \neg x_6)$
0	0

Таблица 3.2

x_3	x_4	x_5	x_6
0	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
0	0 (¬1)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
1	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)

Переменные x_5 и x_6 используются в двух рассмотренных уравнениях (втором и третьем), поэтому для каждого варианта (набора значений) переменных x_5

и x_6 в таблице 3.2 нужно определить количество наборов переменных x_5, x_6, x_7, x_8, имеющихся в таблице 2.2 (с учётом ранее определённых количеств вариантов для x_7, x_8).

Таблица 3.2 (дополненная)

x_3	x_4	x_5	x_6	Кол-во вариантов с учётом x_7, x_8, x_9, x_{10}
0	1 (¬0)	0	1 (¬0)	9 (3+3+3) ❶
		0	0 (¬1)	9 (3+3+3) ❷
		1	1 (¬0)	9 (3+3+3) ❸
0	0 (¬1)	0	1 (¬0)	9 (3+3+3) ❶
		0	0 (¬1)	9 (3+3+3) ❷
		1	1 (¬0)	9 (3+3+3) ❸
1	1 (¬0)	0	1 (¬0)	9 (3+3+3) ❶
		0	0 (¬1)	9 (3+3+3) ❷
		1	1 (¬0)	9 (3+3+3) ❸

Таблица 2.2

x_5	x_6	x_7	x_8	Кол-во вариантов
0	1 (¬0)	0	1 (¬0)	3
		0	0 (¬1)	3
❶		1	1 (¬0)	3
0	0 (¬1)	0	1 (¬0)	3
		0	0 (¬1)	3
❷		1	1 (¬0)	3
1	1 (¬0)	0	1 (¬0)	3
		0	0 (¬1)	3
❸		1	1 (¬0)	3

4. Аналогично анализируется первое уравнение: $(x_1 \wedge \neg x_2) \vee (x_3 \wedge \neg x_4) = 0$.

Таблица 4.1

$(x_1 \wedge \neg x_2)$	$(x_3 \wedge \neg x_4)$
0	0

Таблица 4.2

x_1	x_2	x_3	x_4
0	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
0	0 (¬1)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)
1	1 (¬0)	0	1 (¬0)
		0	0 (¬1)
		1	1 (¬0)

Переменные x_3 и x_4 используются в двух уравнениях (первом и втором), поэтому для каждого варианта (набора значений) переменных x_3 и x_4 в таблице 4.2 нужно определить количество наборов переменных x_3, x_4, x_5, x_6, имеющихся в таблице 3.2 (с учётом ранее определённых количеств вариантов для x_5, x_6).

Таблица 4.2 (дополненная)

x_1	x_2	x_3	x_4	**Кол-во вариантов с учетом x_5, x_6, x_7, x_8, x_9, x_{10}**
0	1 (¬0)	0	1 (¬0)	27 (9+9+9) ❶
		0	0 (¬1)	27 (9+9+9) ❷
		1	1 (¬0)	27 (9+9+9) ❸
0	0 (¬1)	0	1 (¬0)	27 (9+9+9) ❶
		0	0 (¬1)	27 (9+9+9) ❷
		1	1 (¬0)	27 (9+9+9) ❸
1	1 (¬0)	0	1 (¬0)	27 (9+9+9) ❶
		0	0 (¬1)	27 (9+9+9) ❷
		1	1 (¬0)	27 (9+9+9) ❸
ИТОГО				9 27 = 243 варианта

Таблица 3.2

x_3	x_4	x_5	x_6	**Кол-во вариантов**
0	1 (¬0)	0	1 (¬0)	9
❶		0	0 (¬1)	9
		1	1 (¬0)	9
0	0 (¬1)	0	1 (¬0)	9
❷		0	0 (¬1)	9
		1	1 (¬0)	9
1	1 (¬0)	0	1 (¬0)	9
❸		0	0 (¬1)	9
		1	1 (¬0)	9

Ответ: 243 варианта.

Раздел 5. Элементы теории алгоритмов

A5 Анализ работы автомата, формирующего число по заданным правилам

Конспект

Автомат (греч. *autómatos* — самодействующий) — самостоятельно действующее устройство (или совокупность устройств), выполняющее по заданной программе без непосредственного участия человека процессы получения, преобразования, передачи и использования энергии, материала и информации[1].

Исполнитель — субъект (человек, животное) или устройство, способное выполнить действия, предписываемые алгоритмом. При этом указанные действия выполняются формально. Исполнитель не знает о цели алгоритма, он лишь выполняет все полученные команды.

Каждый исполнитель характеризуется следующими параметрами:

— **среда** — условия, в которых функционирует исполнитель (например, исполнитель Черепаха имеет определённую систему координат, исполнитель Робот перемещается по клетчатому полю и т.д.);

— **система команд** — каждый исполнитель может выполнять команды только из некоторого строго заданного набора, где каждая команда определяет соответствующее элементарное действие.

---

[1] Большая советская энциклопедия. — М.: Советская энциклопедия. 1969—1978.

Вспомогательные таблицы для решения задач с исполнителями — вычислителями

Система счисления	Основание (p)	Алфавит системы счисления	Пример записи числа
Двоичная	2	0, 1	101101_2
Восьмеричная	8	0, 1, 2, 3, 4, 5, 6, 7	12345_8
Десятичная	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	1234_{10}
Шестнадцатеричная	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (=10), B (=11), C (=12), D (=13), E (=14), F (=15)	$F4D9_{16}$

Основание системы счисления	2	3	4	5	6	7	8	9
Max цифра	1	2	3	4	5	6	7	8
Max сумма цифр	10_2	11_3	12_4	13_5	14_6	15_7	16_8	17_9

Основание системы счисления	10	11	12	13	14	15	16
Max цифра	9	A	B	C	D	E	F
Max сумма цифр	18_{10}	19_{11}	$1A_{12}$	$1B_{13}$	$1C_{14}$	$1D_{15}$	$1E_{16}$

Разбор типовых задач

Задача 1*. Автомат получает на вход два трёхзначных числа. По этим числам строится новое число по следующим правилам:

1. Вычисляются три числа — сумма старших разрядов заданных трёхзначных чисел, сумма средних разрядов этих чисел, сумма младших разрядов.
2. Полученные три числа записываются друг за другом в порядке убывания (без разделителей).

Пример. Исходные трёхзначные числа: 835, 196. Поразрядные суммы: 9, 12, 11. Результат: 12119.

Определите, какое из следующих чисел может быть результатом работы автомата.

1) 151303
2) 161410
3) 191615
4) 121613

Решение

В данной задаче речь идёт о сложении цифр чисел, которые неизвестны. Пусть эти числа обозначаются как X и Y и записываются в виде:

$X = x_1x_2x_3$; $Y = y_1y_2y_3$

Суммы цифр, которые вычисляет автомат, есть:

$S_1 = x_1 + y_1$, $S_2 = x_2 + y_2$ и $S_3 = x_3 + y_3$

Складывая цифры, оцениваются возможные значения этих сумм. Старшая цифра в двузначном числе может иметь значение от 1 до 9 (если она равна нулю, то число уже не будет трёхзначным!), а средняя и младшая — значение от 0 до 9. Тогда сумма $S_1 = x_1 + y_1$ может иметь значение от 2 до 18 (исходных чисел — два), а суммы $S_2 = x_2 + y_2$ и $S_3 = x_3 + y_3$ — значение от 0 до 18.

Теперь анализируются предлагаемые варианты ответов.

1. *Число 151303.* Оно может быть составлено из сумм 15, 13 и 03. Но третья сумма (03) имеет незначащий нуль, тогда как в условии не указано, что после сложения цифр исходных чисел автомат должен дополнять полученную сумму нулём до двузначного числа. Поэтому данный вариант ответа не может быть правильным.

2. *Число 161410.* Его можно представить как сочетание сумм 16, 14 и 10. Это вполне удовлетворяет условиям задачи (все три суммы входят в допустимый интервал и записаны по убыванию). Однако желательно проверить и остальные варианты.

3. *Число 191615.* Может быть представлено как сочетание сумм 19, 16 и 15. Однако значение суммы 19 выходит за пределы допустимого диапазона (должно быть не более 18), поэтому данный вариант числа нам не подходит.

4. *Число 121613.* Может быть представлено как сочетание сумм 12, 16 и 13. Эти значения соответствуют допустимому интервалу, но записаны не по убыванию. Значит, данный вариант ответа тоже ложный.

Следовательно, правильным является ответ 161410.

Ответ: число 161410 (вариант ответа №2).

Задача 2. Устройство считывает четырёхзначное восьмеричное число и строит по нему новое число по следующему алгоритму:

1) вычисляется сумма первой и второй цифр;

2) вычисляется сумма третьей и четвёртой цифр;

3) эти суммы записываются друг за другом без разделителей по убыванию.

Пример. Задано число 7145. Суммы:

7 + 1 = 10; 4 + 5 = 11. Результат: 1110.

Какое из чисел может быть результатом работы такого устройства.

1) 119 2) 1213 3) 1411 4) 1715

Решение

Вычисления производятся в недесятичной системе счисления.

Пусть неизвестные цифры числа обозначаются как x, y, z и t. Подразумевается, что само число тогда имеет вид: $xyzt$.

Суммы цифр, которые вычисляет автомат, — это:

$S_1 = x + y$ и $S_2 = z + t$

Складывая восьмеричные цифры, которые могут меняться от 0 до 7, оцениваются возможные значения этих сумм. Тогда, согласно правилам восьмеричной арифметики, сумма двух таких цифр может меняться от 0 до $7_8 + 7_8 = 16_8$.

Теперь просматриваются предлагаемые варианты ответов.

1. *Число 119.* Можно представить его как 11 и 9 или как 1 и 19. Однако ни один из этих вариантов записи нам не годится: в первом случае восьмеричная запись суммы не может содержать цифры больше 7, а во втором число 19 не может быть суммой двух восьмеричных цифр.

2. *Число 1213.* Его можно представить как 12 и 13. Оба эти числа могут быть суммами восьмеричных цифр (так как удовлетворяют допустимому диапазону значений таких сумм). Однако они записаны по возрастанию (а не по убыванию, как требуется в условии задачи). Поэтому данное число не может быть решением.

3. *Число 1411*. Его можно представить как 14 и 11. Обе эти составляющие могут быть значениями сумм восьмеричных цифр, записаны они по убыванию. Значит, это число может быть решением данной задачи.

4. *Число 1715*. Может быть представлено как 17 и 15. Поскольку его первая составляющая (17) превышает максимально возможное значение суммы, данное число не может быть решением задачи.

Следовательно, единственный вариант ответа, который может быть результатом работы описанного в задаче автомата, — это число 1411.

Ответ: число 1411 (вариант ответа №3).

Задача 3*. Автомат получает на вход два трёхзначных числа. По этим числам строится новое число по следующим правилам:

1. Вычисляются три числа — сумма старших разрядов заданных трёхзначных чисел, сумма средних разрядов этих чисел, сумма младших разрядов.

2. Полученные три числа записываются друг за другом в порядке убывания (без разделителей).

Пример. Исходные трёхзначные числа: 835, 196. Поразрядные суммы: 9, 12, 11. Результат: 12119.

Определите, какое из следующих чисел может быть результатом работы автомата.

1) 15012 3) 121111
2) 191313 4) 10911

Решение

Пусть неизвестные нам числа обозначаются как $X = x_1x_2x_3$ и $Y = y_1y_2y_3$.

Суммы цифр, которые вычисляет автомат, — это:

$S_1 = x_1 + y_1$, $S_2 = x_2 + y_2$ и $S_3 = x_3 + y_3$.

Используется десятичная система счисления, поэтому цифры могут меняться от 0 до 9, а их сумма может меняться от 0 до 18.

Теперь просматриваются предлагаемые варианты ответов.

1. *Число 15012*. Можно представить его следующим образом:

- 1, 50, 12 — число 50 превышает максимально возможное значение суммы цифр;
- 15, 0, 12 — значения составляющих допустимы, но записаны не по убыванию.

Поэтому данное число не может быть решением задачи.

2. *Число 191313.* Его можно представить как 19, 13 и 13. Первая составляющая превышает максимально допустимую величину суммы двух десятичных цифр. Поэтому данное число тоже не может быть решением задачи.

3. *Число 121111.* Его можно представить как 12, 11 и 11. Все три составляющие соответствуют диапазону возможных значений суммы двух десятичных цифр и записаны по убыванию (невозрастанию). Следовательно, это число может являться решением задачи.

4. *Число 10911.* Можно представить его следующим образом:

- 10, 9, 11 — значения составляющих допустимы, но записаны не по убыванию;
- 10, 91, 1 — число 91 превышает максимально возможное значение суммы цифр.

Поэтому данное число не может быть решением задачи.

Следовательно, единственный вариант ответа, который может быть результатом работы описанного в задаче автомата, — это число 121111.

Ответ: число **121111** (вариант ответа №3).

Задача 4. Алёша и Лена придумали такую игру с числами. Лена записывает четырёхзначное шестнадцатеричное число, в котором все цифры не превышают 6, а Алёша придумывает новое шестнадцатеричное число по следующему алгоритму:

1) вычисляется шестнадцатеричная сумма двух первых разрядов числа, записанного Леной, и сумма двух последних разрядов этого числа;

2) две вычисленные шестнадцатеричные суммы записываются друг за другом без разделителей по убыванию.

Пример. Лена записала число 3456. Поразрядные суммы: 7, B. Число, придуманное Алёшей: B7.

Какое из чисел может получиться у Алёши (исходное число, записанное Леной, неизвестно).

1) 93 2) D5 3) 119 4) 6B

Решение

В решении при определении возможного диапазона значений сумм нужно учитывать, что в задаче используется шестнадцатеричная система счисления, а на значения цифр исходного числа наложено ограничение: они не могут быть больше 6.

Пусть неизвестные цифры числа обозначаются как x, y, z и t. Подразумевается, что само число тогда имеет вид: $xyzt$.

Суммы цифр, которые вычисляет автомат, — это:

$S_1 = x + y$ и $S_2 = z + t$

Складывая шестнадцатеричные цифры, которые по условию задачи могут меняться от 0 до 6, оцениваются возможные значения этих сумм. Тогда, согласно правилам шестнадцатеричной арифметики, сумма двух таких цифр может меняться от 0 до $6_{16} + 6_{16} = C_{16}$ (12_{10}).

Теперь просматриваются предлагаемые варианты ответов.

1. *Число 93.* Можно представить его как 9 и 3. Обе эти составляющие входят в допустимый диапазон значений суммы, записаны они по убыванию, значит, данное число может быть решением задачи.

2. *Число D5.* Его можно представить как D и 5. Поскольку первая составляющая (D) превышает максимальное значение шестнадцатеричной суммы двух шестёрок, данное число не может быть решением.

3. *Число 119.* Его можно представить как 11 и 9 или как 1 и 19. В обоих вариантах представления числа в нём имеются составляющие, не являющиеся шестнадцатеричными цифрами (11 и 19) и превышающие максимально возможное значение суммы двух шестнадцатеричных шестёрок. Значит, это число тоже не может быть решением данной задачи.

4. *Число 6B*. Может быть представлено как 6 и B. Обе эти составляющие могут быть шестнадцатеричными суммами двух шестерок, но записаны они не по убыванию. Поэтому данное число не может быть решением задачи.

Единственный вариант ответа, который может быть результатом работы описанного в задаче автомата, — это число 93.

Ответ: число 93 (вариант ответа №1).

A13 Робот в лабиринте

Конспект

Исполнитель — любое устройство или живое существо, которое способно точно и однозначно выполнять заданные команды (действия, заданные алгоритмом).

Система команд исполнителя — совокупность команд, которые он способен выполнять.

Алгоритм — запись в той или иной форме (словесной, графической, на языке программирования) последовательности команд для исполнителя. Команды алгоритма должны соответствовать системе команд исполнителя.

Исполнитель РОБОТ — перемещается по клеткам лабиринта.

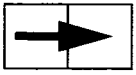
Типичная система команд:

- перемещение: вверх, вниз, влево, вправо;
- проверка наличия препятствия (стенки): сверху свободно, снизу свободно, слева свободно, справа свободно — играют роль условия;
- команда ветвления:

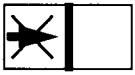
```
ЕСЛИ <условие>
          ТО команда1
          ИНАЧЕ команда2
КОНЕЦ ЕСЛИ
```

— выполняется `команда1` (если `условие` истинно) или `команда2` (если `условие` ложно), где условие — это команда проверки наличия препятствия;

• При совпадении направления проверки наличия препятствия совпадает с направлением движения Робота: ЕСЛИ <справа свободно> ТО вправо



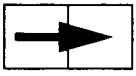
Стенки нет —
Робот перемещается
в соседнюю клетку



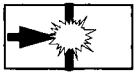
Стенка есть —
Робот остаётся на месте

Проверка наличия препятствия выполняется ***до*** начала движения Робота.

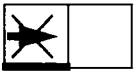
• При несовпадении направления проверки наличия препятствия совпадает с направлением движения Робота: ЕСЛИ <снизу свободно> ТО вправо



Стенки нет ни снизу,
ни в направлении
движения —
Робот перемещается
в соседнюю клетку



Стенки снизу нет,
но есть стенка
в направлении движения —
Робот начинает двигаться
и разбивается



Есть стенка снизу —
Робот остаётся на месте

До начала движения Робота проверяется наличие препятствия с указанной стороны. Если оно есть, то Робот остаётся на месте. Если его нет, то начинается движение Робота в заданном направлении, даже если по направлению его движения имеется стенка (в последнем случае Робот разбивается).

• команда цикла:

```
ПОКА <условие> команда
```

или

```
ПОКА <условие>
      последовательность команд
КОНЕЦ ПОКА
```

— выполняется, пока условие истинно (условие — это команда проверки наличия препятствия).

Если в цикле ПОКА размещено несколько операторов ЕСЛИ, то:

- проверяется условие выполнения цикла; если оно ложно, то цикл не выполняется вообще;
- выполняется первый оператор ЕСЛИ (согласно заданному в нём условию);
- выполняется второй оператор ЕСЛИ (согласно заданному в нём условию);

  ...
- выполняется последний оператор ЕСЛИ (согласно заданному в нём условию);
- только после завершения выполнения всех операторов ЕСЛИ, записанных в теле цикла, выполняется новая проверка условия в цикле ПОКА для выяснения, должен ли цикл быть повторен.

Разбор типовых задач

Задача 1*. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

вверх	вниз	влево	вправо

При выполнении любой из этих команд РОБОТ перемещается на одну клетку соответственно: вверх ↑, вниз ↓, влево ←, вправо →.

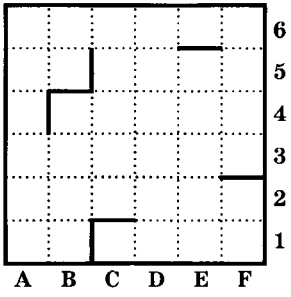
Четыре команды проверяют истинность условия отсутствия стены у каждой стороны той клетки, где находится РОБОТ:

сверху свободно	снизу свободно	слева свободно	справа свободно

Цикл ПОКА *<условие> команда*
выполняется, пока *условие* истинно, иначе происходит переход на следующую строку.

Сколько клеток лабиринта соответствуют требованию, что, выполнив предложенную программу, РОБОТ остановится в той же клетке, с которой он начал движение?

НАЧАЛО
ПОКА <сверху свободно> вправо
ПОКА <справа свободно> вниз
ПОКА <снизу свободно> влево
ПОКА <слева свободно> вверх
КОНЕЦ



1) 1 2) 2 3) 3 4) 4

Решение

Проще всего решить эту задачу, что называется, «в лоб», — поочерёдно перебирая *каждую* клетку лабиринта и «проводя» робота из неё по маршруту согласно заданной программе. Однако, такой способ требует слишком много времени, поскольку приходится проверять 36 вариантов (клеточек). Поэтому основная задача — научиться заранее отсекать заведомо неподходящие варианты, чтобы найти решение достаточно быстро (так как время на ЕГЭ ограничено).

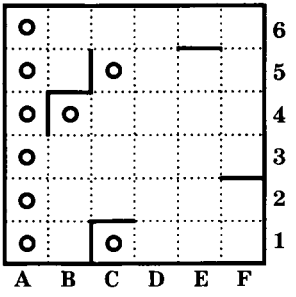
Прежде всего следует обратить внимание: направления движения и условия остановки Робота в каждой команде ПОКА различны. Поэтому при решении такой задачи следует быть более внимательными: РОБОТ либо может остановиться, когда стенка (как условие остановки движения) появится в очередной клетке, в которую он перейдёт, с соответствующего бока, либо может разрушиться, если первой встретится стенка, перекрывающая ему путь (тогда как с соответствующего бока стенки, которая бы его остановила, нет). В условии таких задач обычно особо оговаривается факт разрушения РОБОТа, натолкнувшегося на полном ходу на стенку, хотя в данном случае оно опущено.

Далее, нужно обратить внимание на то, что РОБОТ, с какой бы клеточки он ни начинал движение, однозначно останавливается (согласно программе) в клетке, где с требуемой стороны имеется стенка. Поэтому проще всего сначала определить клеточки, в которых

РОБОТ должен согласно его программе завершать движение. Таких клеточек будет не так много, а дальше можно проверять каждую из них на соответствие условию: мог ли РОБОТ прийти в эту клетку, начав движение с неё же?

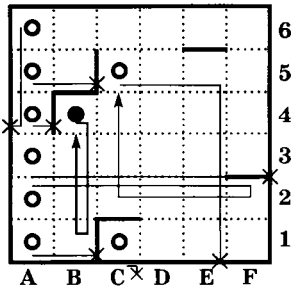
В данном случае РОБОТ завершает программу, если обнаруживает стенку слева.

На схеме лабиринта отмечаются все такие точки, — их оказывается всего 9 (а не 36):



Теперь они поочерёдно перебираются и проверяются, сможет ли РОБОТ, начав путь из какой-либо точки, вернуться в неё. При этом следует учесть, что если движение в каком-то направлении невозможно, так как оно изначально «запрещено» стенкой, то соответствующая строка программы пропускается, и необходимо сразу переходим к следующей, а если РОБОТ при движении сталкивается со стенкой, то выполнение программы прерывается, так как РОБОТ разрушился.

На рисунке ниже показаны маршруты движения РОБОТа из каждой отмеченной точки (крестиком показаны места разрушения РОБОТа при движении на стенку):



Итого, есть только одна клеточка, удовлетворяющая условию.

Ответ: 1 клеточка (вариант ответа №1).

В подобных задачах в командах ПОКА проверяется наличие соответствующей стенки (левой, правой, верхней или нижней) по отношению к **текущей** ячейке, в которой находится РОБОТ, а не по отношению к самому РОБОТу с учётом направления его движения.

Задача 2*. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

вверх	вниз	влево	вправо

При выполнении любой из этих команд РОБОТ перемещается на одну клетку соответственно: вверх ↑, вниз ↓, влево ←, вправо → .

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

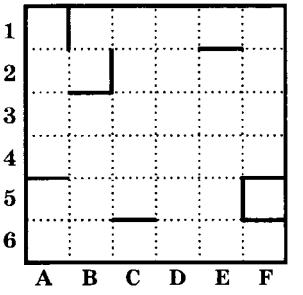
сверху свободно	снизу свободно	слева свободно	справа свободно

Цикл ПОКА <*условие*> *команда*
выполняется, пока *условие* истинно, иначе происходит переход на следующую строку.

Если РОБОТ начнёт движение в сторону находящейся рядом с ним стены, то он разрушится и программа прервётся.

Сколько клеток лабиринта соответствуют требованию, что, выполнив предложенную программу, РОБОТ уцелеет и остановится в той же клетке, с которой он начал движение?

НАЧАЛО
ПОКА <справа свободно> вниз
ПОКА <снизу свободно> влево
ПОКА <слева свободно> вверх
ПОКА <сверху свободно> вправо
КОНЕЦ

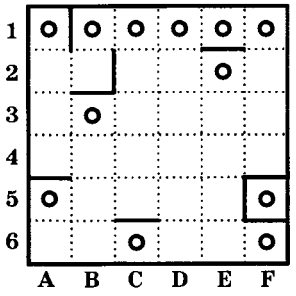


1) 1 2) 2 3) 5 4) 7

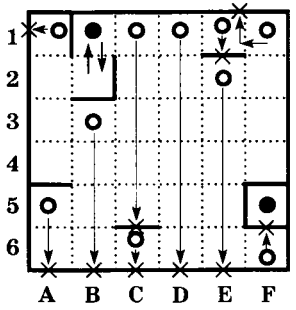
Решение

Данная задача и её решение аналогичны предыдущей.

Вначале размечаются клетки окончания пути по признаку — стенка стоит *сверху* от них:



Потом проверяются найденные клетки (как и раньше, крестиком показаны места разрушения РОБОТа при движении на стенку):



Итого есть одна клетка, в которую РОБОТ возвращается по завершении выполнения программы, и ещё одна клетка, замкнутая со всех сторон, так что РОБОТ вообще будет все время оставаться на месте (что тоже удовлетворяет условиям задачи).

Ответ: 2 клетки (вариант ответа №2).

Задача 3. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

вверх	вниз	влево	вправо

При выполнении любой из этих команд РОБОТ перемещается на одну клетку соответственно: вверх ↑, вниз ↓, влево ←, вправо →.

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

сверху свободно	снизу свободно	слева свободно	справа свободно

Цикл

```
ПОКА <условие> последовательность команд
КОНЕЦ ПОКА
```

выполняется, пока условие истинно.

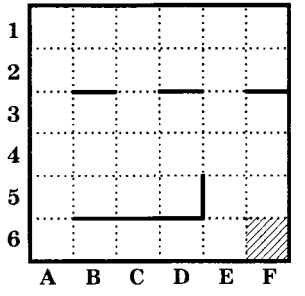
В конструкции

```
ЕСЛИ <условие>
          ТО команда1
          ИНАЧЕ команда2
КОНЕЦ ЕСЛИ
```

выполняется команда1 (если условие истинно) или команда2 (если условие ложно).

Если РОБОТ начнёт движение в сторону находящейся рядом с ним стены, то он разрушится и программа прервётся.

Сколько клеток лабиринта соответствуют требованию, что, начав движение в ней и выполнив предложенную программу, РОБОТ уцелеет и остановится в закрашенной клетке (клетка **F6**)?



```
НАЧАЛО
ПОКА <справа свободно ИЛИ снизу свободно>
        ПОКА <справа свободно> вправо
        КОНЕЦ ПОКА
        ПОКА <снизу свободно> вниз
        КОНЕЦ ПОКА
КОНЕЦ ПОКА
КОНЕЦ
```

1) 8 2) 12 3) 16 4) 20

Решение

Особенность этой задачи по сравнению с предыдущими — в том, что здесь программа для РОБОТа состоит не из линейной последовательности из нескольких циклов ПОКА, каждый из которых определяет количество шагов движения в одном из четырёх направлений, а из вложенных циклов ПОКА. Соответственно, иным будет и путь решения задачи.

Анализируется заданная в условии программа для РОБОТа.

Во-первых, здесь вместо компактной однооператорной записи циклов ПОКА (например: `ПОКА <справа свободно> вниз`) используется полная запись с использованием особой завершающей команды `КОНЕЦ ПОКА`. Однако это — лишь формальное изменение, важное только для обрамляющего цикла ПОКА, тогда как внутренние (вложенные) циклы легко представить в прежней, компактной форме:

```
НАЧАЛО
ПОКА <справа свободно ИЛИ снизу свободно>
        ПОКА <справа свободно> вправо
        ПОКА <снизу свободно> вниз
КОНЕЦ ПОКА
КОНЕЦ
```

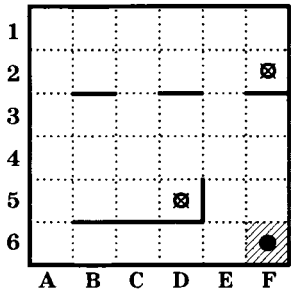
В этом виде программа становится несколько более простой и понятной; в ней легче выявить аналогию с традиционным вариантом задач про РОБОТа.

Во-вторых, внутренние (вложенные) циклы ПОКА, как и раньше, определяют перемещение РОБОТа в соответствующем направлении, пока с указанной стороны от него не будет обнаружена стенка: первый вложенный цикл ПОКА определяет движение вправо до стенки справа, а второй цикл ПОКА — движение вниз до стенки снизу. В сумме же это означает, что РОБОТ должен двигаться «Г-образно»: сначала вправо до препятствия, а потом вниз до препятствия.

В-третьих, внешний цикл ПОКА и его условие <справа свободно ИЛИ снизу свободно> показывают, что вся «Г-образная» траектория движения РОБОТа может повторяться многократно, если после остановки РОБОТа из-за препятствия в виде стенки внизу выяснится, что справа стенки нет: тогда РОБОТ может снова двигаться вправо до препятствия вправо, а потом вниз до препятствия снизу. Особо заметим: согласно принципам работы цикла ПОКА, условие внешнего цикла проверяется тогда, и только тогда, когда РОБОТ уже завершил проход очередного «Г-образного» фрагмента пути, а не, например, по завершении выполнения первого из двух вложенных циклов ПОКА или в процессе их выполнения. То есть отработка каждого из внутренних циклов ПОКА является процессом, полностью независимым от условия внешнего цикла.

Наконец, запись условия внешнего цикла: <справа свободно ИЛИ снизу свободно> означает, что завершение выполнения этого внешнего цикла ПОКА и, соответственно, завершение работы программы РОБОТа происходит, когда препятствие (стенка) имеется И справа, И снизу от текущего расположения РОБОТа. Следовательно, «финальной» ячейкой (имеется в виду «благополучный» финал при корректном завершении работы программы, а не авария, когда РОБОТ разбивается о «незамеченную» им стенку; впрочем, структура вложенных циклов ПОКА такой аварийный случай исключает) может быть только ячейка, ограниченная стенками снизу и справа.

В представленном лабиринте:



этим условиям соответствуют только три ячейки: F6 (требуемая, на рисунке выше отмеченная кружком), F2 и D5 («ложные», на рисунке отмеченные крестиками). При этом «ложные» ячейки можно рассматривать как «ловушки» для РОБОТа, попадание в которые делает требуемый результат работы его программы невозможным.

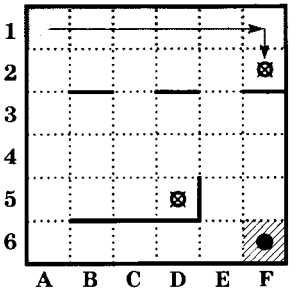
Теперь, возвращаясь к вложенным циклам ПОКА, следует проанализировать как РОБОТ будет согласно этим командам двигаться по лабиринту.

Очевидно, что первый вложенный цикл ПОКА для *любой* исходной ячейки определяет одно и то же движение РОБОТа вправо до ближайшей стенки справа. Следовательно, можно разделить весь лабиринт на такие фрагменты (последовательности ячеек в одной строке), которые завершаются стенкой справа (внешней границей лабиринта или промежуточной стенкой). И если хотя бы одна ячейка каждого такого фрагмента удовлетворяет либо, наоборот, не удовлетворяет условию задачи (является либо не является решением задачи), то тогда и *все* ячейки соответствующего фрагмента являются либо не являются решением. В данном лабиринте эти фрагменты (для их обозначения можно использовать привычную по электронным таблицам запись диапазонов) будут следующими:

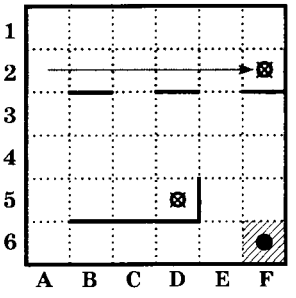
A1:F1, A2:F2, A3:F3, A4:F4, A5:D5, E5:F5, A6:F6.

Учитывая сказанное выше, не обязательно проверять все ячейки лабиринта — достаточно в каждом диапазоне проверить только по одной любой ячейке — например, крайней слева (т.е. ячейки A1, A2, A3, A4, A5, E5 и A6). Однако, в случае успешного перемещения РОБОТа из проверяемой ячейки в «целевую» ячейку F6 нужно подсчитать *все* ячейки соответствующего диапазона, как удовлетворяющие условию задачи.

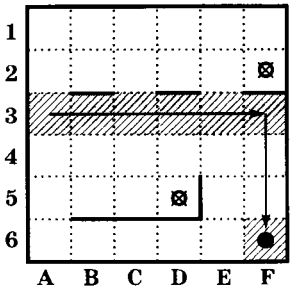
Ячейка A1. Движение РОБОТа вправо будет происходить до правой границы лабиринта, после чего он будет двигаться вниз до ячейки F2 («ловушки»), отмеченной крестиком. Поэтому ни одна ячейка диапазона A1:F1 не подходит.



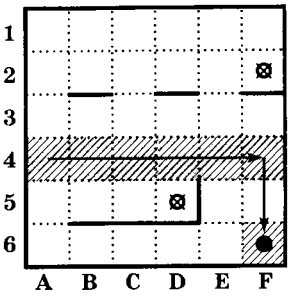
Ячейка A2. РОБОТ из неё тоже будет двигаться вправо до правой границы лабиринта (до ячейки-«ловушки» F2), после чего в ней и останется, — второй вложенный цикл ПОКА просто не будет выполняться, так как снизу от ячейки F2 есть стенка. Значит, ни одна ячейка диапазона A2:F2 тоже не подходит.



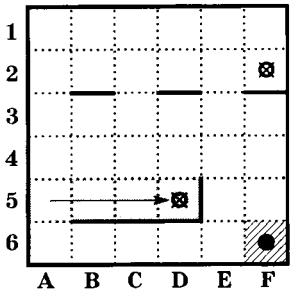
Ячейка A3. Движение РОБОТа вправо будет совершаться до правой границы лабиринта, а затем РОБОТ пойдёт вниз до требуемой ячейки F6. Значит, *все* ячейки диапазона A3:F3 (их — 6) являются решениями данной задачи.



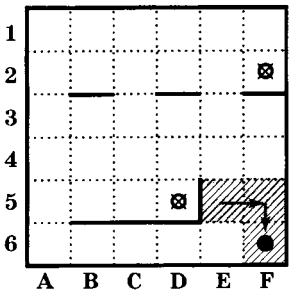
Ячейка A4. Повторив те же самые рассуждения для неё, выясняется, что *все* ячейки диапазона A4:F4 также являются решениями задачи (это ещё 6 ячеек).



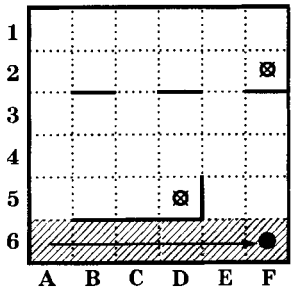
Ячейка A5. Очевидно, что РОБОТ при движении вправо попадёт в ячейку-«ловушку» D5 и останется в ней (так как вниз идти ему некуда). Поэтому *все* ячейки диапазона A5:D5 непригодны.



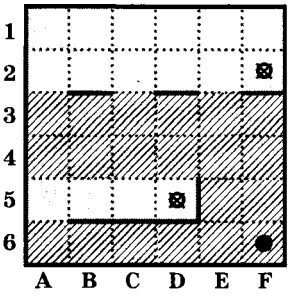
Ячейка E5. Здесь, напротив, РОБОТ, выполняя свою программу, благополучно дойдёт до нужной ячейки F6. Тем самым определились ещё 2 ячейки диапазона E5:F5, которые являются решением задачи.



Ячейка A6. Для неё, а значит, и для *всех* 6 ячеек диапазона A6:F6 выполнение РОБОТом заданной программы благополучно приведёт его в нужную ячейку F6.



Итого решениями задачи являются: 6 ячеек диапазона A3:F3, 6 ячеек диапазона A4:F4, 2 ячейки диапазона E5:F5 и 6 ячеек диапазона A6:F6 (включая «целевую» F6: если РОБОТ изначально в ней находился, то в ней он и останется, т.е. условие задачи тоже будет выполнено), то есть всего —



6 + 6 + 2 + 6 = 20 ячеек.

Ответ: 20 клеток (вариант ответа №4).

Задача 4. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

вверх	вниз	влево	вправо

При выполнении любой из этих команд РОБОТ перемещается на одну клетку соответственно: вверх ↑, вниз ↓, влево ←, вправо →.

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

сверху свободно	снизу свободно	слева свободно	справа свободно

Цикл

```
ПОКА <условие> последовательность команд
КОНЕЦ ПОКА
```

выполняется, пока условие **истинно.**

В конструкции

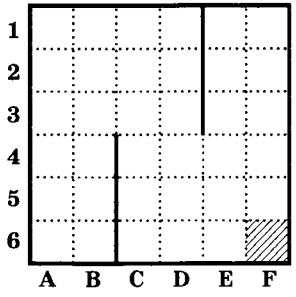
```
ЕСЛИ <условие>
        ТО команда1
        ИНАЧЕ команда2
КОНЕЦ ЕСЛИ
```

выполняется команда1 **(если** условие **истинно) или** команда2 **(если** условие **ложно)**

В конструкциях ПОКА и ЕСЛИ условие может быть составным — с использованием логических операций И, ИЛИ, НЕ.

Если РОБОТ начнёт движение в сторону находящейся рядом с ним стены, то он разрушится и программа прервётся.

Сколько клеток лабиринта соответствуют требованию, что, начав движение в ней и выполнив предложенную программу, РОБОТ уцелеет и остановится в закрашенной клетке (клетка **F6**)?




```
НАЧАЛО
ПОКА <справа свободно ИЛИ снизу свободно>
        ЕСЛИ <справа свободно> вправо
        КОНЕЦ ЕСЛИ
        ЕСЛИ <снизу свободно> вниз
        КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
КОНЕЦ
```

Решение

Отличие этой задачи от задачи №4 в том, что здесь внутри цикла ПОКА вложены операторы ЕСЛИ (а не другие циклы ПОКА).

В результате, если в предыдущей задаче каждый вложенный оператор ПОКА вызывал движение РОБОТа в соответствующем направлении на произвольное число клеток до ограничивающей это движение стенки, то в данной задаче оба вложенных оператора ЕСЛИ реализуют переход в соответствующем направлении (если нет

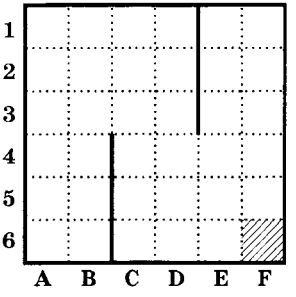
ограничивающей стенки) только на ***одну*** клетку. Таким образом, анализ перемещений РОБОТа производится не с «полосками» клеток (строками или столбцами), а с «зигзагообразными» смещениями (в данном случае вправо и вниз) на каждом очередном проходе цикла ПОКА.

Соответственно, меняется и вид решения.

1. Анализируя условие выполнения цикла ПОКА: `<справа свободно ИЛИ снизу свободно>` означает, что данный цикл завершается (РОБОТ останавливается) в клетке, имеющей стенки **одновременно** справа и снизу, т.е. ⌐.

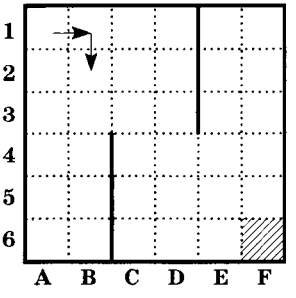
2. Анализируям тело цикла ПОКА: оно состоит из двух операторов ЕСЛИ, где первый выполняет перемещение РОБОТа на ***одну*** клетку вправо (если на этом пути нет стенки), а второй — перемещение РОБОТа на **одну** клетку вниз (также если на этом пути нет стенки). После этого вновь проверяется условие цикла ПОКА, если оно истинно, то снова выполняется пара операторов ЕСЛИ, и т.д.

3. Анализируя клетчатое поле, выделяются на нём диапазоны с одинаковыми условиями «поведения» РОБОТа.

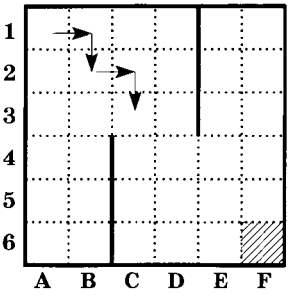


Для удобства клетки можно пронумеровать, как это принято в электронных таблицах, например, угловая клетка вверху слева имеет номер A1.

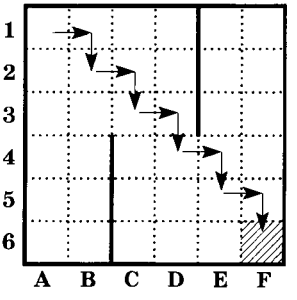
3.1. **Клетка A1**: не является конечной (не имеет стенок справа и снизу), поэтому РОБОТ из неё переместится сначала в клетку B1, а затем — в клетку B2:



Клетка B2 также не является конечной. Из неё РОБОТ перейдёт сначала в клетку C2, а затем в клетку C3:

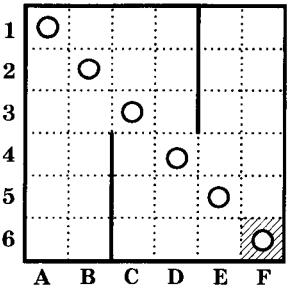


Движение дальше по клеткам продолжается аналогичным образом. Выделяются пары, соответствующие отдельным проходам цикла ПОКА: D3–D4, E4–E5, F5–F6:

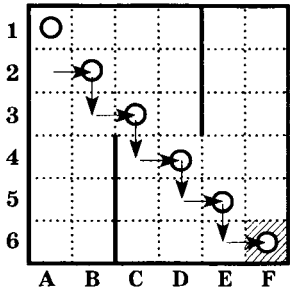


Очередной цикл завершён, а при попытке выполнения следующего его прохода будет обнаружено, что условие цикла уже ложно (так как клетка F6 — конечная). Поэтому выполнение программы завершится, и РОБОТ окажется в искомой клетке.

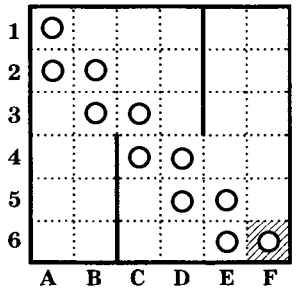
Анализируя пройденный РОБОТом путь, можно сказать, что условию «РОБОТ уцелеет и остановится в закрашенной клетке F6» соответствует как исходная клетка A1, так и все пройденные клетки, в которых начинался новый проход цикла ПОКА («ключевые» клетки): это клетки B2, C3, D4, E5 и собственно F6, потому что если РОБОТ изначально находится в этой клетке, то условие цикла ПОКА изначально ложно, и РОБОТ так в этой клетке и останется. Точки отмечаются кружками:



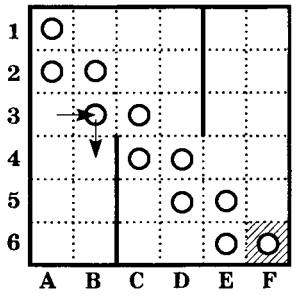
3.2. **Клетка A2**: аналогично ранее рассмотренному, РОБОТ сначала перейдёт по клеткам B2–B3, а затем по клеткам C3–C4, D3–D4 и наконец, E5–E6:



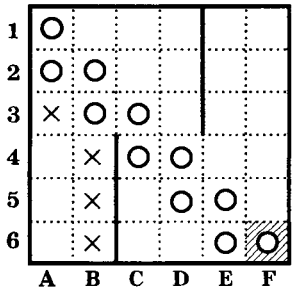
В клетке E6 есть только стенка снизу, т.е. она — не конечная (условие цикла ПОКА всё ещё выполняется). Поэтому будет выполнен первый оператор ЕСЛИ, и РОБОТ перейдёт в ячейку F6. А вот условие второго ЕСЛИ уже будет ложно (снизу — стенка), поэтому второй оператор ЕСЛИ будет пропущен. РОБОТ благополучно пришёл в конечную клетку F6, поэтому отмечаются кружками все «ключевые» клетки и этого маршрута:



3.3. **Клетка А3**: из неё РОБОТ сначала перейдёт в клетку В3, а затем — в клетку В4:

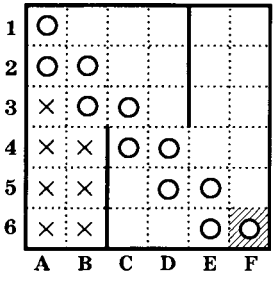


Клетка В4 — не конечная (нет стенки снизу), поэтому цикл ПОКА всё ещё будет работать. Так как справа есть стенка, первый оператор ЕСЛИ будет пропущен, а вот второй оператор ЕСЛИ сработает, и РОБОТ переместится в клетку В5. Поскольку для неё справедливы все высказанные выше рассуждения о клетке В4, РОБОТ далее спустится в ячейку В6. Она — конечная (имеет стенки и снизу, и справа), поэтому РОБОТ остаётся в ней. Но это — «неправильная» клетка (т.е. «ловушка»), поэтому на данном маршруте РОБОТа помечаются все «ключевые» клетки крестиками:

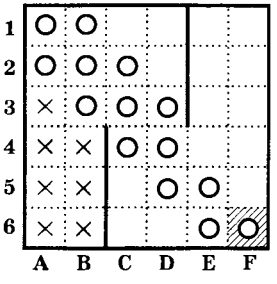


Попадание РОБОТа в «ловушку» В6 будет происходить и когда РОБОТ начнёт движение из клеток А4, А5 и А6. В последнем случае РОБОТ первым оператором ЕСЛИ сместится опять же в В6, а второй ЕСЛИ будет

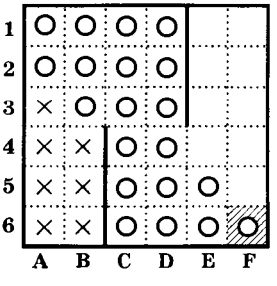
пропущен. Значит, и эти «ключевые» клетки помечаются крестиками:



3.4. Клетка B1: РОБОТ пойдёт по клеткам: C1–C2 и D2–D3. Клетка D3 — не конечная (нет стенки снизу). Поскольку справа есть стенка, первый оператор ЕСЛИ пропускается, а второй выполняется: РОБОТ перейдёт в клетку D4 и дальше «пойдёт по накатанному пути». Вывод: указанные «ключевые» клетки B1, C2 и D3 можно отметить кружочками:

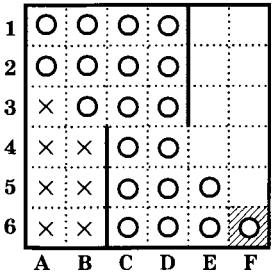


Точно так же РОБОТ поведёт себя и начиная движение из клеток C1, D1 и D2:

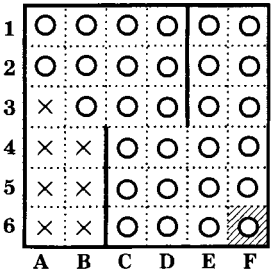


«Натыкаясь» по пути на стенку справа, он будет пропускать первый оператор ЕСЛИ и смещаться вниз, пока стенка не закончится, а дальше пойдёт «своим путём» в клетку F6.

В начале пути из клеток C5, C6 и D6 РОБОТа будет ограничивать стенка снизу (что означает пропуск второго оператора ЕСЛИ), и он будет продвигаться вдоль этой нижней стенки по горизонтали вплоть до целевой клетки F6:



3.5. **Клетка E1**, а также все остальные ещё не рассмотренные клетки: начиная из них движение, РОБОТ будет смещаться вправо и вниз до правой стенки, после чего спокойно «доедет» вниз вдоль этой стенки (за счёт пропусков первого ЕСЛИ) до все той же целевой клетки F6. Значит, все эти клетки годятся:



Остаётся подсчитать количество клеток, отмеченных кружочками. Их 29.

Ответ: 29 клеток.

Задача 5*. Исполнитель Робот ходит по клеткам бесконечной вертикальной клетчатой доски, переходя по одной из команд `вверх`, `вниз`, `вправо`, `влево` в соседнюю клетку в указанном направлении. Робот выполнил следующую программу:

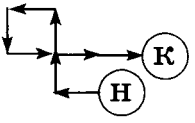
```
влево
вверх
вверх
влево
вниз
вправо
вправо
вправо
```

Укажите наименьшее возможное число команд в программе, приводящей Робота из той же начальной клетки в ту же конечную.

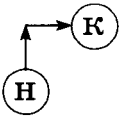
Решение

В данной задаче вид лабиринта неизвестен, поэтому для решения требуется опираться только на информацию об успешности прохождения заданного маршрута. Необходимо построить такую эквивалентную программу, чтобы Робот шёл по тем же самым клеткам, что и по исходной программе.

На рисунке приведён окончательный вид маршрута Робота, получаемый по исходной программе, где «н» — начало, «к» — конец:



В этом маршруте имеется петля, которую можно устранить, тем самым сократив длину программы. Кроме того, можно из начальной точки сразу идти вверх. Оптимизированный маршрут имеет вид:



Соответствующая программа: `вверх, вправо`, что составляет две команды.

Ответ: 2.

(B1, B13) Исполнители

Конспект

Исполнитель — субъект (человек, животное) или устройство, способное выполнить действия, предписываемые алгоритмом. При этом указанные действия выполняются формально. Исполнитель не знает о цели алгоритма, он лишь выполняет все полученные команды.

Каждый исполнитель характеризуется следующими параметрами:

— **среда** — условия, в которых функционирует исполнитель (например, исполнитель Черепашка имеет определенную систему координат, исполнитель Робот перемещается по клетчатому полю и т.д.);

— **система команд** — каждый исполнитель может выполнять команды только из некоторого строго заданного набора, где каждая команда определяет соответствующее элементарное действие.

Алгоритм — запись в той или иной форме (словесной, графической, на языке программирования) последовательности команд для исполнителя. Команды алгоритма должны соответствовать системе команд исполнителя.

В разделе 10 «Программирование» разобраны подробно основные алгоритмические конструкции.

Разбор типовых задач

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

1. **прибавь 1**
2. **умножь на 3**

Выполняя первую из них, Калькулятор прибавляет к числу на экране 1, а выполняя вторую, утраивает его. Запишите порядок команд в программе получения из числа 2 числа 26, содержащей не более 6 команд, указывая лишь номера команд.

(Например, программа 21211 — это программа

умножь на 3
прибавь 1
умножь на 3
прибавь 1
прибавь 1

которая преобразует число 1 в 14.)

Решение

Решать такие задачи проще всего «с конца»: сначала записать число, которое надо получить (в данном случае 26), а затем пытаться от него прийти к исходному числу (2), используя «обратные» команды Исполнителя (в данном случае — **вычесть 1** и **разделить на 3**). При этом, поскольку программа должна содержать мало команд (всего 6), надо прийти к числу 2 как можно быстрее.

Значит, надо по возможности использовать команду деления на 3, а к вычитанию прибегать, если разделить число нацело на 3 не удаётся. Итак:

Команда	Текущее число
	26
вычесть 1	25
вычесть 1	24
разделить на 3	8
вычесть 1	7
вычесть 1	6
разделить на 3	2

Остаётся записать эту программу в обратном порядке (от числа 2 к числу 26), используя изначальные команды Исполнителя — **прибавить 1** и **умножить на 3**:

Команда	Текущее число
	2
умножить на 3	6
прибавить 1	7
прибавить 1	8
умножить на 3	24
прибавить 1	25
прибавить 1	26

Если теперь записать (как требуется в задаче) последовательность *порядковых номеров* этих команд, получается программа: 211211.

Ответ: 211211.

Задача 2*. У исполнителя Утроитель две команды, которым присвоены номера:

1. **прибавь 1**
2. **умножь на 3**

Первая из них увеличивает число на экране на 1, вторая — утраивает его.

Запишите порядок команд в программе преобразования числа 1 в число 22, содержащей не более 5 команд, указывая лишь номера команд. (Например, 21211 — это программа

умножь на 3
прибавь 1
умножь на 3
прибавь 1
прибавь 1

которая преобразует число 1 в 14.)

(Если таких программ существует более одной, то запишите любую из них.)

Решение

Решение задачи аналогично решению задачи 2: сначала надо от числа 22 прийти к исходному числу 1, используя «обратные» команды Исполнителя (**вычесть 1** и **разделить на 3**).

Команда	Текущее число
	22
вычесть 1	21
разделить на 3	7
вычесть 1	6
разделить на 3	2
вычесть 1	1

Затем нужно записать эту программу в обратном порядке (от числа 1 к числу 22), используя изначальные команды Исполнителя — **прибавить 1** и **умножить на 3**:

Команда	Текущее число
	1
прибавить 1	2
умножить на 3	6
прибавить 1	7
умножить на 3	21
прибавить 1	26

Остаётся записать последовательность *порядковых номеров* этих команд. Получается программа: 12121.

Ответ: 12121.

Задача 3*. У исполнителя Кузнечик две команды:

1. прибавь 3

2. вычти 2.

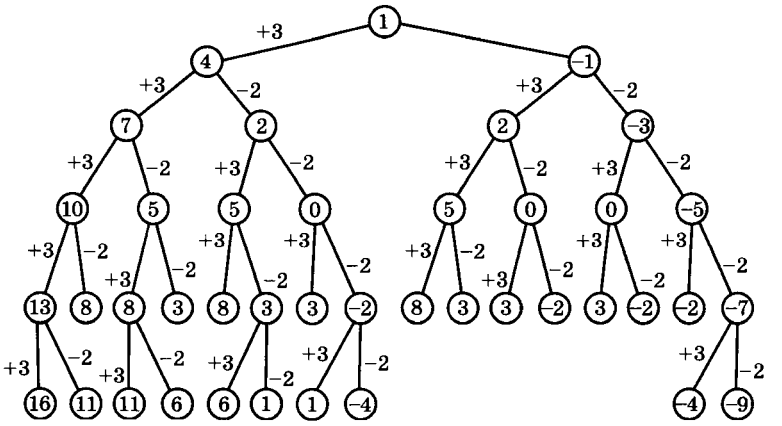
Первая из них увеличивает число на экране на 3, вторая — уменьшает его на 2 (отрицательные числа допускаются).

Программа для Кузнечика — это последовательность команд. Сколько различных чисел можно получить из числа 1 с помощью программы, которая содержит ровно 5 команд?

Решение

В этой задаче нужно определить количество возможных вариантов. Поэтому лучше всего воспользоваться построением графа — дерева, вершинами которого являются числа (причём, корневая вершина — это исходное число). Ветви этого дерева соответствуют возможным каждый раз операциям: их две, поэтому от каждой вершины будет отходить ровно две ветви дерева. А количество таких ветвлений (глубина дерева) будет равно 5 (следует из условия задачи).

Команды



Если дерево получается громоздким, но в его середине встречаются повторяющиеся числа, то их можно исключать уже на этапе построения дерева, «обрубая» в нём лишние ветви и продолжая только одну ветвь из нескольких, соответствующих каждому повторяющемуся числу.

Остаётся только подсчитать количество различных (без учёта повторов!) чисел, соответствующих полученным концевым вершинам. Таких различных чисел — 6.

Ответ: 6 различных чисел.

Этот способ решения является универсальным. Построенное дерево вариантов позволяет узнать также и общее количество чисел (в том числе различных), которые позволяет получить программа для исполнителя, содержащая *не более* указанного количества команд: в этом случае нужно включать в подсчёт *все* вершины построенного дерева (а не только концевые), за исключением корневой.

Задача 5*. Витя пригласил своего друга Сергея в гости, но не сказал ему код от цифрового замка своего подъезда, а послал следующее SMS-сообщение: «в последовательности чисел 3, 1, 8, 2, 6 все числа больше 5 разделить на 2, а затем удалить из полученной последовательности все чётные числа». Выполнив указанные в сообщении действия, Сергей получил следующий код для цифрового замка:

1) 3, 1
2) 1, 1, 3
3) 3, 1, 3
4) 3, 3, 1

Решение

В данной задаче достаточно выполнить указания исполнителю.

Исходная последовательность чисел:

3	1	8	2	6

Все числа больше 5 делим на 2:

3	1	4	2	3

Из полученной последовательности удаляются все чётные числа:

3	1	~~4~~	~~2~~	3

Полученная последовательность:

3	1	3

Ответ: 3, 1, 3 (вариант ответа №3).

Задача 6*. Лена забыла пароль для входа в Windows XP, но помнила алгоритм его получения из символов «A153B42FB4» в строке подсказки. Если последовательность символов «B4» заменить на «B52» и из получившейся строки удалить все трёхзначные числа, то полученная последовательность и будет паролем:

1) ABFB52
2) AB42FB52
3) ABFB4
4) AB52FB

Решение

Исходная последовательность символов:

A153B42FB4

Последовательность символов «B4» меняется на «B52»:

A153**B4**2F**B4** A153**B52**2F**B52**

Из полученной последовательности удаляются все трёхзначные числа:

A~~153B522~~FB52

Полученная последовательность:

ABFB52

Ответ: ABFB52 (вариант ответа №1).

Задача 7*. У исполнителя Утроитель две команды, которым присвоены номера:

1. прибавь 1,

2. умножь на 3.

Первая из них увеличивает число на экране на 1, вторая — утраивает его.

Программа для Утроителя — это последовательность команд.

Сколько есть программ, которые число 1 преобразуют в число 29?

Ответ обоснуйте.

Решение[1]

Для решения можно построить полное дерево вариантов для получения числа 1 из числа 29 (не «обрубая» повторяющиеся ветви!) и подсчитать количество воз-

---

[1] Решение предложено Денисом Королёвым (школа №1360).

можных способов получения этого числа (количество ветвей, равное количеству конечных вершин с учётом всех повторяющихся значений).

Хотя данный способ наиболее понятен и нагляден, для получения максимального балла на экзамене требуется запись аналитического решения задачи.

Пусть $R(n)$ — количество программ, которые преобразуют число 1 в число n.

Для анализа решения лучше (как и в предыдущих задачах) рассматривать обратный процесс — получение числа 1 из числа 29 при помощи обратных команд «вычесть 1» и «делить на 3».

Тогда $R(n) = R(n/3) + R(n - 1)$ {*в общем случае очередное число может быть получено или делением на 3, или вычитанием единицы*}.

Возвращаясь к исходной задаче, расписывается каждый из шагов последовательности действий для получения числа 29 из числа 1.

$R(1) = 1$ {*в любом случае исходное число — «в единственном экземпляре»*}.

$R(2) = R(2/3) + R(2 - 1) = \sout{R(2/3)} + R(2 - 1) = R(2 - 1) = R(1) = 1$ {*слагаемые, дающие нецелый результат, исключаются; для получаемого значения $R(n)$ берётся ранее вычисленное значение*}.

$R(3) = R(3/3) + R(3 - 1) = R(1) + R(2) = 1 + 1 = 2.$

$R(4) = R(4/3) + R(4 - 1) = \sout{R(4/3)} + R(4 - 1) = R(3) = 2.$

$R(5) = R(5/3) + R(5 - 1) = \sout{R(5/3)} + R(5 - 1) = R(4) = 2.$

$R(6) = R(6/3) + R(6 - 1) = R(2) + R(5) = 1 + 2 = 3.$

$R(7) = R(7/3) + R(7 - 1) = \sout{R(7/3)} + R(7 - 1) = R(6) = 3.$

$R(8) = R(8/3) + R(8 - 1) = \sout{R(8/3)} + R(8 - 1) = R(7) = 3.$

$R(9) = R(9/3) + R(9 - 1) = R(3) + R(8) = 2 + 3 = 5.$

$R(10) = R(10/3) + R(10 - 1) = \sout{R(10/3)} + R(10 - 1) = R(9) = 5.$

$R(11) = R(11/3) + R(11 - 1) = \sout{R(11/3)} + R(11 - 1) = R(10) = 5.$

$R(12) = R(12/3) + R(12 - 1) = R(4) + R(11) = 2 + 5 = 7.$

$R(13) = R(13/3) + R(13 - 1) = \sout{R(13/3)} + R(13 - 1) = R(12) = 7.$

$R(14) = R(14/3) + R(14 - 1) = \sout{R(14/3)} + R(14 - 1) =$
$= R(13) = 7.$
$R(15) = R(15/3) + R(15 - 1) = R(5) + R(14) = 2 + 7 = 9.$
$R(16) = R(16/3) + R(16 - 1) = \sout{R(16/3)} + R(16 - 1) =$
$= R(15) = 9.$
$R(17) = R(17/3) + R(17 - 1) = \sout{R(17/3)} + R(17 - 1) =$
$= R(16) = 9.$
$R(18) = R(18/3) + R(18 - 1) = R(6) + R(17) = 3 + 9 = 12.$
$R(19) = R(19/3) + R(19 - 1) = \sout{R(19/3)} + R(19 - 1) =$
$= R(18) = 12.$
$R(20) = R(20/3) + R(20 - 1) = \sout{R(20/3)} + R(20 - 1) =$
$= R(19) = 12.$
$R(21) = R(21/3) + R(21 - 1) = R(7) + R(20) = 3 + 12 = 15.$
$R(22) = R(22/3) + R(22 - 1) = \sout{R(22/3)} + R(22 - 1) =$
$= R(21) = 15.$
$R(23) = R(23/3) + R(23 - 1) = \sout{R(23/3)} + R(23 - 1) =$
$= R(22) = 15.$
$R(24) = R(24/3) + R(24 - 1) = R(8) + R(23) = 3 + 15 = 18.$
$R(25) = R(25/3) + R(25 - 1) = \sout{R(25/3)} + R(25 - 1) =$
$= R(24) = 18.$
$R(26) = R(26/3) + R(26 - 1) = \sout{R(26/3)} + R(26 - 1) =$
$= R(25) = 18.$
$R(27) = R(27/3) + R(27 - 1) = R(9) + R(26) = 5 + 18 = 23.$
$R(28) = R(28/3) + R(28 - 1) = \sout{R(28/3)} + R(28 - 1) =$
$= R(27) = 23.$
$R(29) = R(29/3) + R(29 - 1) = \sout{R(29/3)} + R(29 - 1) =$
$= R(28) = 23.$

Ответ: 23 программы.

Задача 8. У исполнителя «Троитель–шестеритель» две команды, которым присвоены номера:

1. прибавь 6,

2. умножь на 3.

Первая из них увеличивает число на экране на 6, вторая — утраивает его.

Программа для исполнителя — это последовательность команд.

Сколько есть программ, которые число 9 преобразуют в число 87?

Ответ обоснуйте.

Решение

Строится аналогично предыдущей задаче. Поскольку предполагается прибавление не 1, а 6, нужно записывать значения $R(n)$ для n с шагом 6.

$R(9) = 1$ {*в любом случае исходное число — «в единственном экземпляре»*}.

$R(9 + 6) = R(15) = R(15/3) + R(15 - 6) = \sout{R(5)} + R(9) = R(9) = 1$ {*очередную строку записываем для значения n, увеличенного на 6; слагаемые, дающие нецелый результат, исключаются; для получаемого значения R(n) берётся ранее вычисленное значение; если значение R(n) не существует, то оно исключается*}.

Почему нужно расписывать значения $R(n)$ с шагом, равным 6, а не 1?

Если использовать шаг 1, то получим запись:

$R(10) = R(10/3) + R(10 - 6) = R(4)$.

Поскольку вычисления начинаются с числа 9, значение $R(4)$ не существует, и данная запись бессмысленна. Аналогично, несуществующие значения $R(n)$ получаются и в других случаях при попытке записи строк для n с шагом, отличающимся от 6.

$R(21) = R(21/3) + R(21 - 6) = \sout{R(7)} + R(15) = 1$ {*несуществующее значение R(7) исключаем*}.

$R(27) = R(27/3) + R(27 - 6) = R(9) + R(21) = 1 + 1 = 2$.

$R(33) = R(33/3) + R(33 - 6) = \sout{R(11)} + R(27) = 2$ {*несуществующее значение R(11) также исключаем; аналогично поступаем и далее*}.

$R(39) = R(39/3) + R(39 - 6) = \sout{R(13)} + R(33) = 2$.

$R(45) = R(45/3) + R(45 - 6) = R(15) + R(39) = 1 + 2 = 3$.

$R(51) = R(51/3) + R(51 - 6) = \sout{R(17)} + R(45) = 3$.

$R(57) = R(57/3) + R(57 - 6) = \sout{R(19)} + R(51) = 3$.

$R(63) = R(63/3) + R(63 - 6) = R(21) + R(57) = 1 + 3 = 4$.

$R(69) = R(69/3) + R(69 - 6) = \sout{R(23)} + R(63) = 4$.

$R(75) = R(75/3) + R(75 - 6) = \sout{R(25)} + R(69) = 4$.

$R(81) = R(81/3) + R(81 - 6) = R(27) + R(75) = 2 + 4 = 6.$
$R(87) = R(87/3) + R(81 - 6) = \cancel{R(29)} + R(81) = 6.$

Ответ: 6 программ.

Задача 9. У исполнителя Квадратик две команды, которым присвоены номера:

1. прибавь 2,

2. возведи в квадрат.

Первая из них увеличивает число на экране на 2, вторая — возводит в квадрат.

Программа для исполнителя — это последовательность команд.

Сколько есть программ, которые число 4 преобразуют в число 66?

Ответ обоснуйте.

Решение

Записываются строки, содержащие «обратные» операции вычитания пятёрки и вычисления корня квадратного. Кроме того, поскольку в исходной задаче прибавляется число 2, нужно записывать такие строки начиная с исходного числа 4 и с шагом 2.

$R(4) = 1$ {*в любом случае исходное число — «в единственном экземпляре»*}.

$R(4 + 2) = R(6) = R(\sqrt{6}) + R(6 - 2) = \cancel{R(\sqrt{6})} + R(4) =$
$= R(4) = 1$ {*очередную строку записываем для значения n, увеличенного на 2; слагаемые, где квадратный корень извлекается не нацело, исключаются; для получаемого значения R(n) берется ранее вычисленное значение; если значение R(n) не существует, то оно исключается*}.

$R(8) = R(\sqrt{8}) + R(8 - 2) = \cancel{R(\sqrt{8})} + R(6) = R(6) = 1.$

$R(10) = R(\sqrt{10}) + R(10 - 2) = \cancel{R(\sqrt{10})} + R(8) = R(8) = 1.$

$R(12) = R(\sqrt{12}) + R(12 - 2) = \cancel{R(\sqrt{12})} + R(10) = R(10) = 1.$

$R(14) = R(\sqrt{14}) + R(14 - 2) = \cancel{R(\sqrt{14})} + R(12) = R(12) = 1.$

$R(16) = R(\sqrt{16}) + R(16 - 2) = R(4) + R(14) = 1 + 1 = 2.$

$R(18) = R(\sqrt{18}) + R(18-2) = \cancel{R(\sqrt{18})} + R(16) = R(16) = 2.$

$R(20) = R(\sqrt{20}) + R(20-2) = \cancel{R(\sqrt{20})} + R(18) = R(18) = 2.$

$R(22) = R(\sqrt{22}) + R(22-2) = \cancel{R(\sqrt{22})} + R(20) = R(20) = 2.$

$R(24) = R(\sqrt{24}) + R(24-2) = \cancel{R(\sqrt{24})} + R(22) = R(22) = 2.$

$R(26) = R(\sqrt{26}) + R(26-2) = \cancel{R(\sqrt{26})} + R(24) = R(24) = 2.$

$R(28) = R(\sqrt{28}) + R(28-2) = \cancel{R(\sqrt{28})} + R(26) = R(26) = 2.$

$R(30) = R(\sqrt{30}) + R(30-2) = \cancel{R(\sqrt{30})} + R(28) = R(28) = 2.$

$R(32) = R(\sqrt{32}) + R(32-2) = \cancel{R(\sqrt{32})} + R(30) = R(30) = 2.$

$R(34) = R(\sqrt{34}) + R(34-2) = \cancel{R(\sqrt{34})} + R(32) = R(32) = 2.$

$R(36) = R(\sqrt{36}) + R(36-2) = R(6) + R(34) = 1 + 2 = 3.$

$R(38) = R(\sqrt{38}) + R(38-2) = \cancel{R(\sqrt{38})} + R(36) = R(36) = 3.$

$R(40) = R(\sqrt{40}) + R(40-2) = \cancel{R(\sqrt{40})} + R(38) = R(38) = 3.$

$R(42) = R(\sqrt{42}) + R(42-2) = \cancel{R(\sqrt{42})} + R(40) = R(40) = 3.$

$R(44) = R(\sqrt{44}) + R(44-2) = \cancel{R(\sqrt{44})} + R(42) = R(42) = 3.$

$R(46) = R(\sqrt{46}) + R(46-2) = \cancel{R(\sqrt{46})} + R(44) = R(44) = 3.$

$R(48) = R(\sqrt{48}) + R(48-2) = \cancel{R(\sqrt{48})} + R(46) = R(46) = 3.$

$R(50) = R(\sqrt{50}) + R(50-2) = \cancel{R(\sqrt{50})} + R(48) = R(48) = 3.$

$R(52) = R(\sqrt{52}) + R(52-2) = \cancel{R(\sqrt{52})} + R(50) = R(50) = 3.$

$R(54) = R(\sqrt{54}) + R(54-2) = \cancel{R(\sqrt{54})} + R(52) = R(52) = 3.$

$R(56) = R(\sqrt{56}) + R(56-2) = \cancel{R(\sqrt{56})} + R(54) = R(54) = 3.$

$R(58) = R(\sqrt{58}) + R(58-2) = \cancel{R(\sqrt{58})} + R(56) = R(56) = 3.$

$R(60) = R(\sqrt{60}) + R(60-2) = \cancel{R(\sqrt{60})} + R(58) = R(58) = 3.$

$R(62) = R(\sqrt{62}) + R(62-2) = \cancel{R(\sqrt{62})} + R(60) = R(60) = 3.$

$R(64) = R(\sqrt{64}) + R(64-2) = R(8) + R(62) = 1 + 3 = 4.$

$R(66) = R(\sqrt{66}) + R(66-2) = \cancel{R(\sqrt{66})} + R(64) = R(64) = 4.$

Ответ: 4 программы.

Раздел 6. **Архитектура компьютеров и компьютерных сетей**

A4 Файловая система ПК

Конспект

Пример типовой древовидной структуры файловой системы, принятой в ОС MS-DOS и Windows (используемой в задачах ЕГЭ):

Путь к файлу — запись, начинающаяся меткой диска и содержащая имена всех папок, которые нужно одну за другой раскрыть, чтобы кратчайшим способом прийти к файлу.

Полное имя файла — запись пути к файлу, завершаемая именем и расширением этого файла.

В ОС Windows записи пути и полного имени файла метка диска, имена каталогов и имя файла разделяются символом обратной косой черты — «\». В ОС Linux

записи пути и полного имени файла аналогичны, но в качестве символа-разделителя используется символ «/».

Например, для файловой структуры в ОС Windows, изображённой на рисунке выше:

- путь к файлу **Реферат1.doc — С:\Документы\Рефераты** (показан пунктирной стрелкой);
- полное имя файла **Реферат1.doc — С:\Документы\Рефераты\Реферат1.doc.**

Маска (шаблон) имени файла — запись, обозначающая группу файлов, имена которых отвечают заданным в этой маске требованиям. Маска обычно используется в качестве фильтра, чтобы выделить (или отобрать для выборочного показа в списке содержимого папки) файлы с нужными именами (и/или расширениями имени) и отсеять ненужные.

Символы-шаблоны — специальные символы-«джокеры», обозначающие один или несколько любых символов:

- символ «*» (звёздочка) — заменяет собой любое количество любых символов (в том числе нулевое количество — этих символов может не быть вовсе);
- символ «?» (знак вопроса) — заменяет один (и только один) обязательно стоящий в данном месте любой символ.

Маска может содержать как обычные символы (буквы, цифры и прочие знаки, допустимые в именах файлов), так и символы-шаблоны.

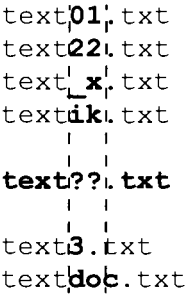
Примеры:

`*.*` — все файлы (т.е. файлы с любым именем и любым расширением);

`*.doc` — все файлы с любыми именами и расширением `doc`;

`text??.txt` — все файлы, имена которых начинаются с букв `text` и завершаются обязательно имеющимися двумя любыми символами, а расширение которых — `txt` (например, это могут быть файлы

text01.txt, text22.txt, text_x.txt, textik.txt, но не text3.txt или textdoc.txt).



Важное различие!

Символ «*» обозначает любое количество любых символов, в том числе нулевое (т.е. когда символов нет вообще).

Символ «?» обозначает один, и только один любой символ; несколько символов «?» подряд обозначают ровно такое же количество любых символов (например, ??? — ровно три любых символа, не больше и не меньше).

Чтобы задать количество любых символов, *не меньшее* заданного, нужно использовать оба указанных символа-шаблона, когда символы «?» задают минимально допустимое число символов, а последующий символ «*» указывает, что символов может быть и больше. Например, маска ???* означает запись, содержащую не менее трёх любых символов (три обязательных — ??? и любое количество, в том числе нулевое, необязательных — *).

Разбор типовых задач

Задача 1*. 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 имен файлов удовлетворяет маске:

?ba*r.?xt

1) bar.txt
2) obar.txt
3) obar.xt
4) barr.txt

Решение

Запись ?ba*r.?xt означает, что ищутся файлы, в имени которых:

- пара символов «ba» обязательно записаны на втором и третьем месте имени файла, а перед ними обязательно стоит один любой символ — в маске он закодирован знаком «?»;
- после символов «ba» может идти любое количество символов (знак «*»), но имя обязательно завершается буквой «r»;
- расширение имени всегда состоит из трёх символов, из которых два последние — «xt».

Анализируя приведённые в качестве вариантов ответа имена файлов на соответствие этим требованиям получается:

1) bar.txt — здесь перед символами «ba» отсутствует символ (который закодирован знаком «?») — данный вариант не подходит;

2) obar.txt — перед символами «ba» имеется символ «o», имя завершается символом «r» (знак «*» может означать и отсутствие символов!), расширение имени состоит из трёх букв и завершается парой символов «xt» — данный вариант ответа годится;

3) obar.xt — хотя структура имени соответствует заданной маске (см. выше), расширение имени здесь двузначно, т.е. данное имя файла не соответствует маске;

4) barr.txt — перед символами «ba» отсутствует символ (который закодирован знаком «?») — данный вариант не подходит.

Таким образом, указанной маске соответствует только имя файла obar.txt.

Ответ: obar.txt (вариант ответа №2).

Задача 2*. Для групповых операций с файлами используются маски имен 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, по какой из масок будет выбрана указанная группа файлов:

```
1234.xls
23.xml
234.xls
23.xml
```

1) `*23*.?x*`
2) `?23?.x??`
3) `?23?.x*`
4) `*23*.???`

Решение

Принцип решения данной задачи состоит в поочередной проверке каждой из предложенных масок (в вариантах ответа) на соответствие указанным именам файлов.

1. Маска `*23*.?x*`. Предполагает, что имя файла обязательно содержит цифры `23`, до и после которых может быть любое количество других символов (но их может и не быть!). В расширении же имени файла обязательно имеется символ «x», перед которым обязательно есть какой-то символ, а после него может (но необязательно) быть любое число символов.

Этой маске не соответствует ни один из заданных файлов, так как в расширениях их имён символ «x» стоит первым, а не вторым. Следовательно, данная маска не является решением задачи.

2. Маска `?23?.x??`. Предполагает, что в имени файла перед и после цифр `23` обязательно есть по одному

какому-то символу (знаки «?» в маске), а в расширении имени символ «x» обязательно стоит самым первым и после него обязательно есть ещё два каких-то символа.

Этой маске не соответствуют имена файлов `23.xml` и `234.xls`, так как в них не обеспечено наличие по одному символу до и после цифр `23`. Следовательно, данная маска также не является решением задачи.

3. Маска `?23?.x*`. Предполагает, что в имени файла перед и после цифр `23` обязательно есть по одному какому-то символу (знаки «?» в маске), а в расширении имени символ «x» обязательно стоит самым первым и после него могут (но не обязательно) стоять какие-то другие символы.

Этой маске (как и предыдущей) не соответствуют имена файлов `23.xml` и `234.xls`, так как в них не обеспечено наличие по одному символу до и после цифр `23`. Следовательно, данная маска тоже не является решением задачи.

4. Маска `*23*.???`. Предполагает, что имя файла обязательно содержит цифры `23`, до и после которых может быть любое количество других символов (но их может и не быть!). В расширении имени обязательно должно быть три любых символа (не больше и не меньше).

Этой маске полностью соответствуют все заданные файлы. Следовательно, данная маска является решением задачи.

Ответ: маска `*23*.???` (вариант ответа №4).

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 которой 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.

В каталоге находятся пять файлов:

```
fort.docx
ford.docx
lord.doc
orsk.dat
port.doc
```

Определите, по какой из масок из них будет отобрана указанная группа файлов:

```
fort.docx
ford.docx
lord.doc
port.doc
```

1) `*o?*.d?*`
2) `?o*?.d*`
3) `*or*.doc?`
4) `?or?.doc?`

Решение

В данной задаче, анализируя каждую из предложенных масок, предполагается проверять не только то, что она позволяет выбрать все указанные требуемые файлы, но и что она не приводит к ошибочному выбору каких-либо других файлов из заданного полного их списка.

1. Маска `*o?*.d?*`. Предполагает, что имя файла обязательно содержит хотя бы две буквы, из которых первая — буква `o` (знак «`?`» после «`o`» указывает наличие ещё одного обязательного символа, а до и после них может стоять любое число любых символов, но их может и не быть). В расширении имени файла обязательно первой стоит буква `d`, а после неё должен быть минимум один любой символ (знак «`?`»).

Этой маске соответствуют все имена файлов, которые должны быть отобраны по ней. Однако ей соответствует также имя файла `orsk.dat`, которое не должно быть отобрано. Следовательно, данная маска не является решением задачи.

2. Маска `?o*?.d*`. Предполагает, что в имени файла буква «`o`» обязательно вторая по счёту (перед ней стоит знак «`?`») и что после неё есть хотя бы один любой

символ (комбинация знаков «*?»). Расширение имени файла обязательно должно начинаться с буквы «d».

Этой маске соответствуют все имена файлов, которые должны быть отобраны по ней. Имя же файла orsk.dat не соответствует данной маске (в нём буква «o» стоит на первом месте, перед ней нет обязательного другого символа, отмеченного в маске знаком «?»), следовательно, данная маска является решением задачи.

3. Маска *or*.doc?. Предполагает, что в имени файла обязательно встречается пара букв «or», а расширение имени начинается с «doc» и всегда завершается любым четвёртым по счёту символом.

Этой маске не соответствуют имена файлов lord.doc и port.doc, которые должны быть отобраны по ней (в них расширение содержит только три символа). Следовательно, данная маска не является решением задачи.

4. Маска ?or?.doc?. Предполагает, что имя файла может содержать только 4 символа, из которых средние — «or», а перед и после них стоит по одному любому символу («обрамляющие» знаки «?»), а расширение имени, как и в предыдущем случае, начинается с «doc» и всегда завершается любым четвёртым по счёту символом.

Этой маске также не соответствуют имена файлов lord.doc и port.doc, которые должны быть отобраны по ней (в них расширение содержит только три символа). Следовательно, данная маска не является решением задачи.

Значит, решением может быть только маска ?o*?.d*.

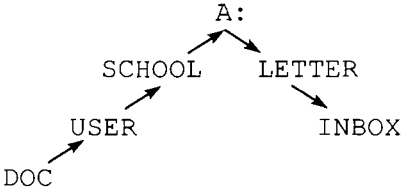
Ответ: маска ?o*?.d* (вариант ответа №2).

Задача 4*. Перемещаясь из одного каталога в другой, пользователь последовательно посетил каталоги DOC, USER, SCHOOL, A:\, LETTER, INBOX. При каждом перемещении пользователь либо спускался в каталог на уровень ниже, либо поднимался на уровень выше. Каково полное имя каталога, из которого начал перемещение пользователь?

1) A:\DOC
2) A:\LETTER\INBOX
3) A:\SCHOOL\USER\DOC
4) A:\DOC\USER\SCHOOL

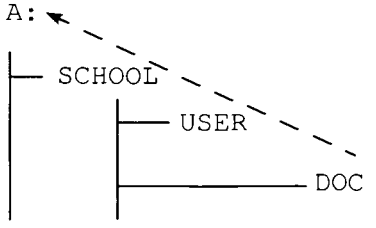
Решение

В середине пройденной цепочки имён каталогов располагается метка диска A:. Это — наивысший уровень иерархии в файловой системе. Следовательно, по пути к этой метке диска пользователь поднимался вверх уровень за уровнем (так как в приведённой записи слева от A: имена каталогов не повторяются, этот подъём не прерывался спуском), а после прохождения этой метки диска — спускался вниз уровень за уровнем:



Левая ветвь такого рисунка строится от метки A: в записи последовательности пройденных каталогов сверху вниз в обратном порядке, а правая — от метки A: сверху вниз в прямом порядке.

Поскольку для решения нужно полное имя исходного каталога, из полученного графического изображения пути рассматривается только левая «ветвь». Для неё можно изобразить следующую файловую структуру:



Теперь уже нетрудно записать полное имя исходного каталога: A:\SCHOOL\USER\DOC

Ответ: A:\SCHOOL\USER\DOC (вариант ответа №3).

B11 Основные принципы функционирования сети Интернет. Протокол TCP/IP

Конспект

IP-адрес — уникальный (не повторяющийся) цифровой индивидуальный номер компьютера в сети. IP-адрес может быть закреплён за данным компьютером постоянно (**выделенный IP-адрес**) или выдаваться компьютеру временно из некоторого имеющегося набора только на время данного сеанса работы в сети.

В стандарте **IP v.4** (рассматриваемом в задачах ЕГЭ) IP-адрес представляет собой запись из четырёх разделённых точками чисел, каждое из которых соответствует одному байту и имеет значение от 0 до 255. Пример:

168.154.0.177

Некоторые IP-адреса являются служебными и используются для специальных целей:

- адрес сети, в котором один или несколько крайних справа байтов равны нулю (примеры: **168.154.15.0**, **168.154.0.0**);
- широковещательный адрес, в котором один или несколько крайних справа байтов равны 255 (примеры: **168.154.15.255**, **168.154.255.255**).

Адрес сети, адрес компьютера в сети, маска IP-адреса

Если компьютер находится в составе локальной сети, которая, в свою очередь, подключена к сети Интернет, то возникает задача — определить IP-адрес всей локальной сети в целом (например, чтобы разослать пакет информации на *все* компьютеры данной локальной сети) и адрес конкретного компьютера внутри этой локальной сети.

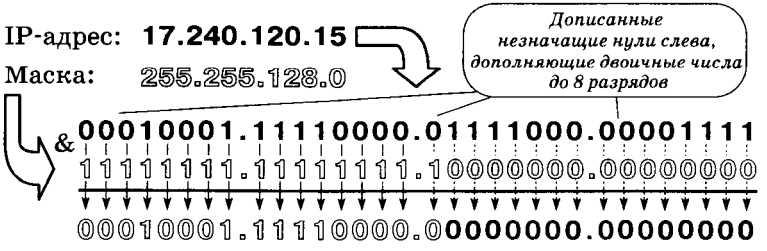
Адрес сети и адрес компьютера в этой сети — это две условно выделяемые части единого IP-адреса данного компьютера, под которым он числится в Интернете:

адрес сети *адрес компьютера в этой сети*

Для разделения единого IP-адреса на адрес локальной сети и адрес компьютера в этой сети используется **маска IP-адреса**. Она имеет такой же вид, как IP-адрес (четыре числа-байта, записанных через точку) и выполняет функцию фильтра.

1. Определение адреса сети по полному IP-адресу и маске:

- исходный IP-адрес записывается в двоичной форме (каждое число — байт переводится в восьмиразрядное двоичное число отдельно и по-прежнему записывается через точку);
- маска также записывается в двоичной форме;
- двоичная запись маски записывается под двоичной записью IP-адреса;
- для определения адреса сети нужно выполнить логическую операцию И для каждой отдельной пары битов: если в маске в данном разряде стоит 1, то в качестве результата в данном разряде переписывается значение (0 или 1) из одноимённого разряда исходного IP-адреса; если в маске в данном разряде стоит 0, то в качестве результата в данном разряде тоже записывается 0;
- полученная двоичная запись преобразуется в десятичную форму (каждое двоичное число — отдельно). Пример:



Ответ: **17.240.0.0**

При переводе десятичных значений чисел — составляющих IP-адреса в двоичный формат нужно помнить: получаемые двоичные числа обязательно должны быть восьмиразрядными! При необходимости надо дополнять полученное двоичное число до 8 разрядов, дописывая к нему слева незначащие нули.

2. Определение адреса компьютера в сети (номера компьютера) по полному IP-адресу и маске:

- исходный IP-адрес записывается в двоичной форме (каждое число — байт переводится в восьмиразрядное двоичное число отдельно и по-прежнему записывается через точку);
- маска также записывается в двоичной форме;
- маска инвертируется (единичные биты заменяются на нулевые и наоборот);
- двоичная запись инвертированной маски записывается под двоичной записью IP-адреса;
- для определения адреса сети нужно выполнить логическую операцию И для каждой отдельной пары битов: если в маске в данном разряде стоит 1, то в качестве результата в данном разряде переписывается значение (0 или 1) из одноимённого разряда исходного IP-адреса; если в маске в данном разряде стоит 0, то в качестве результата в данном разряде тоже записывается 0;
- полученная двоичная запись преобразуется в десятичную форму (каждое двоичное число — отдельно).

Пример:

IP-адрес: 17.240.120.15 ⇒
00010001.11110000.01111000.00001111
Маска: 255.255.128.0 ⇒
11111111.11111111.10000000.00000000
Инвертированная маска:
00000000.00000000.01111111.11111111

00010001.11110000.01111000.00001111
& 00000000.00000000.01111111.11111111
00000000.00000000.01111000.00001111

Результат: 00000000.00000000.01111000.00001111 ⇒
⇒0.0.120.15

3. Определение количества компьютеров (количества адресов) в сети по маске:

- маска записывается в двоичной форме (каждое число — байт переводится в восьмиразрядное двоичное число отдельно и по-прежнему записывается через точку);
- маска инвертируется (единичные биты заменяются на нулевые и наоборот);
- разделяющие точки отбрасываются (т.е. инвертированная маска рассматривается как единое 32-битовое двоичное число);
- в этом числе отбрасываются незначащие нули слева;
- полученное двоичное число переводится в десятичную систему счисления;
- полученное число увеличивается на 1 (чтобы учесть, что нумерация адресов начинается с нулей).

 В задаче может спрашиваться: какое количество адресов компьютеров в сети допускается *теоретически*. Тогда полученное вышеописанным способом значение — и есть ответ.

Если в задаче спрашивается: какое в сети возможно количество различных *реальных* (*допустимых*) адресов компьютеров, то необходимо учесть, что два адреса компьютера (состоящий из всех нулевых битов и состоящий из всех единичных битов) являются специальными, поэтому количество допустимых адресов компьютеров на 2 меньше рассчитанного выше.

Пример:

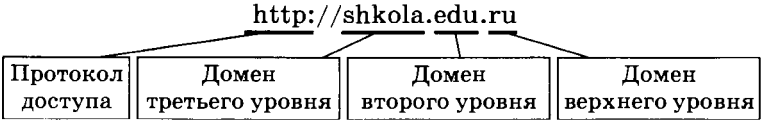
Маска: 255.255.128.0 ⇒
11111111.11111111.10000000.00000000
Инвертированная маска:
00000000.00000000.01111111.11111111

Получаемое из инвертированной маски число: $111111111111111 = 32767_{10}$.

Теоретически возможно адресов компьютеров в сети: 32767 + 1 = 32768.

Реально допускается адресов компьютеров в сети: 32768 – 2 = 32766.

Структура словесного адреса интернет-ресурса (URL)



Разбор типовых задач

Задача 1*. 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.

.64	2.16	16	8.132
А	Б	В	Г

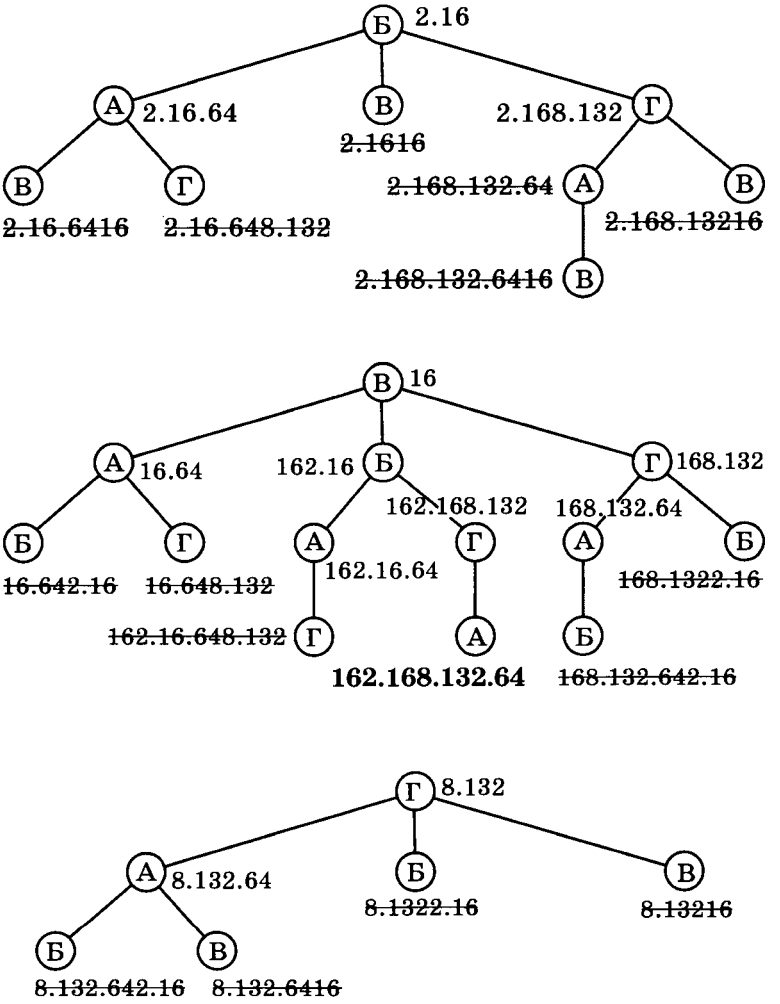
Решение

При решении таких задач нужно помнить следующее:

- ни одно число в записи IP-адреса не должно быть больше 255;
- IP-адрес состоит из четырёх чисел, разделённых точками.

С учётом этих ограничений нужно выполнить перебор всех возможных комбинаций предложенных фрагментов. В общем случае для этого может потребоваться построить 4 дерева вариантов, каждое из которых начинается с одного из этих фрагментов. Однако реально их количество будет меньше, так как с некоторых фрагментов IP-адрес заведомо начинаться не может (в данной задаче это фрагмент А: IP-адрес не может начинаться с точки). Если при построении такого дерева в какой-

то ветви получается недопустимая запись IP-адреса, то дальнейшее построение этой ветви прекращается (ниже такие недопустимые IP-адреса показаны зачёркнутыми).



Таким образом, единственный допустимый вариант — это IP-адрес **162.168.132.64** (ВБГА).

Ответ: ВБГА.

Задача 2*. По заданному IP-адресу и маске определить адрес сети.

IP-адрес: 17.240.120.15
Маска: 255.255.128.0

Решение

Маской IP-адреса называют число (так же, как и IP-адрес, состоящее из четырёх записанных через точку байтовых значений от 0 до 255 каждое), которое определяет, какую часть IP-адреса составляет адрес сети. Для этого нужно представить и исходный IP-адрес, и маску как набор двоичных чисел, а затем поразрядно выполнить их конъюнкцию: если в каком-либо разряде маски записана «1», то значение соответствующего разряда исходного IP-адреса переписывается без изменения, а если в маске записан «0», то соответствующий разряд IP-адреса обнуляется (можно сказать, что «0» в маске «гасит» до 0 соответствующий разряд исходного IP-адреса). Полученная в результате запись и будет указывать IP-адрес сети (каждое из её двоичных чисел можно вновь перевести в десятичную систему счисления).

IP-адрес:
17.240.120.15 → 00010001.11110000.01111000.00001111
&
Маска:
255.255.128.0 → 11111111.11111111.10000000.00000000
↓
00010001.11110000.00000000.00000000
↓
17.240.0.0

Ответ: 17.240.0.0

Задача 3. В терминологии сетей TCP/IP маской подсети называется 32-разрядное двоичное число, определяющее, какие именно разряды IP-адреса компьютера являются общими для всей подсети, — в этих разрядах маски стоит 1. Обычно маски записываются в виде четвёрки десятичных чисел — по тем же правилам, что и IP-адреса. Для некоторой подсети используется маска 255.255.254.0. Сколько различных адресов компьютеров допускает эта маска?

Примечание. На практике для адресации компьютеров не используются два адреса: адрес сети и широковещательный адрес.

Решение

Маска подсети позволяет выделять из всего пространства IP-адресов адрес подсети и адреса отдельных компьютеров в этой подсети следующим образом.

Если дан некий конкретный IP-адрес и дана маска подсети, то нужно, преобразовав их десятичную запись (в виде четырёх чисел от 0 до 255, записанных через точку) в двоичную (четыре 8-разрядных числа, также записанных через точку), записать IP-адрес, под ним — маску, а затем выполнить для них поразрядно следующие операции:

- для получения адреса подсети — для каждого разряда маски, равного 1, просто переписать значение соответствующего ему разряда исходного IP-адреса, а для каждого разряда маски, равного 0, записать нулевой разряд в получаемой записи (т.е. разряды исходного IP-адреса «проходят» неизменными «сквозь» единичные биты маски, но гасятся в 0 нулевыми битами маски);
- для получения адреса компьютера в пределах этой подсети надо вначале инвертировать маску (заменить к ней 0 на 1, а 1 на 0), а затем выполнить те же самые действия с исходным IP-адресом и инвертированной маской.

Полученную двоичную запись адреса подсети (в первом случае) или адреса компьютера в пределах подсети (во втором случае) затем надо снова перевести в набор записанных через точку четырёх десятичных чисел.

В рассматриваемой задаче вопрос поставлен несколько иначе. Здесь по заданной маске нужно определить, какое количество различных адресов компьютеров данная маска образует в пределах подсети. Для этого:

1) маска записывается в двоичном виде:

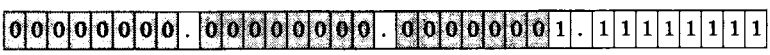
255.255.254.0 $\Rightarrow$

11111111.11111111.11111110.00000000

2) полученная запись маски инвертируется:

00000000.00000000.00000001.11111111

3) данную запись маски можно представить в следующем виде:



Серым фоном выделена та часть IP-адресов компьютеров подсети, которая не может изменяться (ведь, по условию, все эти компьютеры должны располагаться в одной и той же подсети!), а белыми оставлены те биты IP-адресов, которые могут меняться;

4) количество теоретически возможных адресов компьютеров можно определить, зная, что в них биты, соответствующие «белым» битам маски, могут формировать все возможные комбинации нулей и единиц — от «0.00000000» до «1.11111111». Все возможные такие комбинации — это не что иное, как все возможные 9-разрядные двоичные числа от 0 до 111111111. Таких чисел будет

$$1000000000_2 = 2^9 = 512;$$

5) согласно примечанию к условию задачи, из этого теоретического множества в качестве IP-адресов отдельных компьютеров нельзя использовать два адреса:

- адрес сети, в котором все указанные *изменяемые* биты равны нулю;
- широковещательный адрес, в котором все указанные *изменяемые* биты равны единице (отправка пакета информации на такой адрес приводит к рассылке пакета *всем* компьютерам данной подсети, поэтому он и называется «широковещательным»).

Для решения данной задачи не так важны эти объяснения, как информация о том, что из 512 найденных IP-адресов для отдельных компьютеров подсети можно использовать на 2 адреса меньше, т.е. всего 510 адресов.

Ответ: 510 адресов.

Задача 4*. В терминологии сетей TCP/IP маской подсети называется 32-разрядное двоичное число, определяющее, какие именно разряды IP-адреса компьютера являются общими для всей подсети — в этих разрядах маски стоит 1. Обычно маски записываются в виде четвёрки десятичных чисел — по тем же правилам, что и IP-адреса.

Для некоторой подсети используется маска 255.255.252.0. Сколько различных адресов компьютеров теоретически допускает эта маска?

Примечание. На практике используются не все из этих адресов. Например, как правило, не используются IP-адреса, в десятичном представлении которых последнее (самое правое) число равно 0.

Решение

По двоичной записи маски:

11111111.11111111.11111100.00000000

можно определить, что теоретически возможных IP-адресов компьютеров в пределах определяемой этой маской подсети будет $10000000000_2 = 2^{10} = 1024$.

В примечании к задаче сказано, что на практике не используются адреса, в десятичном представлении которых последнее число равно 0. Очевидно, что таких адресов будет четыре (удобно записывать только последние, «изменяемые» биты):

00.00000000
01.00000000
10.00000000
11.00000000

По аналогии с предыдущей задачей «хочется» уменьшить найденное теоретическое количество адресов на 4 и записать в ответе число 1020. Однако, само это примечание добавлено, чтобы запутать! В вопросе к задаче спрашивается именно *теоретически* возможное количество адресов, а не используемое на практике! Так что правильный ответ — именно 1024.

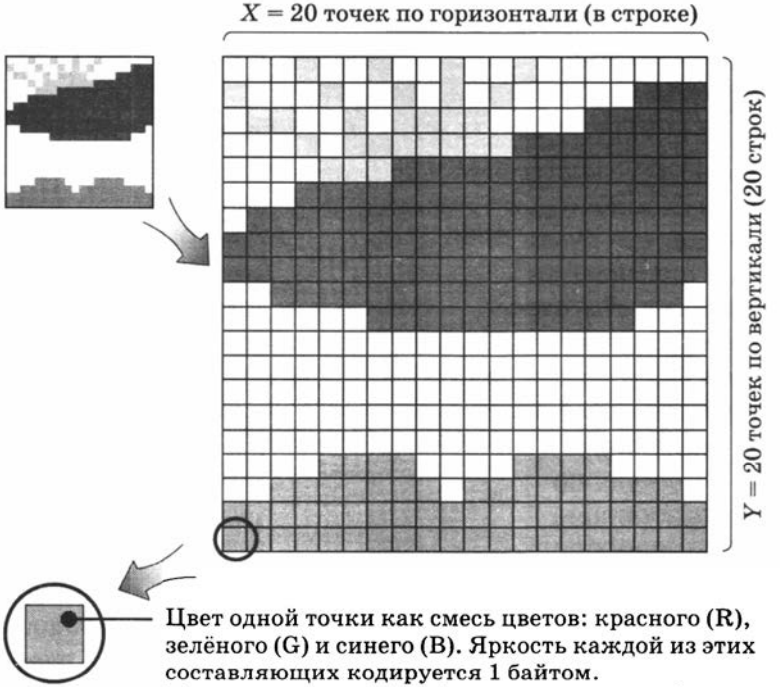
Ответ: 1024 адреса.

Раздел 7. Технология обработки звуковой и графической информации

A8 Определение объёма и скорости передачи цифровой мультимедиа-информации

Конспект

Принципы цифрового кодирования растрового изображения

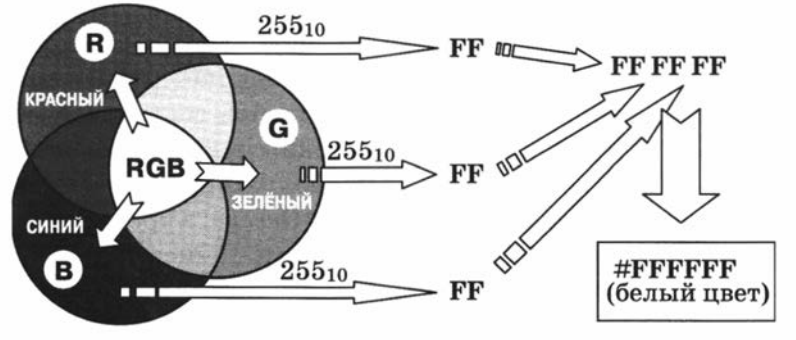


Цвет одной точки как смесь цветов: красного (R), зелёного (G) и синего (B). Яркость каждой из этих составляющих кодируется 1 байтом.
Для монохромного изображения цвет одной точки кодируется 1 битом.

Общий объём информации, байт = (цвет одной точки) × (кол-во точек в строке) × (кол-во строк) = 3 байта × 20 × 20 = 1200 байт.

Принципы кодирования цветовых оттенков

В цветовой системе **RGB** каждый пиксель представляет собой «смесь» трёх точек красного (**R** — ***Red***), зелёного (**G** — ***Green***) и синего (**B** — ***Blue***) цветов. Эти цвета называют «основными» для этой цветовой системы. Каждый из трёх основных цветов может иметь различную яркость, которая кодируется двоичным числом.



В рассматриваемой в задачах ЕГЭ **24-битной RGB-модели** каждый из трёх основных цветов кодируется двоичным числом, разряды которого нумеруются от 0 до 7. То есть речь идёт о *8-битном кодировании цветов в системе RGB*.

Red:

7	6	5	4	3	2	1	0

8 бит $\Rightarrow 2^8 = 256$ градаций

Green:

7	6	5	4	3	2	1	0

8 бит $\Rightarrow 2^8 = 256$ градаций

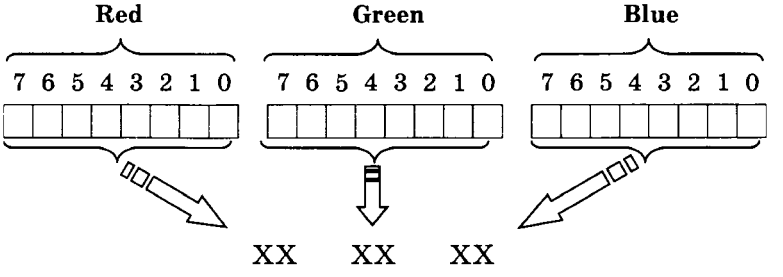
Blue:

7	6	5	4	3	2	1	0

8 бит $\Rightarrow 2^8 = 256$ градаций

Данный способ кодирования цветовых оттенков соответствует широко используемому в современных ПЭВМ 24-битному методу цветового кодирования *TrueColor*. Нетрудно подсчитать, что данный метод обеспечивает $256 \times 256 \times 256 = 2^{8+8+8} = 16\,777\,216$ различных цветовых оттенков.

При использовании 24-битной RGB-модели TrueColor для кодирования цветовых оттенков на web-страницах обычно используется шестнадцатиразрядная запись, в которой подряд записывается шесть шестнадцатиразрядных цифр. Первые две цифры соответствуют шестнадцатеричной записи значения яркости **красного** цвета (***Red***), вторые две — шестнадцатеричной записи значения яркости **зелёного** цвета (***Green***), а третья пара цифр определяет шестнадцатеричную запись значения яркости **синего** цвета (***Blue***):



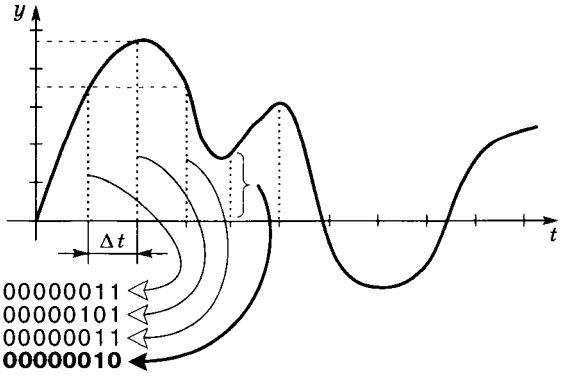
где стрелки обозначают перевод из двоичной в шестнадцатеричную систему счисления: исходное двоичное число (дополненное, если требуется, до 8 разрядов незначащими нулями слева) делится на две половины по 4 бита в каждой, а затем каждая такая «тетрада» битов заменяется одной шестнадцатеричной цифрой по следующей таблице:

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Цвет	Составляющие			Шестнадцатеричный код цвета	Словесное обозначение
	R	G	B		
Белый	FF	FF	FF	#FFFFFF	white
Красный	FF	00	00	#FF0000	red
Зелёный	00	FF	00	#00FF00	lime
Синий	00	00	FF	#0000FF	blue
Тёмно-красный	80	00	00	#800000	darkred
Тёмно-зелёный	00	80	00	#008000	green
Тёмно-синий	00	00	80	#000080	darkblue
Жёлтый	FF	FF	00	#FFFF00	yellow
Голубой	00	FF	FF	#00FFFF	cyan
Фиолетовый	FF	00	FF	#FF00FF	magenta
Чёрный	00	00	00	#000000	black
Серый	80	80	80	#808080	gray

Принципы цифрового кодирования аналогового сигнала (на примере записи звука)

Измерение амплитуды (величины напряжения) через равные малые промежутки времени Δt (с определённой частотой). Получаемые округлённые числовые значения в двоичной форме последовательно записываются в файл.



Частота дискретизации в Герцах (Гц) означает, что измерение электрического сигнала (громкости звука) осуществляется указанное количество раз в секунду, т.е. в файл каждую секунду записывается данное количество двоичных чисел. **Разрешение** в битах определяет разрядность каждого записываемого в файл числа. Если записывается **стереозвук** (т.е. ведётся двухканальная запись), то оцифровке подвергается не один электрический сигнал, а сразу два (с левого и правого микрофона) и, соответственно, удваивается количество сохраняемой цифровой информации; для **монофонической** записи умножение на 2 не требуется.

Общий объём информации, бит = (частота дискретизации, Гц) × (разрешение) × (длительность записи, с) × 2 (для стерео).

Например, при частоте дискретизации 48 кГц (= 48000 Гц), разрешении 24 бита, длительности записи 1 мин (= 60 с) и стереозвуке получается, что:

Общий объём информации, бит = 48 000 × 24 (бит) × 60 (с) × 2 =

Частота дискретизации = кол-во замеров в секунду

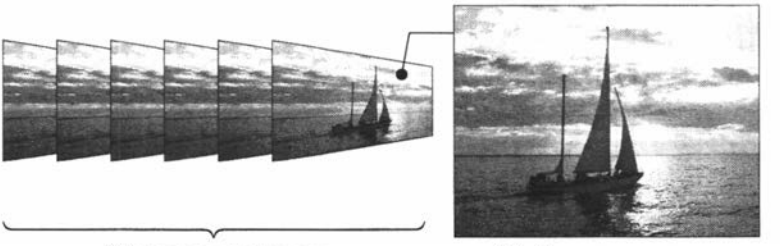
Разрешение = разрядность одного отсчёта

Длительность записи

Кол-во записываемых каналов

= 138240000 бит =
= 17280000 байт =
= 16875 кбайт.

Принципы цифрового кодирования видеозаписи



24 кадра в секунду

Объём одного кадра вычисляется аналогично растровому рисунку

Общий объём информации, байт = (объём одного кадра) × 24 × (длительность, с) =
= (цвет одной точки) × (кол-во точек в строке) ×
× (кол-во строк) × 24 × (длительность, с).

Например, для видеофильма длительностью 2 мин (=120 с) при разрешении 640×480 получается, что:

Общий объём информации = 3 байта × 640 × 480 × × 24 × 120 с = 2654208000 байт = 2592000 кбайт = = 2531,25 Мбайт.

Всё сказанное выше относится к так называемым «несжатым» форматам.

Существуют также различные форматы хранения графической, аудио- и видеоинформации со сжатием, позволяющие (иной раз существенно) уменьшить объём записываемой информации, однако в ЕГЭ обычно рассматриваются именно «чистые» форматы без сжатия.

Разбор типовых задач

Задача 1*. Для хранения растрового изображения размером 128 × 128 пикселей отвели 4 килобайта памяти. Каково максимально возможное число цветов в палитре изображения?

1) 8
2) 2
3) 16
4) 4

Решение

Общее кол-во информации = 4 кб = $4 \cdot 2^{10}$ байт.

Количество пикселей в изображении: 128 × 128 = = 16384.

Объём информации на 1 пиксель =
= $4 \cdot 2^{10}$ байт / 16384 = 4096 байт / 16384 =
= 32768 бит / 16384 = 2 бита.

Два бита (двухразрядное двоичное число) может кодировать 2^2 = 4 цвета.

Ответ: 4 цвета.

Задача 2*. Сколько секунд потребуется модему, передающему сообщения со скоростью 28800 бит/с, чтобы передать цветное растровое изображение размером 640×480 пикселей, при условии, что цвет каждого пикселя кодируется тремя байтами?

Решение

Объём информации, соответствующей изображению = = 3 байта × 640 × 480 = 921600 байт.

Длительность передачи информации =
= 921600 байт / 28800 бит/с =
= 7372800 бит / 28800 бит/с = 256 с.

Ответ: 256 секунд.

Следует не забывать приводить все величины к одной размерности (килобайты к байтам, байты к битам, минуты к секундам и т.д.).

Задача 3*. Укажите минимальный объём памяти (в килобайтах), достаточный для хранения любого растрового изображения размером 64 × 64 пикселя, если известно, что в изображении используется палитра из 256 цветов. Саму палитру хранить не нужно.

1) 128
2) 2
3) 256
4) 4

Решение

Палитра из 256 цветов может быть закодирована восьмиразрядным двоичным числом ($2^8 = 256$), т.е. каждый пиксель соответствует 1 байту.

Объём информации, соответствующей изображению = 1 байт × 64 × 64 = 4096 байтов = 4 кбайта.

Ответ: 4 кбайта.

Задача 4*. Производится одноканальная (моно) звукозапись с частотой дискретизации 16 кГц и 24-битным разрешением. Запись длится 1 минуту, её результаты записываются в файл, сжатие данных не производится.

Какая из приведённых ниже величин наиболее близка к размеру полученного файла?

1) 0.2 Мбайт
2) 2 Мбайт
3) 3 Мбайт
4) 4 Мбайт

Решение

Частота дискретизации = 16 кГц = 16 000 Гц.

Разрешение = 24 бита.

Длительность записи = 1 мин = 60 с.

Одноканальная запись.

Общий объём записываемой информации =
= 16 000 Гц × 24 бит × 60 с × 1 (бит).

Расчёты удобнее всего производить, по возможности переведя все сомножители в степени двойки (поскольку затем потребуется выполнять перевод в байты, килобайты, мегабайты и пр.). Поэтому формула получит вид:

$16\,000 \times 24$ (бит) $\times 60$ (с) $\times 1 =$
$= 125 \cdot 2^7 \times 3 \cdot 2^3 \times 15 \cdot 2^2 = 5625 \cdot 2^{(7+3+2)} =$
$= 5625 \cdot 2^{12}$ (бит).

По условию задачи нужно сравнить полученный результат с различными вариантами ответа, приведёнными в мегабайтах. Поэтому результат вычислений требуется разделить сначала на 2^3, чтобы перевести биты в байты, а затем — на 2^{20} для перевода байтов в мегабайты:

$5625 \cdot 2^{12}$ (бит) $= 5625 \cdot 2^9$ (байт) $= 5625 / 2^{11}$ (Мб) =
$= 5625 / 2048$ (Мб).

Вычисления достаточно выполнить приближённо, так как не требуется указывать точный ответ, а достаточно указать, к какому из имеющихся значений он ближе всего. Поэтому деление на 2048 можно для упрощения вычислений заменить делением на 2000. Тогда объём получаемого аудиофайла примерно равен 2,8 Мб.

Ответ: вариант ответа №3.

Задача 5*. Для кодирования цвета фона web-страницы используется атрибут `bgcolor="#XXXXXX"`, где в кавычках задаются шестнадцатеричные значения интенсивности цветовых компонент в 24-битной RGB-модели. Какой цвет будет у страницы, заданной тегом `<body bgcolor="#00FF00">`?

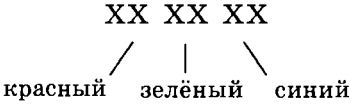
1) белый
2) зелёный
3) красный
4) синий

Решение

В данном случае яркости красного и синего основных цветов — минимально возможные (значение 00, т.е. нуль), а яркость зелёного цвета — максимально возможная ($FF_{16} = 11111111_2 = 255_{10}$). Очевидно, что это — чистый зелёный цвет:

Ответ: зелёный цвет (вариант ответа №2).

Задача 6*. Для кодирования цвета фона интернет-страницы используется атрибут `bgcolor="#XXXXXX"`, где в кавычках задаются шестнадцатеричные значения интенсивности цветовых компонент в 24-битной RGB-модели следующим образом:



К какому цвету будет близок цвет страницы, заданный тегом `<body bgcolor="#747474">`?

1) серый
2) белый
3) фиолетовый
4) чёрный

Решение

В условии задачи значения основных цветов даны произвольные (из допустимого интервала от 00 до FF). Требуется сформулировать экспертную оценку — понять, к какому из названных цветов ближе всего оттенок, заданный таким кодом. Принцип решения таких экспертных задач аналогичен предыдущему примеру.

Все три значения яркостей для всех трёх основных цветов абсолютно одинаковы. Это означает, что какими бы ни были эти значения, получаемый оттенок будет *монохромным* — это будет серый цвет некоторой яркости.

Если все три значения яркостей основных цветов равны 00, то получается «самый тёмный серый цвет» — чёрный, а если все три основных цвета имеют яркость FF, то получается «самый светлый серый цвет» — белый. Во всех «промежуточных» случаях серый цвет тем темнее, чем меньше числовое значение яркости основных цветов, и наоборот.

В задаче, среди приведённых вариантов ответов, имеется три «монохромных» цвета (т.е. фиолетовый отпадает сразу). Далее — не белый цвет (белый кодировался бы числовой записью FFFFFF) и не чёрный (для которого кодовая запись соответствует 000000). Следовательно, записанный в теге цвет — серый:

Ответ: серый цвет (вариант ответа №1).

Раздел 8. **Обработка числовой информации**

A7 **Электронные таблицы. Ссылки. Формулы**

Конспект

Электронные таблицы Exel

Имя текущей ячейки образуется именем столбца и номером строки, на пересечении которых она находится. В данном случае это ячейка **C3**.

Имя *диапазона* (*блока*) ячеек образуется записью имён его верхней левой и нижней правой ячеек через двоеточие. В данном случае это диапазон **B3:D4**.

Ячейка может содержать:

- число (в том числе в специальном формате, например, денежном или процентном);
- дату и/или время;
- текст (значение, не являющееся числом, датой или формулой, либо введённое в кавычках);
- логическое значение (ИСТИНА или ЛОЖЬ);
- формулу.

Формулы

Обращение из одной ячейки к данным в другой ячейке (источнике) производится с помощью формул.

Формула всегда начинается со знака равенства (=) и может содержать:

- константы (числовые, текстовые, логические);
- ссылки на другие ячейки или диапазоны с данными;
- арифметические операции:

+ — сложение,
– — вычитание,
* — умножение,
/ — деление,
% — вычисление процентов;
^ — возведение в степень;

- операции сравнения:

= — равенство,
> — больше,
>= — больше или равно,
< — меньше,
<= — меньше или равно,
<> — не равно;

- операцию конкатенации (& — соединение текстовых строк);
- круглые скобки;
- функции.

Абсолютные, относительные и смешанные ссылки

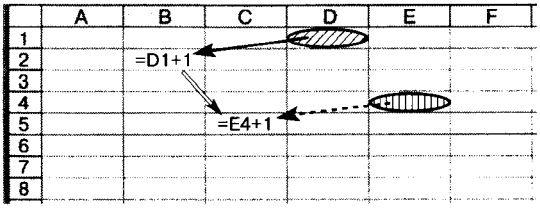
Ссылка на ячейку представляет собой запись имени ячейки. Ссылка на диапазон представляет собой запись имени диапазона.

Примеры:

=A3 + C5 — сложить числа в ячейках **A3** и **C5**;

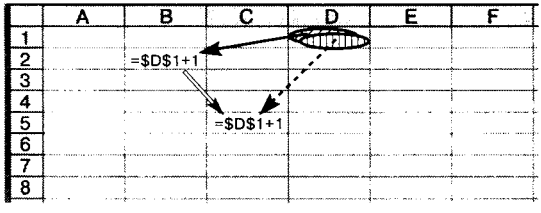
=СУММ(A3:C5) — функция вычисления суммы чисел во всём диапазоне **A3:C5**.

При копировании формулы с такими ссылками в другие ячейки ссылки автоматически изменяются (модифицируются) так, что всегда указывают на ячейки или диапазоны относительно ячейки, содержащей формулу. Поэтому такие ссылки называют **относительными**.



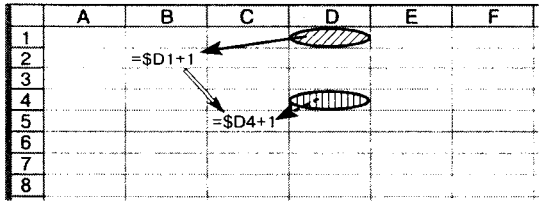
При копировании формулы из ячейки **B2** в ячейку **C5** ссылка меняется так, что всегда указывает на ячейку, расположенную на 2 столбца правее и на 1 строку выше *относительно* ячейки с формулой.

Запись имени ячейки (или имён ячеек в имени диапазона), в которой имя столбца и номер строки предваряются символом $, являются **абсолютными** ссылками. Абсолютная ссылка не меняется при копировании формулы в другую ячейку.

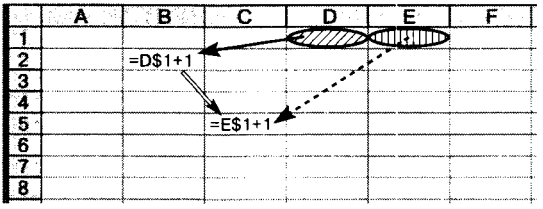


При копировании формулы из ячейки **B2** в ячейку **C5** ссылка не меняется и всегда указывает на одну и ту же ячейку.

Ссылки, в которых символ $ стоит только перед именем столбца или только перед номером строки, называют **смешанными**. Символ $ в смешанной ссылке делает абсолютным только имя столбца или только имя строки, перед которым он стоит.



В данной смешанной ссылке абсолютным является имя столбца. Поэтому при копировании формулы из ячейки **B2** в ячейку **C5** в ссылке имя столбца не меняется, а номер строки меняется относительно ячейки с формулой.



В данной смешанной ссылке абсолютным является номер строки. Поэтому при копировании формулы из ячейки **B2** в ячейку **C5** в ссылке номер строки не меняется, а имя столбца меняется относительно ячейки с формулой.

Функции

В Excel предусмотрены стандартные функции, которые можно использовать в формулах:

- математические — различные вычисления (арифметические, тригонометрические, логарифмические, квадратный корень, степень, округление и т.д.);
- текстовые — работа с текстовыми строками;
- логические — работа с логическими значениями;
- дата и время — работа с датой и временем;
- финансовые — денежные расчёты (проценты по вкладам и пр.);
- статистические — статистическая обработка данных, вероятности и пр.;
- ссылки и массивы — работа с данными в диапазонах (например, поиск значения и возврат адреса ячейки с ним);
- работа с базой данных — работа с записями в электронной таблице как базе данных;
- проверка свойств и значений — определение типа данных в ячейке (число? текст? и т.д.).

В формуле записывается имя функции, после которого в круглых скобках через точку с запятой записываются значения — аргументы.

Функции могут быть вложенными: в качестве аргумента одной функции записывается другая функция со своими аргументами.

Примеры функций:

ПИ()	Возвращает число $\pi = 3{,}14159265358979$
ABS(*число*)	Возвращает модуль (абсолютную величину) числа
ЗНАК(*число*)	Определяет знак вещественного числа: равна 1, если число положительное; 0, если число равно 0, и −1, если число отрицательное
КОРЕНЬ(*число*)	Возвращает значение квадратного корня
ОСТАТ(*число*;*делитель*)	Возвращает остаток от деления аргумента на делитель
СЛЧИС( )	Возвращает равномерно распределенное случайное число из диапазона [0,1)
СТЕПЕНЬ(*число*;*степень*)	Возвращает результат возведения вещественного числа в заданную степень (аналог оператора ^)
СУММ(*число1*;*число2*; ...)	Сумма чисел, заданных в качестве аргументов
СРЗНАЧ(*число1*; *число2*; ...)	Возвращает среднее арифметическое аргументов

Продолжение табл.

СЧЁТЕСЛИ(*диапазон*;*критерий*)	Подсчитывает количество ячеек внутри диапазона, удовлетворяющих заданному критерию (он записывается в форме числа, выражения или текста, который определяет, какие ячейки надо подсчитывать, например: 32, «32», «>32», «яблоки» и т.п.)
МАКС(*число1*;*число2*; ...)	Возвращает наибольшее (максимальное) из набора значений
МИН(*число1*;*число2*; ...)	Возвращает наименьшее (минимальное) из набора значений
ЛЕВСИМВ(*текст*; *количество_знаков*)	Возвращает подстроку из указанного количества символов с начала текстовой строки
ПРАВСИМВ(*текст*;*число_знаков*)	Возвращает подстроку из указанного количества символов, отсчитанного с конца текстовой строки
ПСТР(*текст*; *начальная_позиция*; *число_знаков*)	Возвращает подстроку из указанного количества символов, отсчитанного с указанной начальной позиции текстовой строки
НАЙТИ(*искомый_текст*; *просматриваемый_текст*; *нач_позиция*)	Возвращает номер знаковой позиции, начиная с которой в тексте *просматриваемый_текст* содержится фрагмент *искомый_текст*.

Продолжение табл.

	Нач_позиция — знаковая позиция в исходном тексте, с которой следует начинать поиск (по умолчанию — 1, т.е. поиск с начала исходной строки)
ПОВТОР(*текст*;*число_повторений*)	Повторяет указанный текст заданное количество раз (положительное число)
СЦЕПИТЬ(*текст1*;*текст2*;...)	Соединяет несколько текстовых строк в одну (операция *конкатенации*) (аналог операции &)
ИЛИ(*логическое_значение1*; *логическое_значение2*; ...)	Логическая операция ИЛИ (дизъюнкция, логическое сложение)
И(*логическое_значение1*; *логическое_значение2*; ...)	Логическая операция И (конъюнкция, логическое умножение)
НЕ(*логическое_значение*)	Логическая операция НЕ (отрицание)
ЕСЛИ(*лог_выражение*; *значение_если_истина*; *значение_если_ложь*)	Аналог условного оператора IF: сначала определяется истинность заданного логического выражения, а затем в ячейке, содержащей данную функцию, отображается, соответственно, одно из значений (аргумент *значение_если_ложь* может быть опущен, тогда при ложном значении логического выражения в ячейке ничего не выводится).

Окончание табл.

	Лог_выражение — это любое значение или выражение, принимающее значения **ИСТИНА** или **ЛОЖЬ**. Функции **ЕСЛИ** могут быть вложенными, когда, например, вместо *значение_если_истина* и/или *значение_если_ложь* записывается ещё одна функция **ЕСЛИ** со своими тремя (либо двумя) аргументами
СЕГОДНЯ( )	Возвращает текущую дату (по показаниям системного календаря) в числовом формате

Разбор типовых задач

Задача 1*. В электронной таблице значение формулы **=СУММ(B1:B2)** равно 5. Чему равно значение ячейки **B3**, если значение формулы **=СРЗНАЧ(B1:B3)** равно 3?

1) 8 2) 2 3) 3 4) 4

Решение

Речь идёт о таблице, состоящей из трёх ячеек: **B1**, **B2** и **B3**. При этом **B1** + **B2** равно 5, а среднее значение, т.е. (**B1** + **B2** + **B3**)/3 равно 3. Требуется определить значение **B3**.

Значение первой суммы подставляется во второе равенство:

(5 + **B3**)/3 = 3.

Полученное уравнение остаётся решить относительно значения (переменной) **B3**:

5 + **B3** = 9;

отсюда **B3** = 9 − 5 = 4.

Ответ: 4 (вариант ответа №4).

Задача 2*. В динамической (электронной) таблице приведены значения пробега автомашин (в км) и общего расхода дизельного топлива (в литрах) в четырёх автохозяйствах с 12 по 15 июля. В каком из хозяйств средний расход топлива на 100 км пути за эти четыре дня наименьший?

Название автохозяйства	12 июля		13 июля		14 июля		15 июля		За четыре дня	
	Пробег	Расход	Пробег	Расход	Пробег	Расход	Пробег	Расход	Пробег	Расход
Автоколонна №11	9989	2134	9789	2056	9234	2198	9878	2031	38890	8419
Грузовое такси	490	101	987	215	487	112	978	203	2942	631
Автобаза №6	1076	147	2111	297	4021	587	1032	143	8240	1174
Трансавтопарк	998	151	2054	299	3989	601	1023	149	8064	1200

1) Автоколонна № 11
2) Грузовое такси
3) Автобаза №6
4) Трансавтопарк

Решение

Необходимо сравнивать средний расход топлива на 100 км пути за четыре дня. Его можно вычислить как частное от деления значения в графе «Расход» на значение в графе «Пробег» из колонки таблицы «За четыре дня» (остальная информация в данном случае избыточна).

Составим таблицу:

Название автохозяйства	Пробег за четыре дня	Расход за четыре дня	Средний расход (приблизительно)
Автоколонна №11	38890	8419	0,216
Грузовое такси	2942	631	0,214
Автобаза №6	8240	1174	0,142
Трансавтопарк	8064	1200	0,49

Вывод: наименьший средний расход топлива на 100 км пути за четыре дня — в автохозяйстве «Автобаза №6».

Ответ: Автобаза №6 (вариант ответа №3)

Задача 3*. В динамической (электронной) таблице приведены значения посевных площадей (в га) и урожая (в центнерах) четырёх зерновых культур в четырёх хозяйствах одного района. В каком из хозяйств достигнута максимальная урожайность зерновых (по валовому сбору)? (Урожайность измеряется в центнерах с гектара.)

Зерновые культуры	Названия хозяйства							
	Заря		Первомайское		Победа		Рассвет	
	Посевы	Урожай	Посевы	Урожай	Посевы	Урожай	Посевы	Урожай
Пшеница	600	15300	900	23800	300	7500	1200	31200
Рожь	100	2150	500	1200	50	1100	250	5500
Овёс	100	2350	400	10000	50	1200	200	4800
Ячмень	200	6000	200	6300	100	3100	350	10500
Всего зерновые	1000	25800	2000	52100	500	12900	2000	52000

1) Заря
2) Первомайское
3) Победа
4) Рассвет

Решение

Необходимо определить максимальную урожайность всех зерновых в центнерах с гектара. Поэтому для расчётов нужно брать данные из последней строки таблицы («Всего зерновые»). Урожайность определяется как отношение значения урожая (в центнерах) к значению посевной площади («Посев») в гектарах.

Составляется таблица:

Хозяйство	Площадь земель	Урожай	Урожайность
Заря	1000	25800	25,8
Первомайское	2000	52100	26,05
Победа	500	12900	25,8
Рассвет	2000	52000	26

Вывод: максимальная урожайность получена в хозяйстве «Первомайское».

Ответ: Первомайское (вариант ответа №2).

Задача 4*. Три страны: Королевство Бельгия. Королевство Нидерланды и Великое Герцогство Люксембург образуют экономико-политический союз, который носит название Бенилюкс. Ниже приведён фрагмент электронной таблицы, характеризующий каждую из стран союза и союз в целом:

	A	B	C	D
1	Страна	Население (тыс. чел.)	Площадь (кв. км.)	Плотность насел. (чел. / кв. км.)
2	Бельгия	10 415	30 528	341
3	Нидерланды	16 357	41 526	394
4	Люксембург	502	2 586	194
5	Бенилюкс	27 274	74 640	

Какое значение должно стоять в ячейке **D5**?

1) 365 2) 929 3) 310 4) 2,74

Решение

Плотность населения вычисляется как частное от деления значения в графе «Население» на значение в графе «Площадь» в каждой строке таблицы.

Разделив в строке «Бенилюкс в целом» значение 27 274 000 (население указано в тысячах человек) на значение 74 640 (площадь), получается искомое значение в ячейке **D5**: 365,4.

Поскольку в списке ответов точное значение отсутствует, оно округляется до: $365{,}4 \approx 365$.

Ответ: 365 чел / кв.км. (вариант ответа №1).

Следует не забывать переводить приведённые в условии задачи значения в нужную размерность.

Если среди приведённых в задаче вариантов ответа отсутствует вычисленное точное значение, нужно найти в вариантах ответа округлённое значение.

B3 Электронные таблицы. Графики и диаграммы

Конспект

Диаграммы

Диаграмма строится по числовым значениям в некотором диапазоне (блоке) таблицы. При этом подписи диаграммы обычно также берутся из соответствующих диапазонов таблицы (из её «шапки» и/или левого столбца с названиями строк).

При этом в таблице определяются исходные и зависимые величины. Исходные величины называют **категориями**, а зависящие от них величины называют **рядами данных**.

Пример (таблица зависимости координаты тела, движущегося по прямой, от времени для трёх различных экспериментов):

t	1	2	3	4	5	6	7	8	Категории
x_1	2	2,5	3	4	6	9	15	27	
x_2	1,5	2	2,5	3	3,5	4	4,5		Ряды данных
x_3	1	4	9	16	25	36	49	64	

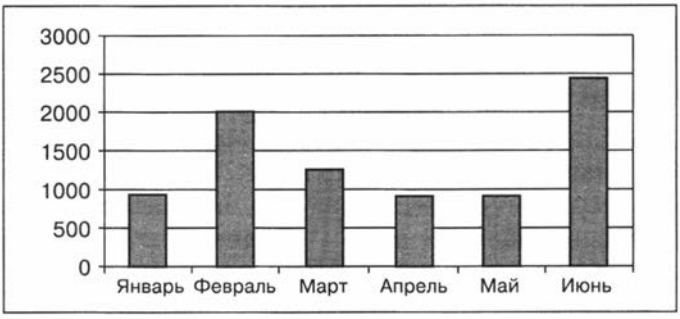
На диаграмме, в зависимости от её вида, может быть отображён только один ряд данных либо несколько рядов данных. Значения категорий при этом являются подписями к элементам диаграммы, соответствующим значениям ряда/рядов, либо определяют значения по оси X (тогда значения рядов данных — это значения по оси Y).

Имена рядов данных, как правило, отображаются на диаграмме в составе её **легенды** — таблицы, определяющей соотношение имени каждого ряда данных и его графического отображения (цвета линии, вида значков, соответствующих точкам и т.д.).

Основные виды диаграмм

1. **Гистограмма** — состоит из вертикальных прямоугольников. Каждый столбик соответствует одному значению таблицы для соответствующей категории.

Пример чтения гистограммы:



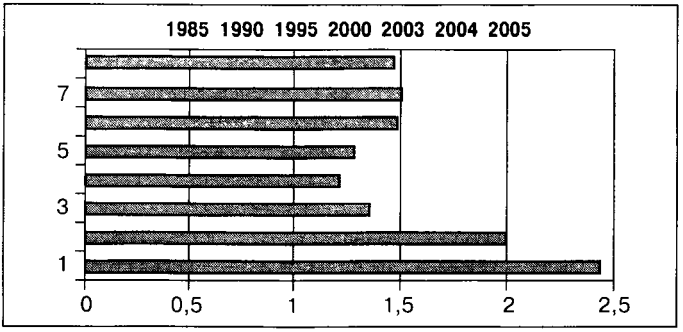
Ось категорий располагается по горизонтали (здесь — содержит названия месяцев).

Ряд данных — один. Числовые значения соответствуют высотам столбиков и приближённо определяются по вертикальной оси (слева).

Месяц	январь	февраль	март	апрель	май	июнь
Значение	900	2000	1200	900	900	2500

2. **Линейчатая диаграмма** — состоит из горизонтальных полосок. Каждая полоска соответствует одному значению таблицы для соответствующей категории.

Пример чтения линейчатой диаграммы:



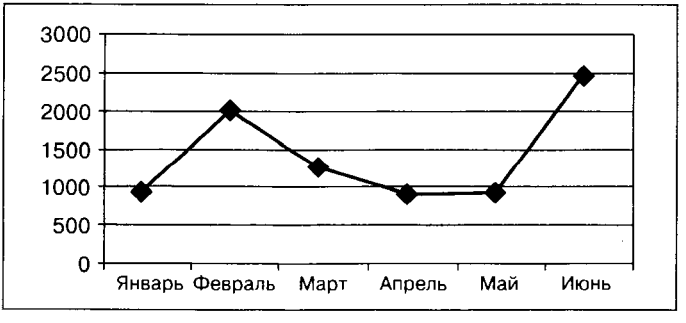
Ось категорий располагается по вертикали, слева (здесь — содержит числа — порядковые номера).

Ряд данных — один. Числовые значения соответствуют длинам полосок и приближенно определяются по горизонтальной оси.

№ п/п	1	2	3	4	5	6	7	8
Значение	2,4	2	1,35	1,2	1,3	1,49	1,5	1,45

3. График — представляет собой набор точек и/или соединяющую их линию. Каждая точка соответствует одному значению таблицы для соответствующей категории.

Пример чтения графика:



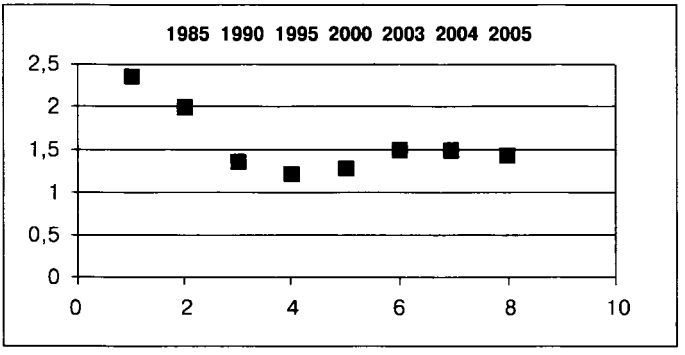
Ось категорий располагается по горизонтали (здесь — содержит названия месяцев).

Ряд данных — один. Числовые значения соответствуют вертикальным координатам расположения точек и приближённо определяются по вертикальной оси (слева).

Месяц	январь	февраль	март	апрель	май	июнь
Значение	900	2000	1200	900	900	2500

4. **Точечная диаграмма** — аналогична графику (представляет собой набор точек и/или соединяющую их линию), но здесь при построении диаграммы компьютер автоматически выстраивает числовые значения категорий по возрастанию по оси *X* и строит график зависимости от них значений ряда (рядов) данных. Именно точечная диаграмма является в электронных таблицах аналогом графиков в математике.

Пример чтения точечной диаграммы:



Ось категорий располагается по горизонтали и содержит значения *X*.

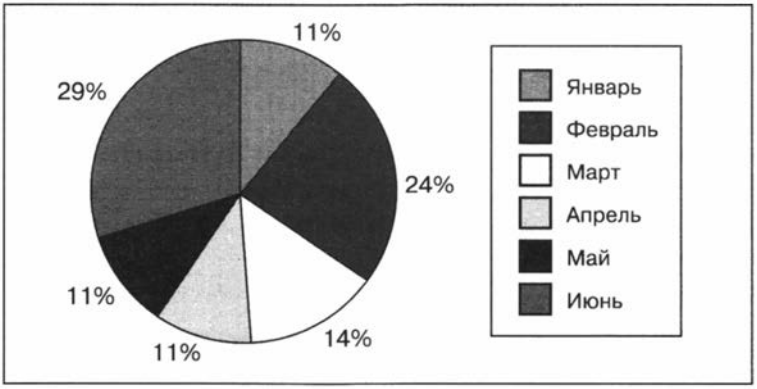
Ряд данных — один. Числовые значения соответствуют вертикальным координатам расположения точек и приближённо определяются по вертикальной оси (слева).

X	1	2	3	4	5	6	7	8
Y	2,4	2	1,35	1,2	1,3	1,49	1,5	1,45

В гистограмме, линейчатой диаграмме, графике, точечной диаграмме для вычисления процентных долей значений ряда данных относительно суммы для всего ряда нужно сложить все значения ряда данных, определить тем самым общее значение и вычислять процентные доли делением каждого значения ряда данных на сумму их значений.

5. **Круговая диаграмма** — представляет собой набор секторов круга. Каждый сектор соответствует процентной доле одного значения таблицы для соответствующей категории относительно суммы всех значений ряда данных.

Пример чтения круговой диаграммы:



Категории определяются по легенде или надписям на самой диаграмме (здесь — названия месяцев).

Ряд данных — один (круговая диаграмма всегда отображает один ряд данных). Процентные доли каждого значения ряда данных, соответствующего той или иной категории, могут быть надписаны на диаграмме либо определяются приближённо как процентные доли углов соответствующих секторов относительно всего круга (360°).

Круговая диаграмма не даёт информации об абсолютных значениях ряда данных! Для их определения необходимо знать суммарное значение по всему ряду данных либо абсолютное значение хотя бы одного элемента данных (дальнейший расчёт выполняется пропорционально).

Пусть известно, что, например, июню (29%) соответствует числовое значение 870. Тогда можно по имеющейся диаграмме восстановить (рассчитать) значения для остальных категорий:

Месяц	январь	февраль	март	апрель	май	июнь	СУММАРНО
Доля, %	11	24	14	11	11	29	100
Значение	330	720	420	330	330	870	3000

6. **Гистограмма для нескольких рядов данных** — состоит из наборов соответствующих количеств столбцов разного цвета или фактуры (каждому ряду данных соответствует свой цвет или фактура, указанный в легенде).

Пример чтения гистограммы для нескольких рядов данных:

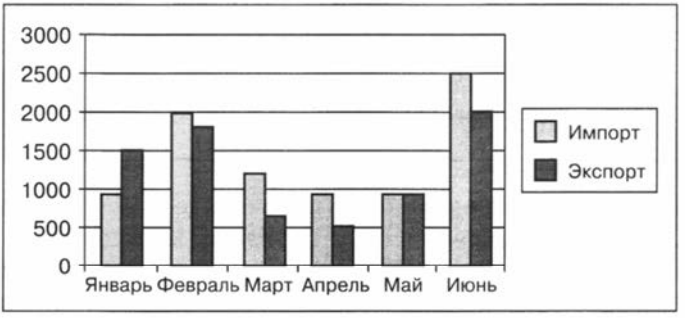


Диаграмма читается аналогично обычной гистограмме, но нужно отдельно считывать значения для столбиков каждого ряда данных.

Ось категорий располагается по горизонтали (здесь — содержит названия месяцев).

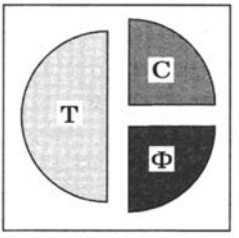
Рядов данных — два (первый, «импорт», — светло-серый цвет, второй, «экспорт», — тёмно-серый цвет). Числовые значения соответствуют высотам столбиков и приближённо определяются по вертикальной оси (слева).

Месяц	январь	февраль	март	апрель	май	июнь
Импорт	900	2000	1200	900	900	2500
Экспорт	1500	1800	700	500	900	2000

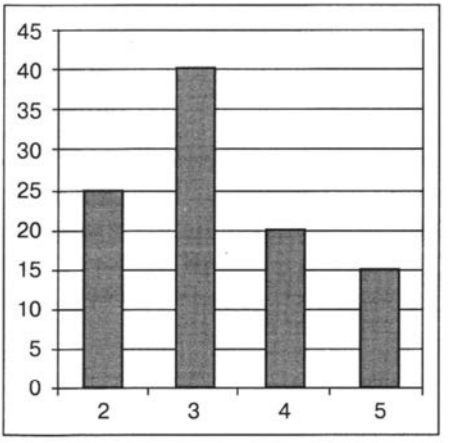
Разбор типовых задач

Задача 1*. В цехе трудятся рабочие трёх специальностей — токари (Т), слесари (С) и фрезеровщики (Ф). Каждый рабочий имеет разряд, не меньший второго и не больший пятого. На диаграмме I отражено распределение рабочих по специальностям, а на диаграмме II — количество рабочих с различными разрядами. Каждый рабочий имеет только одну специальность и один разряд.

I



II



Какое из утверждений:

А) Среди слесарей найдётся хотя бы один третьего разряда

Б) Среди токарей найдётся хотя бы один второго разряда

В) Все токари могут иметь четвёртый разряд

Г) Все фрезеровщики могут иметь третий разряд следует из диаграмм?

1) А 2) Б 3) В 4) Г

Решение

Из диаграммы II следует, что больше всего рабочих 3-го разряда (40 чел.); второй разряд имеют 25 чел.; четвёртый разряд имеют 20 чел.; меньше всего рабочих 5-го разряда (15 чел.). Общее же число рабочих равно 40 + 25 + 20 + 15 = 100 чел.

Из диаграммы I следует, что токарей в цехе больше всего, их 50%, т.е. 50 чел. Количества же слесарей и фрезеровщиков одинаковы — по 25%, т.е. по 25 чел.

Последовательно анализируется справедливость каждого из приведённых утверждений в соответствии с обеими диаграммами.

А) Среди слесарей найдётся хотя бы один третьего разряда. Токарей и фрезеровщиков: 50 + 25 = 75 чел., а рабочих 3-го разряда — 40 чел. Поэтому может оказаться так, что все рабочие 3-го разряда будут только токарями и фрезеровщиками, а все слесари будут иметь 2-й, 4-й или 5-й разряд.

Б) Среди токарей найдётся хотя бы один второго разряда. Слесарей и фрезеровщиков (как следует из диаграммы I) 50 чел., а 2-й разряд имеют 25 чел. Поэтому может оказаться, что 2-й разряд имеют только слесари и фрезеровщики, но ни одного токаря.

В) Все токари могут иметь четвертый разряд. Это невозможно: токарей (как следует из диаграммы I) 50 чел., а 4-й разряд имеют только 20 чел. Значит, как минимум 30 токарей должны иметь разряд, отличный от 4-го.

Г) Все фрезеровщики могут иметь третий разряд. Это вполне возможно: фрезеровщиков — 25 чел., а имеющих 3-й разряд — 40 чел. Следовательно, все 25 фрезеровщиков могут иметь 3-й разряд.

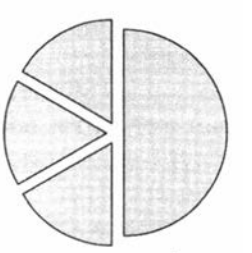
Вывод: истинным из четырёх приведённых в задаче высказываний является высказывание Г.

Ответ: высказывание Г (вариант ответа №4).

Задача 2*. Дан фрагмент электронной таблицы:

	A	B	C	D
1	3		3	2
2	=(C1 + A1)/2	=C1 – D1	=A1 – D1	=B1/2

Какое число должно быть записано в ячейке **B1**, чтобы построенная после выполнения вычислений диаграмма по значениям диапазона ячеек **A2:D2** соответствовала рисунку:



Решение

В предыдущих задачах требовалось по результатам расчётов в таблице указать, какая диаграмма по ней будет построена (это были задачи группы А). Теперь задача стала сложнее — надо по виду диаграммы (причём «слепой» — без легенды и подписей!) определить недостающее значение в ячейке таблицы.

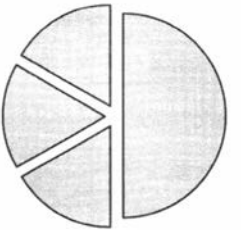
Искомое число в ячейке **B1** обозначается как x и выполняются (по возможности и с учётом этой неизвестной) все вычисления по предложенным в строке **2** формулам:

	A	B	C	D
1	3	x	3	2
2	(3 + 3)/2	3 – 2	3 – 2	x/2

или

	A	B	C	D
1	3	x	3	2
2	3	1	1	x/2

Полученный набор числовых значений в строке **2** сравнивается с диаграммой:



На этой диаграмме — три равных сектора, в сумме дающих 180°, и значит, равных 60° каждый, и один сектор в 180° (т.е. равный трём остальным вместе). Такая ситуация возможна, если большой сектор соответствует значению 3, а малые секторы — каждый значению 1.

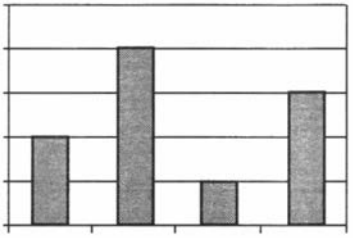
Тогда $x / 2 = 1$, откуда $x = 2$.

Ответ: в ячейке **B1** должно быть записано число 2.

Задача 3*. Дан фрагмент электронной таблицы:

	A	B	C	D
1	3		5	
2	=C1 – 3	=(A1 + C1)/2	=A1/3	=(B1 + A2)/2

Какое число должно быть записано в ячейке **B1**, чтобы построенная после выполнения вычислений диаграмма по значениям диапазона ячеек **A2:D2** соответствовала рисунку?



Решение

1. В формулы подставляются все известные константы (значения соответствующих ячеек) и выполняются все возможные вычисления:

	A	B	C	D
1	3		5	
2	5 – 3 = 2	(3 + 5)/2 = 4	3/3 = 1	(B1 + 2)/2

Диапазон значений для построения диаграммы:

	A	B	C	D
2	2	4	1	(B1 + 2)/2

2. Диаграмму, приведённая на рисунке — это гистограмма. Её столбцы непосредственно определяют соответствующие числовые значения. В данном случае эти значения пропорциональны:

2	4	1	3

3. При построении гистограммы по значениям некоторого диапазона порядок следования столбцов гистограммы соответствует порядку следования ячеек. Сопоставляя известные значения в ячейках **A2:D2** и значения, считанные с гистограммы, получается, что коэффициент пропорциональности равен 1, а значение ячейки **D2** должно быть равно 3.

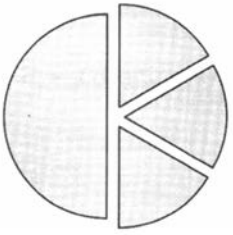
4. Тогда (**B1** + 2)/2 = 3. Отсюда **B1** + 2 = 6, тогда **B1** = 4.

Ответ: **4**.

Задача 4*. Дан фрагмент электронной таблицы:

	A	B	C	D
1			5	4
2	=D1 – 3	=C1 – D1	=(A2 + B2)/2	=B1 – D1 + C2

Какое число должно быть записано в ячейке **B1**, чтобы построенная после выполнения вычислений диаграмма по значениям диапазона ячеек **A2:D2** соответствовала рисунку?



Решение

1. В формулы подставляются все известные константы (значения соответствующих ячеек) и выполняются все возможные вычисления:

	A	B	C	D
1			5	4
2	4 – 3 = 1	5 – 4 = 1	(1 + 1)/2 = 1	B1 – 4 + 1

Диапазон значений для построения диаграммы:

	A	B	C	D
2	1	1	1	B1–3

2. Диаграмму, приведённая на рисунке — это круговая диаграмма. Её секторы определяют процентное соотношение соответствующих числовых значений. В данном случае эти соотношения можно оценить так:

$^1/_2$	$^1/_6$	$^1/_6$	$^1/_6$

3. При построении круговой диаграммы по значениям некоторого диапазона порядок следования секторов соответствует порядку следования ячеек, но отсчёт секторов может начинаться с любой ячейки. Сопоставляя известные значения в ячейках **A2:D2** и значения, считанные с круговой диаграммы, можно сделать следующие выводы:

- три равных сектора, соответствующие $^1/_6$ частям диаграммы, соответствуют значениям ячеек **A2:C2** (числовое значение 1);
- сектор, соответствующий $^1/_2$ части диаграммы, соответствует значению ячейки **D2**, а его числовое значение втрое больше значений в ячейках **A2:C2**, т.е. равно 3.

  Тогда **B1** – 3 = 3. Отсюда **B1** = 6.

  Ответ: 6.

Раздел 9. Технологии поиска и хранения информации

А6 Базы данных. Сортировка данных. Запросы в базах данных

Конспект

База данных (БД) — организованная по определённым правилам (в соответствии с некоторой схемой) совокупность логически связанных и длительно хранимых в памяти ПК или на информационном носителе данных, характеризующая состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Отличительные признаки БД:

1) БД хранится и обрабатывается в вычислительной системе (т.е. некомпьютерные хранилища информации — архивы, картотеки и прочее — не считаются базами данных);

2) данные в БД логически структурированы (систематизированы) для обеспечения возможности их эффективного поиска и обработки в вычислительной системе;

3) БД включает в себя схему (метаданные), описывающие логическую структуру БД в формальном виде.

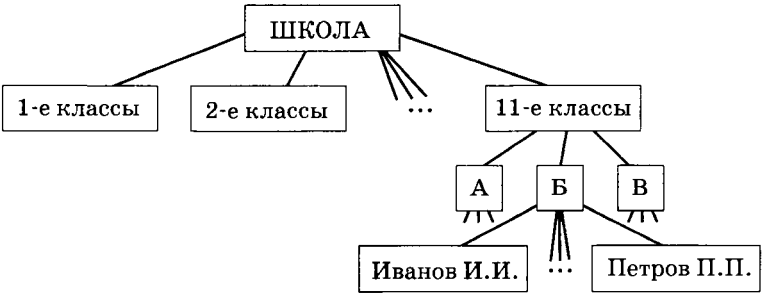
Система управления базами данных (СУБД) — комплекс программных и языковых средств для создания и модификации структуры БД, добавления, изменения, удаления, поиска и отбора данных, их представления на экране ПК и в печатном виде, разграничения прав доступа к информации и выполнения других операций с БД.

Нельзя путать понятия *базы данных* и *системы управления базами данных*!

База данных — это структурированные данные. Система управления базой данных — это инструмент для работы с базой данных.

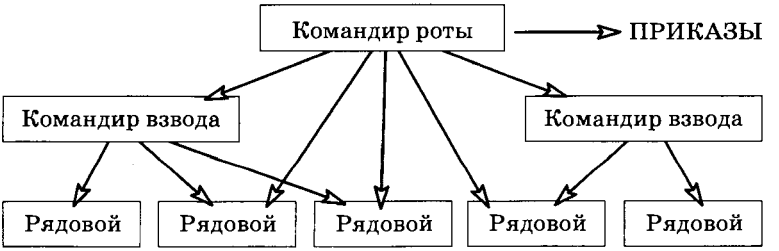
Классификацию БД обычно проводят по типам используемых в них **моделей данных**:

1) **иерархическая модель данных** — БД представлена в виде древовидной (иерархической) структуры (графа), состоящей из объектов данных различных уровней; при этом объект может включать в себя несколько объектов более низкого уровня.



2) **сетевая модель данных** состоит из набора записей и набора связей между этими записями; она в чём-то аналогична иерархической модели, но в сетевой БД связи являются направленными и могут соединять объекты разных ветвей дерева.

Пример:



3) **реляционная БД** состоит из нескольких взаимосвязанных таблиц:

Таблица 1

ID	Фамилия	Имя	Отчество
1	Иванов	Иван	Иванович
2	Петров	Пётр	Петрович
3	Олегов	Олег	Олегович
4	Васин	Василий	Петрович

Таблица 2

ID	№ заказа	Товар
15	134/15	Картридж
18	157/18	Принтер
2	111/12	Сканер
6	325/32	Монитор

Таблица 3

ID	Адрес	Телефон
17	. . .	. . .
11	. . .	. . .
2	. . .	. . .
8	. . .	. . .

Рассматривая таблицы, связанные по значениям поля **ID**, можно определить, что, например, Петров Пётр Петрович (**ID** = 2 — *таблица 1*) заказал сканер, № заказа = 111/12 (*таблица 2*), а также узнать его адрес и телефон (*таблица 3*).

Связи («реляции») между двумя какими-либо таблицами осуществляются через общее для них по смыслу (но не обязательно одинаковое по названию) поле. При этом возможны связи:

- **«один к одному»** — одной записи первой таблицы соответствует одна, и только одна запись второй таблицы, и наоборот (пример: в ОС MS-DOS полному имени файла однозначно соответствует запись номера начального кластера);
- **«один ко многим»** — одной записи первой таблицы может соответствовать много записей второй таблицы (пример: один и тот же учитель может вести уроки в нескольких классах);
- **«многие к одному»** — много записей первой таблицы могут соответствовать одной записи второй таб-

лицы (пример: у нескольких учеников занятия по предмету ведёт один и тот же учитель); связи «многие к одному» и «один ко многим» являются аналогами друг друга;

- **«многие ко многим»** — много записей в первой таблице могут быть связаны с многими записями второй таблицы (пример: одного и того же ученика могут учить разные учителя, а один и тот же учитель может учить множество учеников). *Подобный тип связей в реляционных БД не допускается и при необходимости реализуется как две связи «один ко многим» через промежуточную таблицу (в приведённом только что примере учитель связывается с учеником через номер класса и предмет).*

Показатель количеств связываемых объектов называют **кардинальностью** связи:

Тип связи	Кардинальность
«один к одному»	1:1
«один ко многим»	1:*N*
«многие к одному»	*N*:1
«многие ко многим»	*N*:*N*

Поиск данных в реляционной БД требует перехода от одной БД к другой в соответствии с имеющимися связями (в том числе многократно).

Основой реляционной БД является **таблица**.

Поля БД — это характеристики объектов (сущностей), информация о которых хранится в БД. Поля БД соответствуют столбцам таблицы.

Записи БД — это информация о каждом из объектов (одному объекту соответствует одна запись), выраженная в виде значений соответствующих полей. Записям БД соответствуют строки таблицы.

Поля: / Записи:	Поле 1	Поле 2	...	Поле *n*
Запись 1	*Данное*	*Данное*	...	*Данное*
Запись 2	*Данное*	*Данное*	...	*Данное*
...	...	...	...	...
Запись *m*	*Данное*	*Данное*	...	*Данное*

Характеристики, отражённые в виде полей БД, являются едиными (общими) для всех объектов. Объекты в БД должны различаться хотя бы одним значением какой-либо характеристики.

Ключевое поле — поле БД, значения которого гарантированно различаются для разных объектов. По значению ключевого поля всегда можно однозначно выделить соответствующий объект.

Выборка данных из БД — операция отбора записей БД (строк таблицы), соответствующих заданному условию (**запросу на выборку**).

Условие (запрос) может быть **простым** (накладывается на значения какого-то одного поля либо выражено в сравнении двух каких-либо полей) или **составным** (простые условия объединяются при помощи логических операций **И, ИЛИ, НЕ**).

Распределённая БД — совокупность логически взаимосвязанных БД, размещённых на различных узлах компьютерной сети.

Практические приёмы работы с БД

1. Поиск (выборка) информации в однотабличной БД

Запрос на поиск задаётся в виде структуры, аналогичной структуре записи БД, где отдельные поля могут быть пусты (пропущены в запросе), содержат константное значение или условие, накладываемое на значение данного поля.

Процесс поиска заключается в поочередном просмотре записей БД и выборе (с формированием отдельной вре́менной таблицы, называемой *выборкой*) таких запи-

сей, значения соответствующих полей которых равны заданной константе либо удовлетворяют заданному условию. Структура формируемой таблицы (выборки) совпадает со структурой исходной БД, но некоторые поля исходной БД в выборке могут быть пропущены по желанию пользователя, выдавшего поисковый запрос.

Ручной поиск в БД:

- записи БД просматриваются поочерёдно;
- если значение первого по счёту (слева направо) непустого поля запроса совпадает с константой, заданной в том же поле в запросе, либо удовлетворяет условию, заданному в том же поле в запросе, то помечается эта запись БД;
- проверяются в этой записи БД остальные поля (слева направо) на соответствие константам либо условиям в тех же полях запроса;
- если *все* эти поля записи БД удовлетворяют значениям/условиям в таких же полях запроса, то эта запись БД *включается в выборку*; если *хотя бы одно* поле записи не удовлетворяет значению/условию в таком же поле запроса, то эта запись БД *пропускается* (не включается в выборку).

2. Сортировка записей БД

Для выполнения сортировки записей БД задаются:

- одно или несколько названий полей, по содержимому которых нужно выполнить сортировку (порядок перечисления полей в запросе на сортировку *важен* и не обязательно соответствует порядку следования этих полей слева направо в структуре записи БД);
- для каждого такого поля — условие сортировки (для чисел — по возрастанию или по убыванию, для текста — по алфавиту или в порядке, обратном алфавитному).

Процесс сортировки заключается в том, что записи БД сначала переставляются в таблице по порядку, соответствующему условию сортировки по первому заданному полю. Затем в пределах *групп* записей с *одинаковым* значением в первом поле выполняется переста-

новка этих записей по порядку, соответствующему условию сортировки по второму заданному полю. Далее аналогично в пределах *подгрупп* записей с *одинаковым* значением во втором поле (а также в первом!) выполняется перестановка записей по порядку, соответствующему условию сортировки по третьему заданному полю и т.д.

Ручная сортировка в БД

Пусть нужно отсортировать записи БД — адресной книги по полям «Фамилия» (по алфавиту), «Имя» (по алфавиту) и «Отчество» (по алфавиту):

Фамилия	Имя	Отчество
Иванов	Дмитрий	Николаевич
Бунин	Андрей	Карлович
. . .		
Львов	Илья	Андреевич
Иванов	Устин	Сергеевич
Александров	Николай	Иванович
. . .		
Иванов	Дмитрий	Алексеевич

Тогда сначала выполняется такая перестановка записей БД, что в поле «Фамилия» значения (сверху вниз) следуют по алфавиту либо повторяются, например:

Фамилия	Имя	Отчество
Александров	Николай	Иванович
Бунин	Андрей	Карлович
. . .		
Иванов	Устин	Сергеевич
Иванов	Дмитрий	Николаевич
Иванов	Дмитрий	Алексеевич
. . .		
Львов	Илья	Андреевич

 Записи БД переставляются (меняются местами) только *целиком* (целыми строками таблицы). Нельзя менять местами только значения в сортируемом поле, оставляя на прежних местах значения остальных полей!

В таблице присутствует группа записей с одинаковым значением первого сортируемого поля («Фамилия» — «Иванов»). В пределах этой группы (и только в ней!) выполняется сортировка записей по значениям второго заданного поля «Имя»:

Фамилия	Имя	Отчество
Александров	Николай	Иванович
Бунин	Андрей	Карлович
. . .		
Иванов	*Дмитрий*	Николаевич
Иванов	*Дмитрий*	Алексеевич
Иванов	*Устин*	Сергеевич
. . .		
Яхин	Роман	Константинович

Выделяется подгруппа записей с одинаковым значением второго сортируемого поля («Имя» — «Дмитрий»). В пределах этой подгруппы (и только в ней!) выполняется сортировка записей по значениям третьего заданного поля «Отчество»:

Фамилия	Имя	Отчество
Александров	Николай	Иванович
Бунин	Андрей	Карлович
. . .		
Иванов	Дмитрий	*Алексеевич*
Иванов	Дмитрий	*Николаевич*
Иванов	Устин	Сергеевич
. . .		
Яхин	Роман	Константинович

База данных отсортирована.

Если на каком-то этапе сортировки *не образуется групп/подгрупп* записей с одинаковыми значениями в поле, по которому выполнен этот этап сортировки, то операцию сортировки можно прервать досрочно (не рассматривать последующие поля запроса на сортировку).

3. Поиск в многотабличной БД

В реляционной БД, состоящей из нескольких взаимосвязанных таблиц, поиск выполняется с учётом переходов от одной таблицы к другой в соответствии со связями между их полями. Такой переход может быть многократным и выполняться в обоих направлениях.

Ручной поиск объекта в многотабличной БД

Пусть задана реляционная БД, состоящая из двух таблиц (под «ID» подразумевается уникальный индивидуальный номер человека, выступающий в качестве ключевого поля):

1) «ФИО»–«ID»–«пол»;

2) «ID родителя»–«ID сына/дочери».

Требуется найти племянника (племянников) заданного человека.

Процесс поиска:

1) в таблице 1 ищется ФИО заданного человека и определяется его ID;

2) *переход в таблицу 2*;

3) племянник — сын брата/сестры; брат/сестра — сын/дочь того же самого родителя; поэтому в таблице 2 для ID исходного человека, найденного в поле «ID сына/дочери», ищутся все ID его родителей;

4) для этих ID родителей, найденных в поле «ID родителя», ищутся все ID детей; при этом ID исходного человека из поиска исключается как уже рассмотренный;

5) для этих ID братьев/сестёр, найденных в поле «ID родителя», ищутся все ID их детей;

6) *возврат в таблицу 1*;

7) для найденных ID детей (это племянники и племянницы исходного человека) в таблице 1 ищутся их ФИО; среди них выбираются те, которые соответствуют людям мужского пола (определяется по полю «пол»).

Примеры такого поиска рассмотрены при разборе решений типовых задач.

Разбор типовых задач

Задача 1*. Ниже приведены фрагменты таблиц базы данных участников конкурса исполнительского мастерства:

Страна	Участник
Германия	Силин
США	Клеменс
Россия	Холево
Грузия	Яшвили
Германия	Бергер
Украина	Численко
Германия	Феер
Россия	Каладзе
Германия	Альбрехт

Участник	Инструмент	Автор произведения
Альбрехт	флейта	Моцарт
Бергер	скрипка	Паганини
Каладзе	скрипка	Паганини
Клеменс	фортепиано	Бах
Силин	скрипка	Моцарт
Феер	флейта	Бах
Холево	скрипка	Моцарт
Численко	фортепиано	Моцарт
Яшвили	флейта	Моцарт

Представители скольких стран исполняют Моцарта?

1) 5 2) 2 3) 3 4) 4

Решение

Подобные задачи решаются достаточно просто, — нужно только определить, с какой таблицы начать просмотр данных и как таблицы связаны между собой.

В данном случае требуется определить, представители скольких стран исполняли произведения Моцарта.

Указание авторов произведений имеется во второй таблице, откуда можно определить, что Моцарта исполняли следующие 5 участников:

Участник	Инструмент	Автор произведения
Альбрехт	флейта	Моцарт
Бергер	скрипка	Паганини
Каладзе	скрипка	Паганини
Клеменс	фортепиано	Бах
Силин	скрипка	Моцарт
Феер	флейта	Бах
Холево	скрипка	Моцарт
Численко	фортепиано	Моцарт
Яшвили	флейта	Моцарт

Далее надо определить, к каким странам относятся эти участники. Данная информация имеется в первой таблице, в которой выбираются строки с ранее найденными фамилиями участников:

Страна	Участник
Германия	Силин
США	Клеменс
Россия	Холево
Грузия	Яшвили
Германия	Бергер
Украина	Численко
Германия	Феер
Россия	Каладзе
Германия	Альбрехт

Поскольку из Германии было двое из выбранных участников, количество стран, представителями которых являются участники-исполнители музыки Моцарта, равно 4.

Ответ: 4 страны (вариант ответа № 4).

Задача 2*. База данных о торговых операциях дистрибутора состоит из трёх связанных таблиц. Ниже даны фрагменты этих таблиц.

Таблица зарегистрированных дилеров

Наименование организации	ID дилера	Регион	Адрес
ООО «Вектор»	D01	Башкортостан	г. Уфа, ул. Школьная, 15
АО «Луч»	D02	Татарстан	г. Казань, ул. Прямая, 17
АОЗТ «Прямая»	D03	Адыгея	г. Майкоп, просп. Мира, 8
ООО «Окружность»	D04	Дагестан	г. Дербент, ул. Замковая, 6
ИЧП Скаляр	D05	Дагестан	г. Махачкала, ул. Широкая, 28
АО «Ромб»	D06	Татарстан	г. Набережные Челны, ул. Заводская, 4

Таблица отгрузки товара

Номер накладной	Отгружено дилеру	Артикул товара	Отгружено упаковок	Дата отгрузки
001	D01	01002	300	5/01/2009 г.
002	D02	01002	100	5/01/2009 г.
003	D06	01002	200	5/01/2009 г.
004	D01	02002	20	5/01/2009 г.
005	D02	02002	30	5/01/2009 г.
006	D02	01003	20	6/01/2009 г.

Таблица товаров

Наименование товара	Артикул	Отдел	Количество единиц в упаковке	Брутто вес упаковки
Фломастеры, пачка 24 шт.	01001	Канцтовары	24	5
Бумага А4. пачка 500 листов	01002	Канцтовары	5	10
Скрепки металлические 1000 шт.	01003	Канцтовары	48	20
Розетки трёхфазные	02001	Электротовары	12	2
Лампа накаливания 60 Вт	02002	Электротовары	100	8
Выключатель 2-клавишный	02003	Электротовары	48	7

Сколько пачек бумаги было отгружено в Татарстан 5 января 2009 г.?

1) 100
2) 200
3) 500
4) 1500

Решение

1. Требуется определить отгрузку пачек бумаги — ищется этот вид товара в таблице 3:

Наименование товара	Артикул	Отдел	Количество единиц в упаковке	Брутто вес упаковки
Бумага А4. пачка 500 листов	01002	Канцтовары	5	10

Полем для связи этой таблицы с таблицей 2 является поле «Артикул», поэтому запоминается, что его значение равно **01002.**

2. Сведения о республиках и об отгрузке в Татарстан содержатся в таблице 1, где Татарстану соответствуют две строки:

Наименование организации	ID дилера	Регион	Адрес
АО «Луч»	D02	Татарстан	г. Казань, ул. Прямая, 17
АО «Ромб»	D06	Татарстан	г. Набережные Челны, ул. Заводская, 4

Здесь полем для связи с таблицей 2 является поле «ID дилера» и его значения, равные **D02** и **D06**.

3. В таблице 2 выбираются строки, в которых ***одновременно*** ID дилера и артикул товара равны найденным ранее значениям, и при этом дата отгрузки соответствует 5 января 2009 г.:

Номер накладной	Отгружено дилеру	Артикул товара	Отгружено упаковок	Дата отгрузки
001	D01	01002	300	5/01/2009 г.
002	**D02**	**01002**	**100**	**5/01/2009 г.**
003	**D06**	**01002**	**200**	**5/01/2009 г.**
004	D01	02002	20	5/01/2009 г.
005	D02	02002	30	5/01/2009 г.
006	D02	01003	20	6/01/2009 г.

Все три этих условия соблюдены только в двух строках таблицы. Суммируя указанные в них количества упаковок, получается, что заданному условию соответствуют 100 + 200 = 300 упаковок бумаги. Чтобы узнать, сколько отгружено пачек бумаги, найденное количество упаковок надо умножить на 5 (см. таблицу 3, где указано, что для бумаги количество единиц в упаковке равно 5). Итого: 300 · 5 = 1500 пачек бумаги.

Ответ: 1500 пачек бумаги (вариант ответа №4).

Задача 3*. В фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию и инициалы бабушки Ивановой А.И.

1) Иванов Т.М.

2) Черных И.А.

3) Цейс Т.Н.

4) Петренко Н.Н.

Таблица 1

ID	Фамилия_И.О.	Пол
71	Иванов Т.М.	М
85	Петренко И.Т.	М
13	Черных И.А.	Ж
42	Петренко А.И.	Ж
23	Иванова А.И.	Ж
96	Петренко Н.Н.	Ж
82	Черных А.Н.	М
95	Цейс Т.Н.	Ж
10	Цейс Н.А.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

Решение

Первое, на что следует обратить внимание: каждому человеку, чьи ФИО и пол отражены в базе данных в таблице 1, присвоен конкретный индивидуальный номер.

Вторая особенность: таблица 2 определяет родственные связи между людьми (закодированными их индивидуальными номерами) по принципу «родитель — ребёнок».

Исходное лицо: *Иванова А.И.*

1) Ищется запись о ней в табл. 1 (проверяя, что значение поля «Пол» для неё должно быть равно «Ж») и определяется её индивидуальный номер: **23**.

Таблица 1

ID	Фамилия_И.О.	Пол
71	Иванов Т.М.	М
85	Петренко И.Т.	М
13	Черных И.А.	Ж
42	Петренко А.И.	Ж
23	Иванова А.И.	Ж
96	Петренко Н.Н.	Ж
82	Черных А.Н.	М
95	Цейс Т.Н.	Ж
10	Цейс Н.А.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

2) По таблице 2 определяется, кто является родителем человека с найденным индивидуальным номером 23. Для этого ищутся в таблице 2 все строки с номером 23 в поле «ID_Ребёнка» и определяется, какие индивидуальные номера соответствуют в найденных записях полю «ID_Родителя». Это номера **13** и **85**: очевидно, мать и отец Ивановой А.И. (Вернувшись к таблице 1, по найденным номерам можно определить по значению поля «Пол», кто есть кто, хотя для решения данной задачи это не требуется.)

Таблица 1

ID	Фамилия_И.О.	Пол
71	Иванов Т.М.	М
85	Петренко И.Т.	М
13	Черных И.А.	Ж
42	Петренко А.И.	Ж
23	Иванова А.И.	Ж
96	Петренко Н.Н.	Ж
82	Черных А.Н.	М
95	Цейс Т.Н.	Ж
10	Цейс Н.А.	М
...		

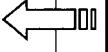


Таблица 2

ID_Родителя	ID_Ребёнка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

3) Родители Ивановой А.И. определены. Но по условию задачи требуется найти, кто является её бабушкой. Бабушка — это (в контексте рассматриваемого задания) мать одного из родителей Ивановой А.И., поэтому вышеописанную операцию поиска ID родителей по ID детей надо повторить для найденных на предыдущем шаге матери и отца Ивановой А.И.

- В таблице 2 в поле «ID_Ребёнка» ищется индивидуальный номер 85. Поскольку такое значение в указанном поле отсутствует, поиск по данной ветви генеалогического древа можно прекратить.
- В таблице 2 в поле «ID_Ребёнка» ищется индивидуальный номер 13 и определяются все значения индивидуальных номеров в поле «ID_Родителя», соответствующих указанному значению поля «ID_Ребёнка». Это номера **82** и **95** (мать и отец матери Ивановой А.И., т.е. её дед и искомая бабушка).

Таблица 1

ID	Фамилия_И.О.	Пол
71	Иванов Т.М.	М
85	Петренко И.Т.	М
13	Черных И.А.	Ж
42	Петренко А.И.	Ж
23	Иванова А.И.	Ж
96	Петренко Н.Н.	Ж
82	Черных А.Н.	М
95	Цейс Т.Н.	Ж
10	Цейс Н.А.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

4) Найдя ID деда и бабушки, в таблице 1 по этим номерам и значениям поля «Пол» определяется, кто из них кто.

Таблица 1

ID	Фамилия_И.О.	Пол
71	Иванов Т.М.	М
85	Петренко И.Т.	М
13	Черных И.А.	Ж
42	Петренко А.И.	Ж
23	Иванова А.И.	Ж
96	Петренко Н.Н.	Ж
82	Черных А.Н.	М
95	Цейс Т.Н.	Ж
10	Цейс Н.А.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
23	71
13	23
85	23
82	13
95	13
85	42
82	10
95	10
...	...

Очевидно, что бабушкой Ивановой А.И. является *Цейс Т.Н.*

Ответ: Цейс Т.Н. (вариант ответа №3).

Задача 4. В фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных, фамилию и инициалы племянника Черных Н.И.

Примечание: племянник — сын сестры или брата.

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	Черных Н.И.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
...	...

1) Петров А.Т.
2) Петров Т.М.
3) Гуревич А.И.
4) Гуревич И.Т.

Решение

Исходное лицо: *Черных Н.И.*

1) По таблице 1 определяется уникальный идентификационный номер Черных Н.И. Он равен **10**.

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	Черных Н.И.	М
...		

Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
...	...

2) В таблице 2 отражены родственные связи только типа «родитель — ребёнок», а по условию задачи необходимо выявить родственную связь «сын сестры / брата».

Сестра или брат являются детьми тех же самых родителей, что и интересующего лица. Поэтому, зная ID исходного человека (10), в таблице 2 определяются все строки, для которых в поле «ID_Ребёнка» записан код 10. Такая запись в таблице 2 одна и содержит в поле «ID_Родителя» идентификационный номер **85**. (Вернувшись к таблицу 1, можно по этому идентификатору определить, что речь идёт об отце Черных Н.И. — Гуревиче И.Т.)

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	**Черных Н.И.**	М
…		

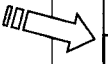


Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
…	…

3) Теперь (по таблице 2) определяются для найденного ID родителя идентификационные номера ***всех*** его детей. Для этого в таблице 2 пишутся все записи (строки), где в поле «ID_Родителя» записан код 85. Соответствующие ID детей равны: **13, 82** и **10**.

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	**Черных Н.И.**	М
…		

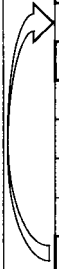


Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
…	…

4) По таблице 1 для найденных идентификаторов определяется информация о людях (ФИО и пол):

код 13 — Цейс И.И. (пол «Ж»);

код 82 — Гуревич А.И. (пол «М»);

код 10 — Черных Н.И. (пол «М»).

Из группы найденных людей Черных Н.И. — исходное лицо. Оно исключается из рассмотрения.

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	**Черных Н.И.**	М
…		

Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
…	…

5) Брат и сестра Черных Н.И. определены. Теперь нужно найти племянника Черных Н.И., зная, что это сын его брата или сестры. Для этого в таблице 2 ищутся записи (строки), где в поле «ID_Родителя» записано значение 13 или 82. (Записи для кода 82 в таблице отсутствуют, что облегчает задачу поиска.) Таких записей две, в них в поле «ID_Ребёнка» записаны значения кодов **42** и **23**.

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	Цейс А.Т.	Ж
71	Петров Т.М.	М
23	Петров А.Т.	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	Черных Н.И.	М
…		

Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
…	…

6) Вернувшись к таблице 1, для найденных кодов определяются ФИО и пол соответствующих лиц:

- код 42 — Цейс А.Т. (пол «Ж»);
- код 23 — Петров А.Т. (пол «М»).

Таблица 1

ID	Фамилия_И.О.	Пол
85	Гуревич И.Т.	М
82	Гуревич А.И.	М
42	**Цейс А.Т.**	Ж
71	Петров Т.М.	М
23	**Петров А.Т.**	М
13	Цейс И.И.	Ж
95	Черных Т.Н.	Ж
10	Черных Н.И.	М
…		

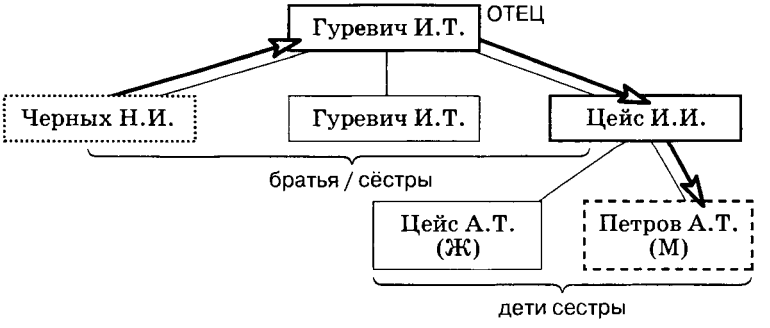
Таблица 2

ID_Родителя	ID_Ребёнка
95	82
85	13
71	42
85	82
13	42
71	23
13	23
95	13
85	10
…	…

Из указанных лиц Цейс А.Т. не подходит, поскольку по условию задачи нужно было определить племянника, а не племянницу. Поэтому единственный человек, подходящий в качестве решения задачи, — это Петров А.Т.

Ответ: Петров А.Т. (вариант ответа №1).

Примечание. Для наглядности можно изобразить «генеалогическое древо» всех рассмотренных в задаче лиц с указанием стрелками пути поиска племянника.



Задача 5*. Путешественник пришёл в 08:00 на автостанцию населённого пункта КАЛИНИНО и обнаружил следующее расписание автобусов:

Пункт отправления	Пункт прибытия	Время отправления	Время прибытия
КАМЫШИ	КАЛИНИНО	08:15	09:10
КАЛИНИНО	БУКОВОЕ	09:10	10:15
РАКИТІІНО	КАМЫШИ	10:00	11:10
РАКИТІІНО	КАЛИНИНО	10:05	12:25
РАКИТІІНО	БУКОВОЕ	10:10	11:15
КАЛИНИНО	РАКИТИНО	10:15	12:35
КАЛИНИНО	КАМЫШИ	10:20	11:15
БУКОВОЕ	КАЛИНИНО	10:35	11:40
КАМЫШИ	РАКИТИНО	11:25	12:30
БУКОВОЕ	РАКИТИНО	11:40	12:40

Определите самое раннее время, когда путешественник сможет оказаться в пункте РАКИТИНО согласно этому расписанию.

1) 12:25 2) 12:30 3) 12:35 4) 12:40

Решение

1. Исходный пункт — КАЛИНИНО. В таблице определяются и выделяются все строки, в которых в графе «Пункт отправления» записано «КАЛИНИНО»:

Пункт отправления	Пункт прибытия	Время отправления	Время прибытия
КАМЫШИ	КАЛИНИНО	08:15	09:10
КАЛИНИНО	**БУКОВОЕ**	**09:10**	**10:15**
РАКИТИНО	КАМЫШИ	10:00	11:10
РАКИТИНО	КАЛИНИНО	10:05	12:25
РАКИТИНО	БУКОВОЕ	10:10	11:15
КАЛИНИНО	**РАКИТИНО**	**10:15**	**12:35**
КАЛИНИНО	**КАМЫШИ**	**10:20**	**11:15**
БУКОВОЕ	КАЛИНИНО	10:35	11:40
КАМЫШИ	РАКИТИНО	11:25	12:30
БУКОВОЕ	РАКИТИНО	11:40	12:40

2. Анализируя первую найденную строку:

КАЛИНИНО	БУКОВОЕ	09:10	10:15

получается, что конечным пунктом является БУКОВОЕ. В таблице определяются строки, где пунктом отправления является БУКОВОЕ:

БУКОВОЕ	КАЛИНИНО	10:35	11:40
БУКОВОЕ	РАКИТИНО	11:40	12:40

Строка «БУКОВОЕ — КАЛИНИНО» не подходит, поскольку это — обратный путь. Вторая же строка соответствует требуемому пункту назначения.

Анализируя времена отправления/прибытия:

- «КАЛИНИНО — БУКОВОЕ» — 09:10 — **10:15**;
- «БУКОВОЕ — РАКИТИНО» — **11:40** — 12:40,

получается, что время отправления из БУКОВОЕ позже, чем время прибытия в БУКОВОЕ, поэтому данный маршрут возможен. Время прибытия в конечный пункт маршрута при этом равно 12:40.

3. Анализируя вторую найденную строку:

КАЛИНИНО	РАКИТИНО	10:15	12:35

в данном случае маршрут сразу содержит требуемый пункт назначения (РАКИТИНО). Время прибытия в этот конечный пункт — 12:35.

4. Анализируя третью найденную строку:

КАЛИНИНО	КАМЫШИ	10:20	11:15

конечным пунктом является КАМЫШИ. В таблице определяются строки, где пунктом отправления является КАМЫШИ:

Пункт отправления	Пункт прибытия	Время отправления	Время прибытия
КАМЫШИ	КАЛИНИНО	08:15	09:10
КАМЫШИ	РАКИТИНО	11:25	12:30

Строка «КАМЫШИ — КАЛИНИНО» не подходит (это обратный путь). Вторая строка соответствует требуемому пункту назначения.

Анализируя времена отправления/прибытия:

- «КАЛИНИНО — КАМЫШИ» — 10:20 — **11:15**;
- «КАМЫШИ — РАКИТИНО» — **11:25** — 12:30.

Время отправления из КАМЫШИ позже, чем время прибытия в КАМЫШИ, поэтому данный маршрут возможен. Время прибытия в конечный пункт маршрута при этом равно 12:30.

5. Таким образом, имеются три возможных маршрута из пункта КАЛИНИНО в пункт РАКИТИНО (один прямой и два — с пересадкой). Соответствующее им время прибытия в пункт РАКИТИНО:

- «КАЛИНИНО — БУКОВОЕ — РАКИТИНО» — 12:40;
- «КАЛИНИНО — РАКИТИНО» — 12:35;
- «КАЛИНИНО — КАМЫШИ — РАКИТИНО» — 12:30.

Самое раннее возможное время прибытия в пункт РАКИТИНО — 12:30.

Прямой маршрут — не обязательно более быстрый (по времени прибытия в конечный пункт), чем маршруты с пересадками.

Ответ: 12:30 (вариант ответа №2).

Возможны случаи, когда время ***прибытия*** в промежуточный пункт маршрута ***позже***, чем время ***отправления*** из него (т.е. путешественник не может пересесть на другой маршрут в течение текущих суток). Такие варианты отбрасываются как невозможные. Однако составители заданий ЕГЭ могут усложнить их. Если обнаруживаются подобные «нестыковки по времени» во ***всех*** возможных маршрутах, то необходимо не отбрасывать такие варианты, а анализировать их, учитывая, что путешественник, не успевший пересесть на другой маршрут сразу же, может сделать это на следующий день (т.е. в расчёты надо включать эту почти суточную задержку).

B12 Поиск информации в сети Интернет. Поисковые запросы

Конспект

Поисковый запрос для поисковой системы в Интернете представляет собой ключевое слово или несколько ключевых слов, соединенных между собой знаками логических операций **И, ИЛИ, НЕ**[1].

Процесс поиска в поисковой системе на основании поискового запроса аналогичен выборке данных в БД в соответствии с заданным условием отбора:

- если задано только одно ключевое слово, то производится поиск (выборка и включение в список найденного) всех web-страниц, в которых содержится данное ключевое слово;
- если ключевое слово задано с операцией **НЕ**, то производится поиск всех web-страниц, в которых *не содержится* данное ключевое слово;
- если ключевые слова связаны логической операцией **И**, то производится поиск web-страниц, в которых содержатся *все* эти ключевые слова;
- если ключевые слова связаны логической операцией **ИЛИ**, то производится поиск web-страниц, в которых содержится *хотя бы одно* ключевое слово.

Ранжирование поисковых запросов

Для разных поисковых запросов (содержащих различное число ключевых слов, связанных разными логическими операциями) количество найденных страниц будет разным. При этом важно помнить простые правила:

- **операция «И» (&, ∧) *сокращает*** объём получаемого при поиске результата (уменьшает количество найденных сайтов), причём чем *больше* в ней задействовано *операндов*, тем *меньше* будет *объём* получаемого списка найденных сайтов;

[1] Язык поисковых запросов, реализованный на почтовых сервисах (Яндекс, Google и пр.), включает также ряд других операций с ключевыми словами, позволяющих осуществлять поиск более гибко.

- **операция «ИЛИ» (|, ∨)** ***увеличивает*** объём получаемого при поиске результата (увеличивает количество найденных сайтов), причём чем *больше* в ней задействовано *операндов*, тем *больше* будет *объём* получаемого списка найденных сайтов.

Запросы, состоящие из одного-единственного операнда (ключевого слова) и не содержащие логических операций, можно рассматривать как запросы с операцией «ИЛИ» и с самым маленьким количеством операндов, т.е. в ранжированном списке такой запрос располагается непосредственно перед всеми остальными «ИЛИ»-запросами.

Причины этого очевидны: если операнды объединены при помощи операции «И», то найденные документы (web-страницы) должны содержать *все* эти операнды (ключевые слова). Тогда, например, для запроса *операнд1 & операнд2 & операнд3* получается следующий результат:

Документ содержит:			**Документ найден?**
операнд 1	**операнд 2**	**операнд 3**	
			нет
+			**нет**
	+		**нет**
		+	**нет**
+	+		**нет**
	+	+	**нет**
+		+	**нет**
+	+	+	**да**

Если же операнды объединены при помощи операции «ИЛИ», то найденные документы (web-страницы) могут содержать *хотя бы один* из этих операндов (ключевых слов). Тогда для запроса *операнд1* | *операнд2* | *операнд3* результат будет таким:

Документ содержит:			Документ найден?
операнд 1	**операнд 2**	**операнд 3**	
			нет
+			да
	+		да
		+	да
+	+		да
	+	+	да
+		+	да
+	+	+	да

Таким образом, для поисковых запросов, включающих в себя только один вид логических операций (только «И» или только «ИЛИ») можно сразу определить их место в формируемом списке ранжирования. Если в задаче требуется расположить запросы по возрастанию[1] количества найденных документов, то:

- запросы с операцией «И» будут располагаться в начале списка, и чем больше в них задействовано операндов, тем такие запросы ближе к началу списка;

[1] Если требуется, наоборот, выстроить поисковые запросы по порядку ***уменьшения*** количества найденных документов, то задача решается аналогично, но расположение запросов в списке будет обратным.

- запросы с операцией «ИЛИ» будут располагаться в конце списка, и чем больше в них задействовано операндов, тем такие запросы ближе к концу списка;
- запрос из одного-единственного ключевого слова будет располагаться в списке непосредственно перед запросами с операцией «ИЛИ».

Более сложен случай, когда поисковые запросы содержат оба типа логических операций — и «И», и «ИЛИ».

Иногда можно было получить правильный ответ просто методом исключения: такой смешанный запрос располагается в ранжированном списке где-то посередине — между расположенным в его начале блоком запросов с «И» и расположенным в конце блоком запросов с «ИЛИ».

Возможны задачи, в которых смешанных запросов будет предложено несколько. В этом случае надо немного порассуждать, как ранжировать такие смешанные запросы между собой.

Пример. Имеются два смешанных запроса:

(кошки & собаки) | кролики

и

(кошки | собаки) & кролики

В первом случае будут найдены документы, в которых есть *оба* слова — «кошки» и «собаки», и к ним будут добавлены *все* документы со словом «кролики».

Во втором случае будут найдены документы, содержащие или слово «кошки», или слово «собаки», а из них будут отобраны только те документы, которые содержат также слово «кролики».

(Кошки & Собаки) | Кролики

(Кошки | Собаки) & Кролики

Можно считать, что «действие» (уменьшение или увеличение количества найденных документов) логических операций «И» и «ИЛИ» «ослабляется», когда они стоят в скобках, и «усиливается», когда эти операции расположены вне скобок. То есть когда операция «И» стоит в скобках, а «ИЛИ» — вне скобок, будет найдено больше документов, чем когда операция «ИЛИ» стоит в скобках, а «И» — вне скобок.

Вычисление количества найденных страниц

В подобных задачах считается, что существует некоторое ограниченное количество web-документов, часть которых (в том числе все они либо ни один из них) может быть найдена при помощи определённого поискового запроса. В задаче рассматривается набор запросов, включающих одни и те же ключевые слова (полный или неполный набор) и связанных различными логическими операциями. Для этих запросов указаны количества найденных документов, а по одному из запросов это количество требуется определить.

В этом случае найденные по каждому элементарному запросу (по какому-то одному ключевому слову) web-документы рассматриваются как пересекающиеся (операция **И**) либо объединяемые (операция **ИЛИ**) множества, а количества найденных документов вычисляются как объёмы этих множеств и/или их подмножеств.

Пример:

В таблице приведено количество страниц, которое поисковая система находит по каждому запросу.

Запрос	Количество найденных страниц
Кошки | Собаки | Кролики	2600
Кошки	1300
Собаки	800
Кролики & Кошки	300
Собаки & Кошки	200
Кролики & Собаки	200
Кролики & Собаки & Кошки	100

Требуется определить, какое количество страниц будет найдено по запросу «Кролики»?

Совокупность запросов можно представить в виде диаграммы Эйлера-Венна (ключевым словам соответствуют пересекающиеся круги), а получившиеся при этом области нумеруются. Для нумерации используются цифры в кружках.

Области, соответствующие отдельным ключевым словам (кругам), а также области попарных пересечений кругов, соответствующие составным запросам с операцией И, состоят из нескольких пронумерованных областей. Например, круг «Кошки» — это объединение областей ❶, ❷, ❹ и ❺, а пересечение кругов «Кошки» и «Собаки» — это объединение областей ❷ и ❹. Аналогично, составным запросам с операцией ИЛИ соответствуют объединения пронумерованных областей, относящихся ко всем использованным в запросе ключевым словам. Например, запросу «Кошки | Собаки» соответствует объединение областей ❶, ❷, ❸, ❹, ❺ и ❻ (причём каждая пронумерованная область берётся только по одному разу).

Такие объединения пронумерованных областей записываются как суммы соответствующих порядковых номеров (цифр в кружках).

Далее для выполнения расчётов количеств запросов номера областей (цифры в кружочках) рассматриваются как своеобразные переменные, значения которых равны количествам запросов, приходящихся на каждую такую область. Тогда приведённую в условии таблицу запросов можно преобразовать в систему уравнений:

Запрос	Уравнение	Комментарий
Кошки | Собаки | Кролики	❶ + ❷ + ❸ + ❹ + + ❺ + ❻ + ❼ = 2600	Все области, соответствующие всем трём кругам
Кошки	❶ + ❷ + ❹ + ❺ = = 1300	Области, соответствующие кругу «Кошки»
Собаки	❷ + ❸ + ❹ + ❻ = = 800	Области, соответствующие кругу «Собаки»
Кролики & Кошки	❹ + ❺ = 300	Области, соответствующие пересечению кругов «Кошки» и «Кролики»
Собаки & Кошки	❷ + ❹ = 200	Области, соответствующие пересечению кругов «Кошки» и «Собаки»
Кролики & Собаки	❹ + ❻ = 200	Области, соответствующие пересечению кругов «Собаки» и «Кролики»
Кролики & Собаки & Кошки	❹ = 100	Область пересечения всех трёх кругов (в центре диаграммы)

Соответственно, искомый запрос «Кролики» соответствует сумме: ❹ + ❺ + ❻ + ❼ (области, соответствующие кругу «Кролики»).

Решение задачи сводится к решению получившейся системы уравнений путём подстановок известных значений переменных и/или их сумм. Как правило, не обязательно доводить решение системы до вычисления значений всех переменных по отдельности, — достаточно подстановками заменить числами все переменные в искомой сумме, а затем произвести её вычисление.

В данном случае решение может быть таким:

$$\begin{cases} ❶ + ❷ + ❸ + ❹ + ❺ + ❻ + ❼ = 2600 \\ ❶ + ❷ + ❹ + ❺ = 1300 \\ ❷ + ❸ + ❹ + ❻ = 800 \\ ❹ + ❺ = 300 \\ ❷ + ❹ = 200 \\ ❹ + ❻ = 200 \\ \boxed{❹ = 100} \end{cases}$$

Искомая сумма:
$❹ + ❺ + ❻ + ❼$

Во все уравнения подставляется известное значение ❹:

$$\begin{cases} ❶ + ❷ + ❸ + 100 + ❺ + ❻ + ❼ = 2600 \\ ❶ + ❷ + 100 + ❺ = 1300 \\ ❷ + ❸ + 100 + ❻ = 800 \\ 100 + ❺ = 300 \\ ❷ + 100 = 200 \\ 100 + ❻ = 200 \end{cases}$$

Искомая сумма:
$100 + ❺ + ❻ + ❼$

Вычисляются значения ❷, ❺ и ❻ и подставляются в остальные уравнения:

$$\begin{cases} ❶ + 100 + ❸ + 100 + 200 + 100 + ❼ = 2600 \\ ❶ + 100 + 100 + 200 = 1300 \\ 100 + ❸ + 100 + 100 = 800 \\ \boxed{❺ = 200} \\ \boxed{❷ = 100} \\ \boxed{❻ = 100} \end{cases}$$

Искомая сумма:
$100 + 200 + 100 + ❼$

Из второго и третьего уравнений определяются значения ❶ и ❸ и подставляются в первое уравнение:

$$\begin{cases} 900 + 100 + 500 + 100 + 200 + 100 + ❼ = 2600 \\ \boxed{❶ = 900} \\ \boxed{❸ = 500} \end{cases}$$

Искомая сумма:
$100 + 200 + 100 + ❼$

Из первого уравнения вычисляется значение ❼ и подставляется в запись искомой суммы:

$$\boxed{❼ = 700}$$

Искомая сумма: $100 + 200 + 100 + 700$

Таким образом, в искомой сумме все переменные заменены числовыми значениями:
100 + 200 + 100 + 700 = 1100.

Ответ: запросу «Кролики» соответствует 1100 найденных страниц.

Разбор типовых задач

Задача 1*. В таблице приведены запросы к поисковому серверу. Расположите обозначения запросов в порядке возрастания количества страниц, которые найдёт поисковый сервер по каждому запросу.

Для обозначения логической операции «ИЛИ» в запросе используется символ |, а для логической операции «И» — символ &.

1	канарейки | щеглы | содержание
2	канарейки & содержание
3	канарейки & щеглы & содержание
4	разведение & содержание & канарейки & щеглы

Решение

Запросы с операцией «И» в списке, ранжированном по возрастанию количества найденных документов, располагаются в самом начале, и чем больше в эти запросы включено операндов, тем они ближе к началу списка. Значит, список будет начинаться с запросов 4, 3 и 2 (именно в этом порядке).

Запросы с операцией «ИЛИ» располагаются в конце ранжированного списка, — значит, запрос 1 будет последним.

Смешанные по типу операций запросы (при их наличии) определяются по методу исключения и располагаются в списке посередине между запросами с «И» и запросами с «ИЛИ».

Ответ: 4321.

Задача 2*. В языке запросов поискового сервера для обозначения логической операции «ИЛИ» используется символ «|», а для логической операции «И» — символ «&».

В таблице приведены запросы и количество найденных по ним страниц некоторого сегмента сети Интернет.

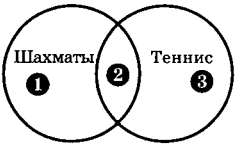
Запрос	Найдено страниц (в тысячах)
Шахматы | *Теннис*	7770
Теннис	5500
Шахматы & *Теннис*	1000

Какое количество страниц (в тысячах) будет найдено по запросу *Шахматы*?

Считается, что все запросы выполнялись практически одновременно, так что набор страниц, содержащих все искомые слова, не изменялся за время выполнения запросов.

Решение

Подобные задачи можно решать путём логических рассуждений. Однако существует универсальный способ их решения, который позволяет сделать решение всех таких задач типовым.



Строится примерная диаграмма Венна. «Примерная» — потому что в таких задачах *всегда* лучше сначала рисовать пересекающиеся области, хотя, пересечение этих областей может и отсутствовать. Все получившиеся при этом «элементарные» области нумеруются по порядку.

Теперь, считая порядковые номера выделенных областей своеобразными «переменными» (под значениями которых понимаются объёмы соответствующих множеств, т.е. количества найденных документов), для указанных в таблице в условии задачи запросов составляется система уравнений. При этом следует помнить, что область «Шахматы» состоит из двух выделенных «элементарных» областей ❶ и ❷, точно так же как область «Теннис» состоит из двух «элементарных» областей ❷ и ❸.

$$\begin{cases} ❶ + ❷ + ❸ = 7770; \\ ❷ = 1000; \\ ❷ + ❸ = 5500. \end{cases}$$

Аналогично, искомый запрос *Шахматы* при этом трансформируется в значение суммы «переменных» ❶ + ❷.

Для решения этой системы уравнений значение ❷ подставляется в оба других уравнения. Получается:

$$\begin{cases} ❶ + 1000 + ❸ = 7770; \\ 1000 + ❸ = 5500. \end{cases} \Rightarrow \begin{cases} ❶ + ❸ = 6770; \\ ❸ = 4500. \end{cases}$$

Подставляя теперь значение ❸ в первое уравнение: ❶ + 4500 = 6770 ⇒ ❶ = 2270.

Искомая сумма ❶ + ❷ = 2270 + 1000 = 3270.

Ответ: 3270.

Задача 3*. В языке запросов поискового сервера для обозначения логической операции «ИЛИ» используется символ «|», а для логической операции «И» — символ «&».

В таблице приведены запросы и количество найденных по ним страниц некоторого сегмента сети Интернет.

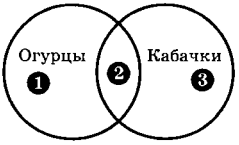
Какое количество страниц (в тысячах) будет найдено по запросу *Огурцы* ?

Считается, что все запросы выполнялись практически одновременно, так что набор страниц, содержащих все искомые слова, не изменялся за время выполнения запросов.

Запрос	Найдено страниц (в тысячах)
Огурцы | Кабачки	12000
Огурцы & Кабачки	6500
Кабачки	7700

Решение

Строится примерная диаграмма Венна (внешне она будет точно такой же, как и в предыдущей задаче):



Составляется система уравнений, соответствующих указанным в таблице запросам:

$$\begin{cases} ❶ + ❷ + ❸ = 12000; \\ ❷ = 6500; \\ ❷ + ❸ = 7700. \end{cases}$$

Выражение, соответствующее запросу *Огурцы*, имеет вид:

❶ + ❷.

Решается система уравнений. Для начала подставляется значение «переменной» ❷, известное по второму уравнению, в оба других уравнения системы, понижая тем самым её размерность до двух:

$$\begin{cases} ❶ + 6500 + ❸ = 12000; \\ 6500 + ❸ = 7700. \end{cases}$$

Из второго уравнения получившейся системы нетрудно получить значение «переменной» ❸: оно равно 7700 – 6500 = 1200.

Подставив его в первое уравнение, получается значение последней «переменной» — ❶:

❶ + 6500 + 1200 = 12000 ⇒ ❶ = 4300.

Тогда, запросу *Огурцы* соответствует:

❶ + ❷ = 4300 + 6500 = 10800 найденных документов.

Ответ: 10800.

Задача 4. Некоторый сегмент сети Интернет состоит из 1000 сайтов. Поисковый сервер в автоматическом режиме составил таблицу ключевых слов для сайтов этого сегмента. Вот её фрагмент:

Ключевое слово	**Количество сайтов, для которых данное слово является ключевым**
скутер	200
байк	250
мопед	450

Сколько сайтов будет найдено по запросу

(байк | скутер) & мопед

если по запросу *байк | скутер* было найдено 450 сайтов, по запросу *байк & мопед* — 40, а по запросу *скутер & мопед* — 50 сайтов?

Решение

В этом случае вид диаграммы Венна будет другим — она будет состоять из трёх кругов:

Составляется система уравнений для каждого заданного запроса (как в таблице, так и вне её), не забывая также добавить условие, что всего сайтов было 1000:

$$\begin{cases} \textit{Скутер}: & ❶ + ❷ + ❹ + ❺ = 200; \\ \textit{Байк}: & ❷ + ❸ + ❺ + ❻ = 250; \\ \textit{Мопед}: & ❹ + ❺ + ❻ + ❼ = 450; \\ \textit{Байк | скутер}: & ❶ + ❷ + ❸ + ❹ + ❺ + ❻ = 450; \\ \textit{Байк \& мопед}: & ❺ + ❻ = 40; \\ \textit{Скутер \& мопед}: & ❹ + ❺ = 50; \\ \textit{Всего сайтов}: & ❶ + ❷ + ❸ + ❹ + ❺ + ❻ + ❼ = 1000. \end{cases}$$

Запрос *(байк | скутер) & мопед* при этом соответствует выражению:

❹ + ❺ + ❻.

Решается система уравнений.

Из четвёртого уравнения вычитаются первые два:

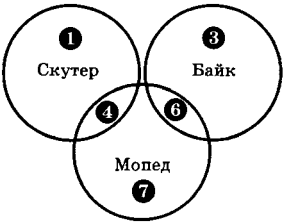
$$❶ + ❷ + ❸ + ❹ + ❺ + ❻ - (❶ + ❷ + ❹ + ❺) - (❷ + ❸ + ❺ + ❻) = 450 - 200 - 250 = 0 \Rightarrow$$
$$\Rightarrow - ❷ - ❺ = 0 \Rightarrow ❷ + ❺ = 0.$$

Значения наших «переменных», обозначенных цифрами в кружках, — это *количества найденных документов* (т.е. числа натурального ряда), *которые не*

могут быть отрицательными. Следовательно, если сумма таких «переменных» равна нулю, то *все* переменные, входящие в эту сумму (в данном случае ❷ и ❺), также обязаны равняться нулю. Следовательно:

❷ = 0; ❺ = 0.

Это является объяснением тому, что диаграмма Венна называется «примерной»: равенство нулю этих двух «переменных» означает, что запросу *байк & скутер* не соответствует ни один документ (сайт), т. е. диаграмма Венна на самом деле должна выглядеть так:



Складывается пятое и шестое уравнения, одновременно подставляя уже известное нулевое значение «переменной» ❺:

❺ + ❻ + ❹ + ❺ = 40 + 50 ⇒ ❹ + ❻ = 90.

С учётом нулевого значения «переменной» ❺ — это и есть значение искомого выражения для запроса *(байк | скутер) & мопед*.

Задача решена. При необходимости, «расплетая» систему уравнений дальше, можно было бы вычислить значения всех используемых «переменных» — от ❶ до ❼ — и определить количества найденных документов для *любого* запроса с указанными ключевыми словами.

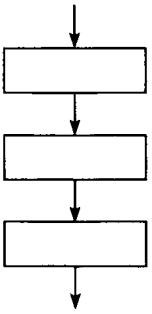
Ответ: 90.

B2 Условный оператор

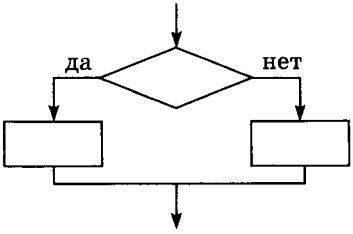
Конспект

Основные алгоритмические конструкции

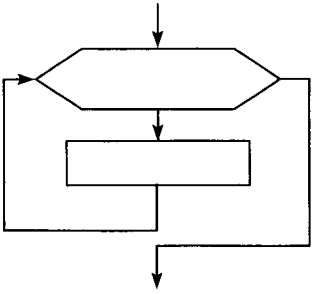
Линейный алгоритм



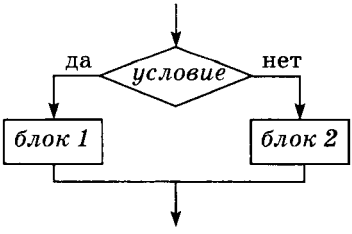
Алгоритм с ветвлением



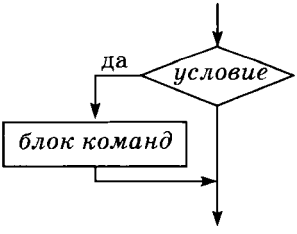
Циклический алгоритм



Полная конструкция «Ветвление»



Неполная конструкция «Ветвление»



Конструкция ветвления в языке Паскаль

Полная	Неполная

if (<условие>) then
begin
<блок 1>
end
else
begin
<блок 2>
end;
	if (<условие>) then
	begin
	<блок>
	end;

блок 1 — *блок 2* — *блок команд*

Если блоки команд состоят только из одного оператора каждый, то использовать операторные скобки `begin ... end` не обязательно:

Полная	Неполная

if (<условие>) then
<команда 1>
else
<команда 2>;
	if (<условие>) then
	<команда>;

В конструкции вложенных операторов `if`:

```
if (<условие 1>) then <команда 1>
   else if (<условие 2>) then <команда 2>
           else <команда 3>;
```

оператор `else` ***всегда*** относится к последнему по счёту оператору `if`. Поэтому, если условие 1 ложно, то весь второй оператор `if` не будет выполнен.

Во избежание путаницы рекомендуется записывать вложенные конструкции `if ... then ... else` в операторных скобках `begin ... end;`

В языке Паскаль завершающий знак «;» после содержимого ветви `then` (перед оператором `else`) ***не ставится***, так как вся конструкция `if ... then ... else` представляет собой один единый оператор, а знак «;» — это знак завершения оператора.

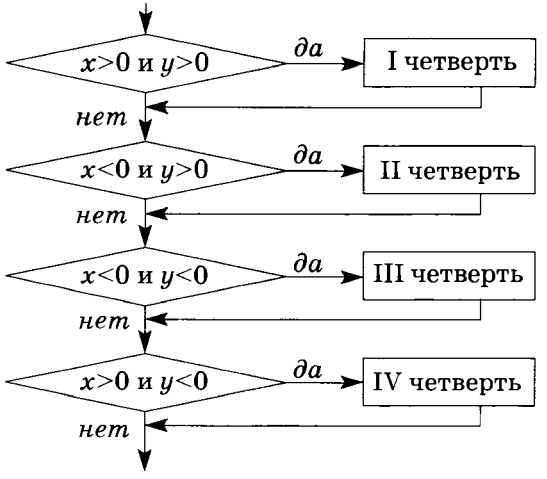
Полные и неполные условия в цепочке операторов `if`

Пусть требуется при помощи нескольких операторов `if` выделить все возможные варианты сочетаний условий (например, по заданным значениям x и y определить, в какой координатной четверти располагается точка с такими координатами). Тогда проверку соответствующих условий можно выполнять двумя способами.

1. Использование полных условий

В этом случае каждый оператор `if` (соответствующий одному из возможных сочетаний условий) содержит полный набор таких условий (операций сравнения), определяющий соответствующий вариант. В примере задачи с принадлежностью точки координатной плоскости это две операции сравнения для x и y. В этом случае каждый оператор `if` не зависит от других (так как его условие независимо от условий, определяющих другие варианты), и в программе все эти операторы `if` могут быть записаны в любом порядке друг за другом:

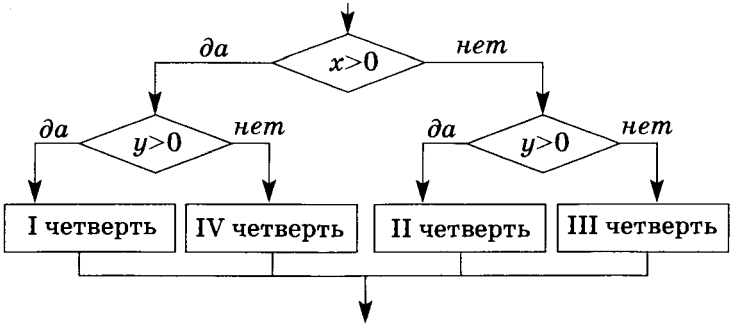
```
if (x>0) and (y>0) then writeln('I четверть');
if (x<0) and (y>0) then writeln('II четверть');
if (x<0) and (y<0) then writeln('III четверть');
if (x>0) and (y<0) then writeln('IV четверть');
```



2. Использование частичных условий

Частичное условие является неполным и определяет несколько возможных ситуаций (например, условие $x > 0$ указывает, что точка расположена в правой полуплоскости). Для определения конкретной ситуации нужно использовать ещё одно ветвление, проверяющее оставшиеся условия (дополняющие уже проверенное до полного набора условий), например, условие $y > 0$, проверяемое в ветви `then` оператора `if` с условием $x > 0$, однозначно определяет принадлежность точки I-й координатной четверти. При таком построении алгоритма необходимо использовать вложенные операторы конструкции `if ... then ... else`:

```
if (x > 0) then begin
     if (y > 0) then writeln('I четверть')
                else writeln('IV четверть')
                  end
              else begin
     if (y > 0) then writeln('II четверть')
                else writeln('III четверть')
                  end;
```



Использование меняющихся значений условий

Возможны задачи, в которых конкретные ситуации определяются одинаковыми условиями, но с разными значениями (константами). Например, если требуется определить сезон (зима, весна, лето, осень) по номеру

месяца, то это можно сделать, проверяя равенство заданного номера месяца числам:

- 1, 2, 12 — зима,
- 3, 4, 5 — весна,
- 6, 7, 8 — лето,
- 9, 10, 11 — осень,
- другие значения номера месяца — ошибка ввода.

Решение такой задачи также может осуществляться с использованием как полных, так и неполных условий.

1. Решение с полными условиями

В каждом операторе `if` можно задать условие, однозначно определяющее соответствующий диапазон значений номера месяца:

```
if (n = 1) or (n = 2) or (n = 12) then
    writeln('зима');
if (n >= 3) and (n <= 5) then writeln('весна');
if (n >= 6) and (n <= 8) then writeln('лето');
if (n >= 9) and (n <= 11) then writeln('осень');
if (n < 1) or (n > 12)  then writeln('ошибка
    ввода');
```

В данном случае в каждом операторе `if` соответствующее условие однозначно определяет ситуацию, поэтому последовательность записи этих операторов может быть любой.

2. Решение с неполными (меняющимися) условиями

Можно решить ту же задачу иначе, воспользовавшись фактом, что значение одной и той же переменной может быть многократно переприсвоено и при этом актуальным остаётся последнее присвоенное ей значение.

```
if (n < 1) or (n > 12) then s := 'ошибка ввода'
    else begin
      if (n >= 1) then s := 'зима';
      if (n >= 3) then s := 'весна';
      if (n >= 6) then s := 'лето';
      if (n >= 9) then s := 'осень';
      if (n = 12) then s := 'зима';
end;
writeln(s);
```

Работа этой программы для случая $n = 7$:

- первый оператор проверяет возможное наличие ошибки; значение $n = 7$ корректно, поэтому выполняется ветвь `else` (содержащая собственно операторы определения названия сезона);
- $n = 7$, поэтому условие $n >= 1$ выполняется; переменной s присваивается значение «зима»;
- условие $n >= 3$ также выполняется; переменной s взамен прежнего значения «зима» присваивается значение «весна»;
- условие $n >= 6$ также выполняется; переменной s взамен прежнего значения «весна» присваивается значение «лето»;
- условие $n >= 9$ уже не выполняется; оператор присваивания в ветви `then` пропускается, переменная s сохраняет значение «лето»;
- условие $n = 12$ также не выполняется; оператор присваивания в ветви `then` пропускается, переменная s сохраняет значение «лето».

В данном случае порядок записи операторов `if` важен! Если изменить его, то программа перестанет работать правильно. Например, изменив порядок записи операторов на противоположный:

```
if (n < 1) or (n > 12) then s := 'ошибка ввода'
   else begin
     if (n = 12) then s := 'зима';
     if (n >= 9) then s := 'осень';
     if (n >= 6) then s := 'лето';
     if (n >= 3) then s := 'весна';
     if (n >= 1) then s := 'зима';
   end;
writeln(s);
```

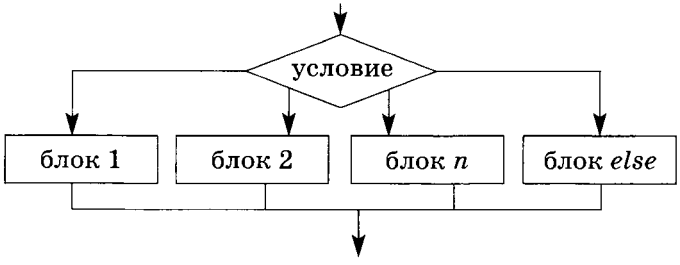
Результат работы такой программы (для $n = 7$):

- первый оператор проверяет возможное наличие ошибки; значение $n = 7$ корректно, поэтому выполняется ветвь `else` (содержащая собственно операторы определения названия сезона);
- $n = 7$, поэтому условие $n = 12$ не выполняется; оператор присваивания в ветви `then` пропускается;

- условие *n* >= 9 также не выполняется; оператор присваивания в ветви then пропускается;
- условие *n* >= 6 выполняется; переменной *s* присваивается значение «лето»;
- условие *n* >= 3 также выполняется; переменной *s* взамен прежнего значения «лето» присваивается значение «весна»;
- условие *n* >= 1 также выполняется; переменной *s* взамен прежнего значения «весна» присваивается значение «зима».

В итоге программа вместо правильного значения «лето» выдаст ошибочное «зима».

Конструкция множественного ветвления (конструкция выбора)



Конструкция выбора в языке Паскаль

```
case <ключ> of
  <селектор 1>: begin <блок 1> end;
  <селектор 2>: begin <блок 2> end;
  . . . . .
  <селектор n>: begin <блок n> end;
  else begin <блок else>
end;
```

Ключ — выражение или переменная, имеющие целое или символьное значение.

В качестве селектора допускаются: константы, списки констант (через запятую), интервалы (через знак ..) и символы.

Если одной и той же ситуации соответствует несколько различных значений ключа, то эти значения можно записать через запятую в одной программной строке. Например, программа для определения количества дней в месяце по его номеру *n* может иметь вид:

```
case n of
  1, 3, 5, 7, 8, 10, 12 : d := 31;
  2 : d := 28;
  4, 6, 9, 11 : d := 30;
  else writeln('ошибка ввода n');
end;
```

Разбор типовых задач

Задача 1*. Определите значение переменной *c* после выполнения следующего фрагмента программы (записанного ниже на разных языках программирования):

Бейсик	Паскаль

a = 40
b = 10
b= -a / 2 * b
If a < b Then
  c = b - a
Else c = a - 2 * b
End If
	a := 40;
	b := 10;
	b := -a / 2 * b;
	if a < b then
	  c := b - a
	else
	  c := a - 2 * b;

Си	**Алгоритмический язык**

a = 40;
b = 10;
b = -a / 2 * b;
if (a < b)
  c = b - a;
else
c = a - 2 * b;
	a := 40
	b := 10
	b := -a / 2 * b
	если a < b
	  то c := b - a
	  иначе c := a - 2 * b
	все

Решение

Подобные задачи решаются путём выполнения ручной трассировки (покомандного выполнения программы) в табличной форме. При этом в таблицу включаются все переменные, значения которых изменяются в программе. Дополнительно в левом столбце могут быть записаны соответствующие операторы программы.

В строках таблицы трассировки, соответствующих условному оператору, желательно отдельно отмечать истинность или ложность условия ветвления.

В данном случае используются три переменные. Факт изменения значения той или иной переменной отмечен жирным подчёркнутым шрифтом и фоновым закрашиванием ячеек таблицы.

Оператор	*a*	*b*	*c*
`a := 40;`	**40**	–	–
`b := 10;`	40	**10**	–
`b := -a / 2 * b;`	40	**−200**	–
`if a < b then` *Условие не выполнено — выполняется ветвь* `else`	40	−200 	–
`else` `  c := a - 2 * b;`	40	−200	**440**

Ответ: $c = 440$.

 Внимание! При выполнении трассировки программ следует:

- до присваивания переменной первого значения содержимое этой переменной считать неопределённым и отмечать в таблице прочерком;
- при выполнении операторов присваивания, в которых одна и та же переменная стоит слева и справа нужно не ошибиться в определении, откуда берётся значение переменной для вычисления выражения после знака присваивания (из **предыдущей** строки таблицы) и в какой столбец записывать результат (в столбец, соответствующий переменной, которая записана **до знака присваивания**).

Задача 2*. Определите значение переменной *c* после выполнения следующего фрагмента программы, в котором *a*, *b* и *c* — переменные вещественного (действительного) типа.

Бейсик

```
a = 120
b = 100
a = a + b / 2
IF b < a / 2 THEN
    c = b + a
ELSE
    c = b + a / 2
ENDIF
```

Паскаль

```
a := 120;
b := 100;
a := a + b / 2;
if b < a / 2 then
  c := b + a
else
  c := b + a / 2;
```

Си

```
a = 120;
b = 100;
a = a + b / 2;
if (b < a / 2)
  c = b + a;
else
  c = b + a / 2;
```

Алгоритмический язык

```
a := 120
b := 100
a := a + b / 2
если b < a / 2
  то c := b + a
  иначе c := b + a / 2
все
```

1) $c = 105$ 3) $c = 185$
2) $c = 160$ 4) $c = 270$

Решение

Выполняется аналогично предыдущей задаче; в последней команде программы по полученным значениям a и b дополнительно вычисляется значение переменной c.

Оператор	a	b	c
`a := 120;`	**120**	–	–
`b := 100;`	120	**100**	–
`a := a + b / 2;`	**170**	100	–
`if b < a / 2 then` *Условие не выполнено — выполняется ветвь* `else`	(2*85) <	100	–
`else c := b + a / 2;`	170	100	**185**

Ответ: $c = 185$ (вариант ответа №3).

Задача 3*. Определите значение переменной *c* после выполнения следующего фрагмента программы (записанного ниже на разных языках программирования):

Бейсик	Паскаль

`a = 100`
`b = 30`
`a = a - b * 3`
`IF a > b THEN`
`  c = a - b`
`ELSE c = b - a`
`ENDIF`
	`a := 100;`
	`b := 30;`
	`a := a - b *3;`
	`if a > b then`
	`  c := a - b`
	`else c := b - a;`

Си	**Алгоритмический**

`a = 100;`
`b = 30;`
`a = a - b * 3;`
`if (a > b)`
`  c = a - b;`
`else c = b - a;`
	`a := 100`
	`b := 30`
	`a := a - b * 3`
	`если a > b то`
	`  c := a - b`
	`иначе c := b - a`
	`все`

1) $c = 20$ 3) $c = -20$
2) $c = 70$ 4) $c = 180$

Решение

Трассировка выполняется аналогично предыдущим задачам.

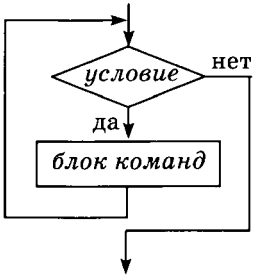
Оператор	*a*	*b*	*c*
`a := 100;`	**100**	–	–
`b := 30;`	100	**30**	–
`a := a - b *3;`	**10**	30	–
`if a > b then` *Условие не выполнено – выполняется ветвь* `else`	10	30 <	–
`else c := b - a;`	10	30	**20**

Ответ: $c = 20$ (вариант ответа №1).

Конспект

Цикл с предусловием (цикл ПОКА)

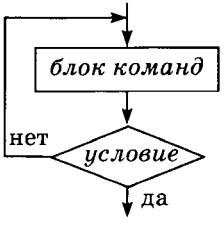
Обеспечивает выполнение блока команд, составляющего тело цикла, пока условие остается истинным (выход из цикла — по ложности условия).



Условие проверяется *до* начала выполнения цикла, поэтому возможна ситуация, что тело цикла не будет выполнено ни разу.

Цикл с постусловием (цикл ДО)

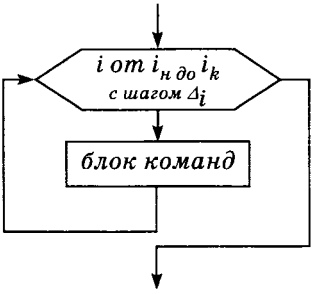
Обеспечивает выполнение команд, составляющих тело цикла, пока условие не станет истинным (цикл выполняется, пока условие ложно).



Условие проверяется *после* выполнения цикла, поэтому тело цикла всегда выполняется хотя бы один раз.

Цикл с параметром (цикл со счётчиком, цикл ДЛЯ)

Обеспечивает выполнение блока команд, составляющего тело цикла, количество раз, заданное начальным, конечным значениями переменной-счётчика (параметра цикла) и шагом её изменения.



Первоначально переменной-счётчику (в данном примере — переменной i) присваивается начальное значение $i_н$ и выполняется тело цикла. После выполнения тела цикла значение счётчика увеличивается на заданную величину шага Δi и выполняется проверка: не превысило ли значение счётчика заданное конечное значение $i_к$. Если не превысило, то вновь выполняется тело цикла. Иначе происходит выход из цикла.

Внимание!

В цикле с параметром количество выполнений тела цикла задано однозначно. В циклах же ПОКА и ДО количество выполнений тела цикла заранее не известно и определяется ситуацией. Поэтому необходимо проследить, чтобы во время выполнения циклов ПОКА и ДО происходило какое-то изменение переменных, задействованных в условии цикла (чтобы в какой-то момент истинность условия изменилась и цикл завершился). Несоблюдение этого требования приводит к бесконечному выполнению цикла («зацикливанию» программы).

Конструкции циклов в языке Паскаль

1. Цикл с предусловием:

```
while <условие> do
  begin
   <блок команд>
  end;
```

или

```
while <условие>
do <команда>;
```

Цикл с постусловием:

```
repeat
   <команды>
until <условие>;
```

Цикл со счётчиком:

Цикл с шагом 1	Цикл с шагом −1

for <i> := <$i_н$> to <$i_к$> do
  begin
  <блок команд>
  end;
или
for <i> := <$i_н$> to <$i_к$> do <команда>;
	for <i> := <$i_к$>
	downto <$i_н$> do
	  begin
	  <блок команд>
	  end;
	или
	for <i> := <$i_к$>
	downto <$i_н$> do
	<команда>;

Тело цикла может включать в себя другой цикл. «Глубина вложенности» циклов теоретически может быть любой.

При выполнении вложенных циклов:

- начинает выполняться внешний цикл:
  — выполняется тело внешнего цикла, в том числе целиком — внутренний цикл;
- проверяется условие внешнего цикла, если оно соблюдено, снова выполняется внешний цикл:
  — выполняется тело внешнего цикла, в том числе целиком — внутренний цикл;
- и т.д.

Правила построения вложенных циклов:

- циклы не должны «пересекаться», т.е. внутренний цикл должен начинаться и полностью завершаться внутри тела внешнего цикла;
- во внутреннем и во внешнем циклах ДЛЯ должны использоваться разные переменные-счётчики.

Операторы досрочного завершения цикла

Оператор языка Паскаль	Описание
`continue`	**Оператор продолжения** — выполнение данного оператора прекращает текущее выполнение тела цикла и передаёт управление на проверку условия цикла.
`break`	**Оператор прерывания** — выполнение данного оператора прекращает выполнение цикла и передаёт управление на следующий оператор после цикла.

Разбор типовых задач

Задача 1*. Определите, что будет напечатано в результате работы следующего фрагмента программы:

Бейсик

```
Dim k, s As Integer
s = 0
k = 0
While s < 1024
s = s + 10
k = k + 1
End While
Console.Write(k)
```

Паскаль

```
Var k, s : integer;
BEGIN
s := 0;
k := 0;
while s < 1024 do
begin
s := s + 10;
k := k + 1;
end;
write(k);
END.
```

Си

```
{
int к, s;
s = 0;
к = 0;
while (s < 1024) {
s = s + 10;
к = k + 1;
}
printf("%d", k); }
```

Алгоритмический

```
нач
цел к, s
s := 0
k := 0
нц пока s < 1024
s := s + 10; k := k + 1
кц
вывод k
кон
```

Решение

Строится таблица трассировки.

Операторы	*s*	*k*
`s := 0;` `k := 0;`	**0**	**0**
`while s < 1024 do` *Условие истинно — цикл выполняется*	0 ›	0
`s := s + 10;`	**10**	0
`k := k + 1;`	10	**1**
`while s < 1024 do` *Условие истинно — цикл выполняется*	10 ⇑	1
`s := s + 10;`	**20**	1
`k := k + 1;`	20	**2**
`while s < 1024 do` *Условие истинно — цикл выполняется*	20 ⇑	2
`s := s + 10;`	**30**	2
`k := k + 1;`	30	**3**
. . . . .		
`while s < 1024 do` *Условие истинно — цикл выполняется*	1000 ⇑	100
`s := s + 10;`	**1010**	100
`k := k + 1;`	1010	**101**
`while s < 1024 do` *Условие истинно — цикл выполняется*	1010 ⇑	101
`s := s + 10;`	**1020**	101
`k := k + 1;`	1020	**102**
`while s < 1024 do` *Условие истинно — цикл выполняется*	1020 ⇑	102
`s := s + 10;`	**1030**	102
`k := k + 1;`	1030	**103**
`while s < 1024 do` *Условие ложно — цикл прекращается*	1030 ⇑	103
`write(k);`		103

Таким образом, выводится значение переменной k, равное 103.

Ответ: $k = 103$.

Таблицу трассировки можно записывать упрощённо (подразумевая, что условие $s < 1024$ не выполняется, а при его первом же выполнении трассировка прекращается):

Оператор в блоке	$x < 1024$?	s	k
`s := 0;` `k := 0;`		0	0

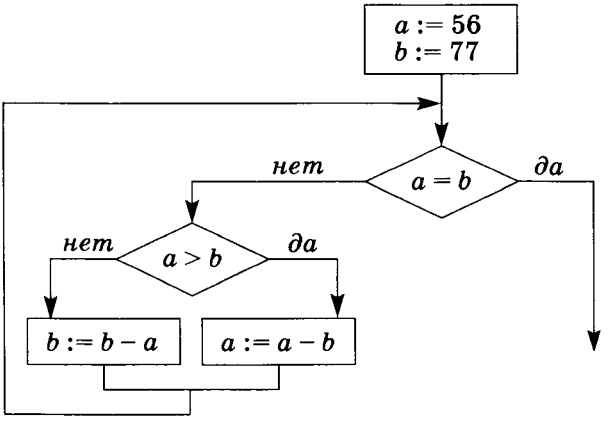
`s < 1024`	Да:
	`s := s + 10;`
	`k := k + 1;`
		10	1

`s < 1024`	Да:
	`s := s + 10;`
	`k := k + 1;`
		20	2

. . . . .			

`s < 1024`	Да:
	`s := s + 10;`
	`k := k + 1;`
		1030	**103**

Задача 2*. Запишите значение переменной a после выполнения фрагмента алгоритма:



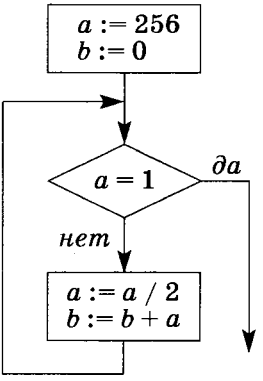
Примечание: знаком := обозначена операция присваивания.

Решение. Строится таблица трассировки.

Операторы	*a*	*b*
`a := 56; b := 77;`	**56**	**77**
`a = b` *Условие ложно — выполняется ветвь else*	56	77
`a > b` *Условие ложно — выполняется ветвь else*	56	77
`b := b - a;`	56	**21**
`a = b` *Условие ложно — выполняется ветвь else*	56	21
`a > b` *Условие истинно — выполняется ветвь then*	56	21
`a := a - b;`	**35**	21
`a = b` *Условие ложно — выполняется ветвь else*	35	21
`a > b` *Условие истинно — выполняется ветвь then*	35	21
`a := a - b;`	**14**	21
`a = b` *Условие ложно — выполняется ветвь else*	14	21
`a > b` *Условие ложно — выполняется ветвь else*	14	21
`b := b - a;`	14	**7**
`a = b` *Условие ложно — выполняется ветвь else*	14	7
`a > b` *Условие истинно — выполняется ветвь then*	14	7
`a := a - b;`	**7**	7
`a = b` *Условие истинно — выполняется ветвь then*	7	7
Прекращение работы программы	**7**	7

Таким образом, значение переменной *a* равно 7.
Ответ: $a = 7$.

Задача 3*. Запишите значение переменной b после выполнения фрагмента алгоритма:



Примечание: знаком := обозначена операция присваивания. В бланк ответа впишите только число.

Решение. Строится таблица трассировки.

Операторы в блоке	a	b
`a := 256` `b := 0`	**256**	**0**
`a = 1` *Условие не выполнено — выполняется «Нет»*	256 ⇑	0
`a := a / 2;`	**128**	0
`b := b + a;`	0	**128**
`a = 1` *Условие не выполнено — выполняется «Нет»*	128 ⇑	128
`a := a / 2;`	**64**	128
`b := b + a;`	0	**192**

Окончание табл.

Операторы в блоке	*a*	*b*
`a = 1` *Условие не выполнено — выполняется «Нет»*	64 ⇑	192
`a := a / 2;`	**32**	192
`b := b + a;`	0	**224**
`a = 1` *Условие не выполнено — выполняется «Нет»*	32 ⇑	224
`a := a / 2;`	**16**	224
`b := b + a;`	0	**240**
`a = 1` *Условие не выполнено — выполняется «Нет»*	16 ⇑	240
`a := a / 2;`	**8**	240
`b := b + a;`	0	**248**
`a = 1` *Условие не выполнено — выполняется «Нет»*	8 ⇑	248
`a := a / 2;`	**4**	248
`b := b + a;`	0	**252**
`a = 1` *Условие не выполнено — выполняется «Нет»*	4 ⇑	252
`a := a / 2;`	**2**	252
`b := b + a;`	0	**254**
`a := a / 2;`	**1**	
`b := b + a;`	0	**255**
`a = 1` *Условие выполнено — выполняется «Да»*	1 ⇑	255
Прекращение работы программы	1	255

Таким образом, по окончании работы программы значение переменной *b* равно 255.

Ответ: $b = 255$.

Задача 4*. Ниже на четырёх языках записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: L и M. Укажите наибольшее из таких чисел x, при вводе которых алгоритм печатает сначала 3, а потом 7.

Бейсик

```
DIM X, L, M AS INTEGER
INPUT X
L = 0: M = 0
WHILE X > 0
 L = L + 1
 IF M < (X MOD 10) THEN
    M = X MOD 10
 ENDIF
 X = X \ 10
WEND
PRINT L
PRINT M
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := 0; M := 0;
  while x > 0 do
   begin
    L := L + 1;
    if M < (x mod 10)
    then
     begin
       M := x mod 10;
     end;
    x := x div 10;
   end;
  writeln(L);
write(M);
end.
```

Си

```
#include<stdio.h>
void main()
{
   int x, L, M;
   scanf("%d", &x);
   L = 0; M = 0;
   while (x > 0){
      L = L + 1;
      if M < x % 10 {
         M = x % 10
      }
      x = x / 10;
   }
  printf("%d\n%d", L, M);
}
```

Алгоритмический язык

```
алг
нач
   цел x, L, M
   ввод x
   L := 0; M := 0
   нц пока x > 0
      L := L + 1
     если M < mod(x,10)
        то
          M := mod(x,10)
     все
     x := div(x,10)
    кц
    вывод L, нс, M
кон
```

Решение

В данном случае трассировка бессмысленна, так как исходное данное неизвестно (его и нужно определить). Поэтому решение заключается в общем анализе алгоритма.

1. Программа должна первым напечатать значение 3. Поскольку первым стоит оператор вывода `writeln(L)`, по завершении выполнения цикла переменная L должна быть равна 3. Изначально $L = 0$. Изменение этой переменной в цикле производится оператором `L := L + 1`, т.е. значение L указывает количество проходов цикла.

2. Цикл выполняется 3 раза. При этом второе печатаемое число (значение переменной M) равно 7.

Анализируя цикл целиком:

```
while x > 0 do
 begin
  L := L + 1;
  if M < (x mod 10) then
   begin
     M := x mod 10;
   end;
  x := x div 10;
 end;
```

Конструкция, включающая в себя сначала вычисление остатка от деления числа x на 10, а затем меняющая значение x на результат целочисленного деления x на 10, в цикле пока $x > 0$, является типовой и реализует последовательное выделение цифр числа x. Однако вычисление остатка от деления (и, соответственно, переприсваивание значения M) выполняется не всегда, а только в случае, если новый остаток больше, чем предыдущий.

Таким образом, данный цикл выполняет поочередное (справа налево — от младших разрядов к старшим) выделение цифр числа x и запоминание в переменной M наибольшей из этих цифр.

3. Итак, нужно найти наибольшее значение числа x, у которого наибольшая из цифр равна 7, а количество цифр равно трём (так как цикл выделения каждой цифры выполнялся трижды).

Такое число должно быть трёхзначным, наибольшая цифра 7 должна стоять первой, а две остальные должны быть максимально возможными, но не превышающими 7, — тогда всё число будет максимальным. Этим условиям соответствует число 777.

Ответ: $x = 777$.

При определении, какое число из возможных является наибольшим, нужно помнить простое правило: чем больше цифры в начале числа, тем больше само число.

Задача 5. Получив на вход число x, программа печатает два числа a и b. Укажите наибольшее из таких чисел x, при вводе которых программа напечатает сначала число 3, а потом число 35.

```
var x, a, b: integer;
  begin
  readln(x);
  a := 0;
  b := 1;
  while x > 0 do begin
    a := a + 1;
    b := b * (x mod 10);
    X := X div 10;
  end;
  writeln(a);
  write(b);
end.
```

Решение

Данная задача решается аналогично предыдущей.

1. Программа должна первым напечатать значение 3. Поскольку первым стоит оператор вывода `writeln(a)`, по завершении выполнения цикла переменная a должна быть равна 3. Изначально $a = 0$. Изменение этой переменной в цикле производится оператором `a := a + 1`, т.е. значение a определяет количество проходов цикла.

2. Таким образом, цикл выполняется 3 раза. При этом второе печатаемое число (значение пер. b) равно 35.

Анализируя цикл целиком:

```
while x > 0 do begin
  a := a + 1;
  b := b * (x mod 10);
  X := X div 10;
end;
```

Конструкция, включающая в себя сначала вычисление остатка от деления числа x на 10, а затем меняющая значение x на результат целочисленного деления x на 10, в цикле пока $x > 0$, является типовой и реализует последовательное выделение цифр числа x.

Данный цикл выполняет поочерёдное (справа налево — от младших разрядов к старшим) выделение цифр числа x и вычисление в переменной b произведения этих цифр.

3. Итак, нужно найти наибольшее значение числа x, у которого произведение цифр равно 35, а количество цифр равно трём (так как цикл выделения каждой цифры выполнялся трижды).

Число 35 можно разложить на множители как $5 \cdot 7$, причём эти множители простые. Чтобы получить три множителя, можно добавить к ним единицу:
$35 = 5 \cdot 7 \cdot 1$.

Таким образом, для получения заданных результатов (чисел 3 и 35) нам годятся значения x, равные всем возможным комбинациям цифр 1, 5 и 7. Наибольшее из таких чисел равно 751.

Ответ: $x = 751$.

Важно не забывать, что при определении исходного значения x надо учитывать не только разложение второго из заданных чисел (здесь — 35) на простые множители, но и количество проходов цикла, определяемое первым из заданных чисел (здесь — 3).

Задача 6. Получив на вход число x, программа печатает два числа a и b. Укажите наименьшее из таких чисел x, при вводе которых программа напечатает сначала число 4, а потом число 24.

```
var x, a, b: integer;
  begin
  readln(x);
  a := 0; b := 1;
  while x > 0 do begin
    a := a + 1;
    b := b * (x mod 10);
    X := X div 10;
  end;
  writeln(a);
  write(b);
end.
```

Решение

Решение заключается в общем анализе алгоритма.

1. Программа должна первым напечатать значение 4, значит, по завершении выполнения цикла переменная a должна быть равна 4. Изначально $a = 0$, изменение этой переменной в цикле производится оператором `a := a + 1`, значит, цикл должен быть выполнен 4 раза. При этом второе печатаемое число (значение переменной b) равно 24.

2. Анализируя цикл целиком:

```
while x > 0 do begin
  a := a + 1;
  b := b * (x mod 10);
  X := X div 10;
end;
```

Конструкция, включающая в себя сначала вычисление остатка от деления числа x на 10, а затем меняющая значение x на результат целочисленного деления x на 10, в цикле пока $x > 0$, является типовой и реализует последовательное выделение цифр числа x.

Данный цикл выполняет поочерёдное (справа налево — от младших разрядов к старшим) выделение цифр числа x и вычисление в переменной b произведения этих цифр.

3. Нужно найти наименьшее значение числа x, у которого произведение цифр равно 24, а количество цифр равно четырём (так как цикл выделения каждой цифры выполнялся четыре раза).

Число 24 можно разложить на четыре однозначных множителя различными способами:

$$24 = 4 \cdot 6 \cdot 1 \cdot 1 = 4 \cdot 3 \cdot 2 \cdot 1 = 2 \cdot 2 \cdot 6 \cdot 1 = 2 \cdot 2 \cdot 2 \cdot 3.$$

Таким образом, в качестве исходного числа x подходят все возможные комбинации цифр: 4, 6, 1, 1 или 4, 3, 2, 1 или 2, 2, 6, 1 или 2, 2, 2, 3.

Находятся наименьшие возможные значения x, которые можно получить из таких цифр в каждой из полученных групп, и сравниваются:

Цифры	4, 6, 1, 1	4, 3, 2, 1	2, 2, 6 ,1	2, 2, 2, 3
Наименьшее возможное число	1146	1234	1226	2223

Тогда наименьшее возможное значение x, удовлетворяющее условиям задачи, равно 1146.

Ответ: $x = 1146$.

Задача 7. Получив на вход число x, программа печатает два числа a и b. Укажите наибольшее из таких чисел x, при вводе которых программа напечатает сначала число 3, а потом число 10.

```
var x, a, b: integer;
  begin
  readln(x);
  a := 0; b := 0;
  while x > 0 do begin
    a := a + 1;
    b := b + (x mod 10);
    X := X div 10;
  end;
  writeln(a);
  write(b);
end.
```

Решение

Решение заключается в общем анализе алгоритма.

1. Программа должна первым напечатать значение 3, значит, по завершении выполнения цикла переменная a должна быть равна 3. Изначально $a = 0$, изменение этой переменной в цикле производится оператором `a := a + 1`, значит цикл должен быть выполнен 3 раза.

2. Второе печатаемое число (значение переменной b) должно быть равно 10.

Анализируя цикл целиком:

```
while x > 0 do begin
  a := a + 1; b := b + (x mod 10);
  X := X div 10;
end;
```

Конструкция, включающая в себя сначала вычисление остатка от деления числа x на 10, а затем меняющая значение x на результат целочисленного деления x на 10, в цикле пока $x > 0$, является типовой и реализует последовательное выделение цифр числа x.

Данный цикл выполняет поочерёдное (справа налево — от младших разрядов к старшим) выделение цифр числа x и вычисление в переменной b суммы этих цифр.

3. Нужно найти наибольшее значение числа x, у которого сумма цифр равна 10, а количество цифр равно трём (так как цикл выделения каждой цифры выполнялся три раза). Поэтому требуется найти разложение числа 10 как суммы трёх цифр, таких, что составленное из них число — максимально возможное.

Такое разложение выполняется: $10 = 9 + 1 + 0$.

Ответ: $x = 910$.

A12 Операции с массивами: анализ программ

Конспект

Массив

Во многих задачах требуется выполнять одни и те же операции с большим количеством данных одного и того же типа. Например, это может быть поиск требуемой фамилии в электронном телефонном справочнике, сортировка по возрастанию цен в прайс-листе интернет-магазина, вычисление средней годовой оценки в классе, изменение яркости растрового изображения путём изменения интенсивности цветовых составляющих для каждого пикселя и т.д.

Для таких применений предусмотрены составные типы данных, одним из наиболее широко используемых среди которых является тип «массив».

Одномерный массив представляет собой пронумерованную последовательность переменных одного и того же типа, имеющих общее **имя**. Обращение к конкретной переменной (**элементу массива**) производится по **имени массива** и порядковому номеру (**индексу**) нужного элемента в нём.

Двумерный массив можно рассматривать как одномерный массив, элементами которого являются одномерные массивы. Другое представление двумерного массива — **матрица**: прямоугольная или квадратная таблица, в которой элементы массива соответствуют ячейкам, а номера строк и столбцов представляют собой два индекса каждого элемента.

Аналогично, трёхмерный массив может быть рассмотрен как *куб данных*, состоящий из элементарных кубиков (элементов массива), каждый из которых имеет три индекса (т.е. трёхмерный массив рассматривается как одномерный массив «слоёв» — матриц).

По тому же принципу могут быть определены массивы большей размерности.

Массивы в языке Паскаль

Объявление n-мерного массива:

array [<диапазон индекса 1>, ..., <диапазон индекса N>] **of** <базовый тип>;

где *<диапазон индекса>* — записанные через две точки начальное и конечное значения индексов элементов по соответствующей размерности (целые числа или символы); *<базовый тип>* — тип элементов массива.

Примеры:

1) объявление одномерного массива из 10 целых чисел (базовый тип — integer), индексы которых меняются от 0 до 9:

```
var Mas : array [0..9] of integer;
```

2) объявление одномерного массива из 15 символов (базовый тип — char), индексы которых меняются от −7 до 7:

```
var Mas : array [-7..7] of char;
```

3) объявление одномерного массива из вещественных чисел (базовый тип — real), в качестве индексов которых используются латинские буквы:

```
var Mas : array ['a'..'z'] of real;
```

4) объявление двумерного массива размером 10 × 15 из целых чисел (базовый тип — integer), индексы которых отсчитываются с единицы:

```
var Mas : array [1..10, 1..15] of integer;
```

или

```
var Mas : array [1..10] of array [1..15] of
integer;
```

Обращение к элементу:

1) одномерного массива: `Mas[i]`

2) двумерного массива: `Mas[i,j]`

При обращении к элементу массива его индекс (индексы) может быть задан константой соответствующего типа (совпадающего с типом индекса в описании массива), переменной или выражением, результат которого имеет соответствующий тип. Это даёт возможность реализовать поочерёд-

ную обработку элементов массива в цикле, в котором требуемым образом меняются индексы элементов. При этом важно следить, чтобы значение индекса при обращении к элементу массива не выходило за пределы массива (за пределы указанного в его описании диапазона изменения индексов).

В программе может быть также описан «типовой» массив как своего рода «шаблон», по которому затем могут быть определены «экземпляры» таких массивов. Пример:

```
type
  Mas = array [1..10] of integer;
var M1, M2 : Mas;
```

Здесь вначале описан «шаблон» одномерного массива из 10 целочисленных элементов, а затем по нему, как по образцу, определены два таких массива с именами M1 и M2. В результате с точки зрения программы оба эти массива полностью идентичны и для них допустима операция присваивания одного массива другому целиком: M1 := M2 (все элементы массива M2 копируются в соответствующие им элементы M1).

Заполнение массива

Производится в цикле (для многомерного массива — во вложенных циклах) поэлементно путём ввода каждого элемента с клавиатуры (оператором read) либо присваивания каждому элементу некоторого значения — константы (например, при обнулении массива) или вычисленного значения выражения.

Другой возможный вариант — указание значений элементов массива при его объявлении.

Примеры:

1) обнуление одномерного массива:

```
var mas : array[1..4] of integer := (0,0,0,0);
```

2) присваивание начальных значений элементам двумерного массива:

```
var mas : array[1..2,1..5] of integer :=
                    ((1,2,3,4,5),(6,7,8,9,10));
```

Вывод массива на экран

Производится в цикле (для многомерного массива — во вложенных циклах) поэлементно путём вывода каждого элемента с клавиатуры (оператором `write`).

Для одномерного массива обычно используется оператор `write`, в котором, кроме обращения к *i*-му элементу массива, предусмотрен разделяющий пробел (пример: `write(Mas[i],' ');`), тогда элементы массива выводятся в строку.

Возможен альтернативный вариант, когда используется оператор `writeln`, в котором указывается значение индекса и значение элемента с этим индексом (пример: `writeln(i,'-й элемент равен ', Mas[i]);`), тогда каждый элемент выводится в отдельной строке.

Для двумерного массива вывод строк обычно производится в одну строку (оператор `write`), а по завершении вывода очередной строки отдельно добавляется пустой оператор `writeln` для перехода на следующую строку.

Обмен местами элементов массива

Производится аналогично обмену значений двух обычных переменных (обычно при помощи дополнительной «буферной» переменной).

Обработка элементов массива (определение максимума/минимума, вычисление суммы, произведения, среднего и пр.)

В цикле (для многомерного массива — во вложенных циклах) производится перебор элементов массива (полный или частичный — для фрагмента массива).

- **При поиске максимума/минимума** за предполагаемый максимум/минимум берётся первый элемент массива либо константа, заведомо меньшая/большая любого элемента. Далее каждый очередной элемент массива сравнивается с предполагаемым мак-

симумом/минимумом, и если этот элемент больше/меньше предполагаемого максимума/минимума, то значение этого элемента запоминается в качестве нового предполагаемого максимума/минимума. Дополнительно при этом в отдельной переменной (переменных) может перезапоминаться индекс (индексы) очередного предполагаемого максимума/минимума.

- **При вычислении суммы/произведения** вначале переменной, выделенной для накопления суммы/произведения, присваивается инициализационное значение (нуль — для суммы, единица — для произведения). Затем в цикле (либо вложенных циклах) выполняется перебор элементов массива и текущее значение суммы/произведения складывается/умножается на текущий элемент массива.

- **При вычислении среднего значения** выполняется суммирование элементов массива, а затем полученная сумма делится на количество элементов в массиве.

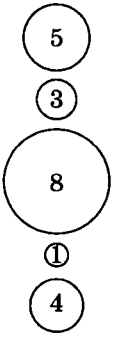
- **При определении максимума/минимума, суммы, произведения, среднего значения элементов, удовлетворяющих заданному условию (например, только положительных)** дополнительно добавляется условный оператор с соответствующим условием, и требуемое действие (проверка и переприсваивание предполагаемого максимума/минимума, сложение, умножение) выполняется только при истинности этого условия. При вычислении среднего значения также предусматривается отдельная переменная-счётчик, которая увеличивается на 1 каждый раз, когда к сумме добавляется очередной удовлетворяющий условию элемент массива, и после вычисления суммы она делится на значение этого счётчика (количество вошедших в сумму элементов).

Сортировка массива

Выполняется путём многократного просмотра массива, попарного сравнения его элементов между собой и при необходимости — соответствующей перестановки этих элементов местами (при сортировке по убыванию на первые места переносятся элементы с большими значениями; при сортировке по убыванию — наоборот).

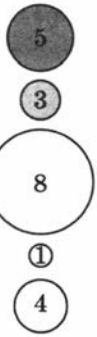
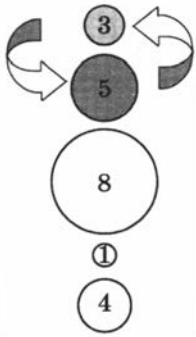
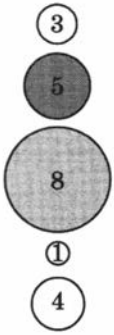
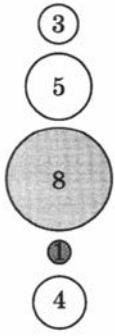
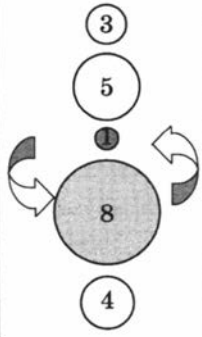
Существует несколько методов сортировки массивов, различающихся сложностью их алгоритма и скоростью работы: сортировка перемешиванием, сортировка выбором, сортировка вставками, быстрая сортировка, пирамидальная, «гномья», слиянием, деревом, сортировка Шелла, но наиболее понятной и наглядной является **пузырьковая сортировка** (она же — **сортировка простыми обменами**).

Метод пузырьковой сортировки можно рассмотреть на примере числового массива из 5 элементов: (5, 3, 8, 1, 4), — который надо отсортировать по возрастанию. Для наглядности элементы располагаются по вертикали, друг над другом, и обводятся кружками разного размера.

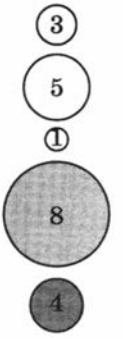
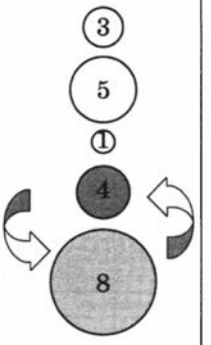


Если бы в условии было всего два элемента (например, числа 5 и 3, с которых начинается наш массив), то достаточно было бы сравнить их друг с другом и, если первое из этих чисел больше второго, то поменять их местами.

То же самое можно сделать и для нашего массива — поочерёдно выбрать пары элементов (начиная с первого), сравнить их и поменять местами, если первый из взятой пары элементов окажется больше второго (так как требуется сортировка по возрастанию). Поскольку необходимо разработать алгоритм для исполнения компьютером, используется конструкция «цикл» (конечное значение цикловой переменной *i* определяется в ходе разбора алгоритма).

№ цикла	№ шага	№№ выбранных элементов (i и $i+1$)	До обмена элементов	После обмена элементов
1	1	1 и 2		
1	2	2 и 3		
1	3	3 и 4		

Окончание табл.

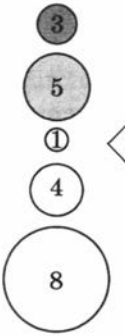
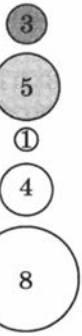
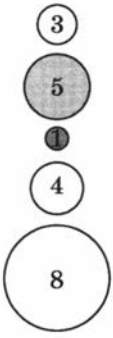
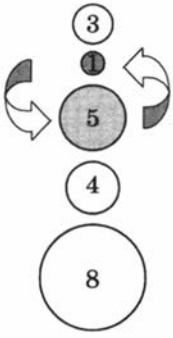
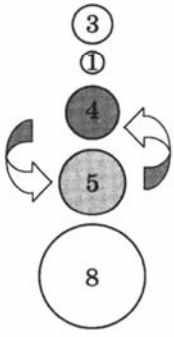
№ цикла	№ шага	№№ выбранных элементов (i и $i+1$)	До обмена элементов	После обмена элементов
1	4	4 и 5		

Следует помнить, что в цикле значение цикловой переменной i должно меняться от 1 до 4, или, для массива произвольной длины n, — от 1 до $(n-1)$.

Кроме того, хотя массив ещё не отсортирован, самое больше число 8 уже «опустилось» в самый низ на соответствующую ему правильную позицию.

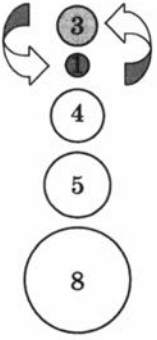
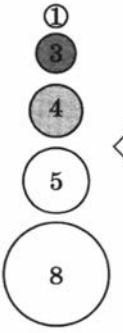
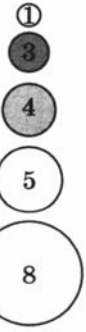
Остаётся повторить цикл ещё раз!

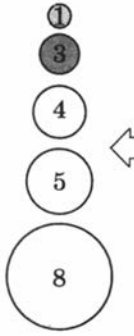
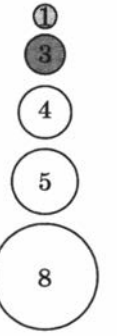
Зная, что последнее число уже стоит на своём месте и его можно исключить из сравнений, меняется значение цикловой переменной от 1 до 3 (т.е. для произвольного массива — от 1 до $(n-2)$).

№ цикла	№ шага	№№ выбранных элементов (*i* и *i*+1)	До обмена элементов	После обмена элементов
2	1	1 и 2		
2	2	2 и 3		
2	3	3 и 4		

Теперь и предпоследняя позиция в массиве тоже занята «правильным» элементом.

Для окончания сортировки в данном случае достаточно поменять местами два первых элемента. Помня, что компьютер является формальным исполнителем, и массив может быть каким угодно, тогда как алгоритм должен обладать свойством универсальности (быть пригодным для сортировки любого массива), требуется выполнить ещё два цикла, в первом из которых переменная i меняется от 1 до 2, а во втором — равна числу 1, даже если один из шагов этого цикла окажется «лишним».

№ цикла	№ шага	№№ выбранных элементов (i и i+1)	До обмена элементов	После обмена элементов
3	1	1 и 2		
3	2	2 и 3		

№ цикла	№ шага	№№ выбранных элементов (i и $i+1$)	До обмена элементов	После обмена элементов
4	1	1 и 2		

В ходе сортировки меньшие числа, подобно воздушным пузырькам в воде, постепенно «всплывают» наверх, а большие, словно камешки, опускаются вниз («тонут»). Именно поэтому данный метод сортировки и получил своё название.

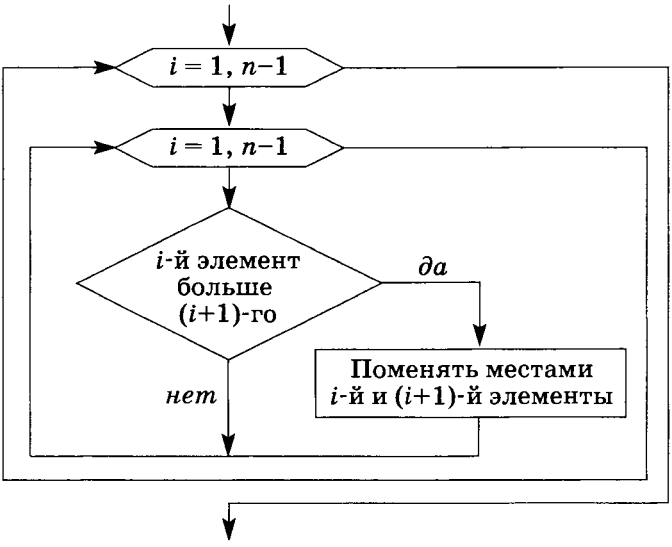
Обобщив только что разобранные действия на случай произвольного массива из n элементов получается, что:

1) потребуется $(n-1)$ раз выполнять циклы просмотра элементов массива;

2) в каждом из этих циклов конечное значение цикловой переменной должно становиться меньше на 1, а в самом первом из этих циклов оно должно быть равно $(n-1)$;

3) на каждом шаге выполняется сравнение пары элементов и (для сортировки по возрастанию) если первый из них больше второго, то они меняются местами.

Очевидно, что реализовать все это можно при помощи конструкции из двух вложенных циклов:



Соответствующая программа (например, на Паскале) имеет вид:

```
program bubble_sort;

const nn = 10;      // максимально возможный
                                размер массива

var i, j : integer; // цикловые переменные
    n : integer;  // реальный размер массива
    mass : array[1..nn] of integer; // массив
    t : integer;  // вспомогательная
                          переменная для обмена

begin
  write('Введите размер массива: ');
  readln(n);

  writeln('Введите (через пробел) элементы массива');
  for i := 1 to n do
    read(mass[i]);
    // сортировка методом пузырька
  for j := 1 to n - 1 do begin
    for i := 1 to n - j do begin
      if mass[i] > mass[i+1] then begin
```

```
            // меняем местами элементы
            t := mass[i];
            mass[i] := mass[i+1];
            mass[i+1] := t;
        end;
    end;
end;

writeln('Отсортированный массив');
for i := 1 to n do
    write(mass[i],' ');
writeln;

end.
```

Разбор типовых задач

Задача 1*. В программе используется одномерный целочисленный массив A с индексами от 0 до 9. Ниже представлен фрагмент программы, записанный на разных языках программирования, в котором значения элементов сначала задаются, а затем меняются.

Бейсик	Паскаль

For i = 0 To 9
A.SetValue(9-i, i)
Next
For i = 0 To 4
K = A.GetValue(i)
A.SetValue(A.GetValue
(9-i), i)
A.SetValue(k, 9-i)
Next
	for i := 0 to 9 do
	A[i] := 9 - i;
	for i := 0 to 4 do
	begin
	k := A[i];
	A[i] := A[9-i];
	A[9-i] := k;
	end;

Си	**Алгоритмический язык**

for (i = 0; i <= 9; i++)
A[i] = 9 - i;
for (i = 0; i <= 4; i++)
{
k = A[i];
A[i] = A[9-i];
A[9-i] = k;
}
	нц для i от 0 до 9
	A[i] := 9 - i
	кц
	нц для i от 0 до 4
	k := A[i]
	A[i] := A[9-i]
	A[9-i] := k
	кц

Чему будут равны элементы этого массива после выполнения фрагмента программы?

1) 9 8 7 6 5 4 3 2 1 0
2) 0 1 2 3 4 5 6 7 8 9
3) 9 8 7 6 5 5 6 7 8 9
4) 0 1 2 3 4 4 3 2 1 0

Решение

1. Модель массива *A*:

0	1	2	3	4	5	6	7	8	9

2. Массив при заполнении обрабатывается весь (диапазон изменения цикловой переменной совпадает с размером массива), а при изменении обрабатывается только часть массива (для i от 0 до 4).

3. Заполнение массива: каждому элементу присваивается значение, равное разности константы 9 и индекса этого элемента:

0	1	2	3	4	5	6	7	8	9
9	8	7	6	5	4	3	2	1	0

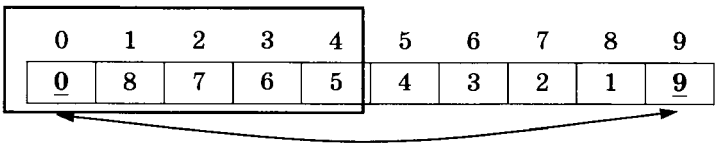
4. Изменение массива выполняется стоящей в теле второго цикла парой операторов:

```
k := A[i];
A[i] := A[9-i];
A[9-i] := k;
```

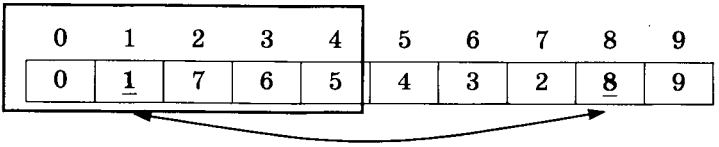
Эта алгоритмическая конструкция — обмен местами двух элементов (i-го и ($9-i$)-го) через буферную переменную k.

Эта операция выполняется последовательно для всех значений i:

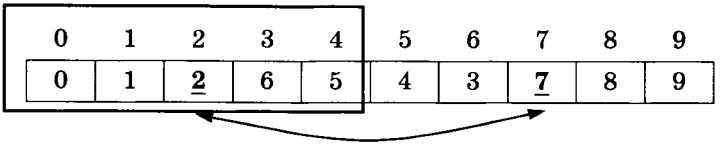
- $i = 0$:



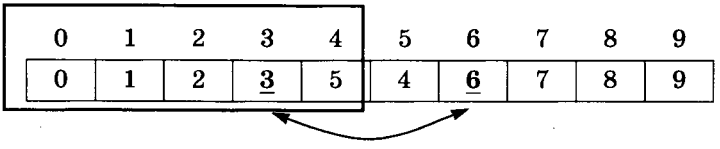
- $i = 1$:



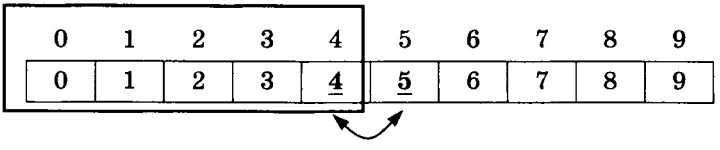
- $i = 2$:



- $i = 3$:



- $i = 4$:



То есть элементы массива меняются местами первый — с последним, второй — с предпоследним и т.д. В результате получается массив 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Ответ: массив 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (вариант ответа №2).

При определении обрабатываемой части массива следует быть внимательными! Если бы данный массив был обработан целиком, то, начиная со значения $i = 5$, происходил бы обратный обмен значений элементов. В результате вновь получился бы исходный массив.

Задача 2*. В программе описан одномерный целочисленный массив A с индексами от 0 до 10. Ниже представлен фрагмент этой программы, записанный на разных языках программирования, в котором значения элементов массива сначала задаются, а затем меняются.

Бейсик	Паскаль

FOR i = 0 TO 10
A(i) = i - 1
NEXT i
FOR i = 10 TO 1 STEP -1
A(i-1) = A(i)
NEXT i
	for i := 0 to 10 do
	  A[i] := i - 1;
	for i := 10 downto 1 do
	  A[i-1] := A[i];

Си	**Алгоритмический язык**

for (i = 0; i <= 10; i++)
  A[i] = i - 1;
for (i = 10; i >= 1; i--)
  A[i-1] = A[i];
	нц для i от 0 до 10
	  A[i] := i - 1
	кц
	нц для i от 10 до 1 шаг -1
	  A[i-1] := A[i]
	кц

Чему окажутся равны элементы этого массива?

1) 9 9 9 9 9 9 9 9 9 9 9
2) 0 1 2 3 4 5 6 7 8 9 9
3) 0 1 2 3 4 5 6 7 8 9 10
4) −1 −1 0 1 2 3 4 5 6 7 8

Решение

1. Модель массива A:

0	1	2	3	4	5	6	7	8	9	10

2. Массив обрабатывается весь.

3. Заполнение массива: каждому элементу присваивается значение, равное его индексу, уменьшенному на 1:

0	1	2	3	4	5	6	7	8	9	10
−1	0	1	2	3	4	5	6	7	8	9

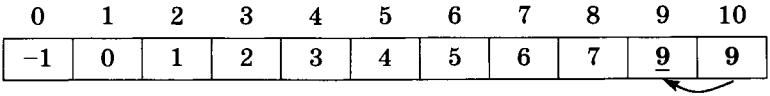
4. Изменение массива выполняется циклом:

```
for i := 10 downto 1 do
  A[i-1] := A[i];
```

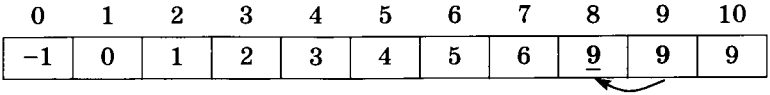
То есть массив проходится справа налево (в обратном порядке — от последних элементов к первым), и текущий элемент записывается (копируется) в элемент с индексом на 1 меньше.

Эта операция выполняется последовательно для всех значений i:

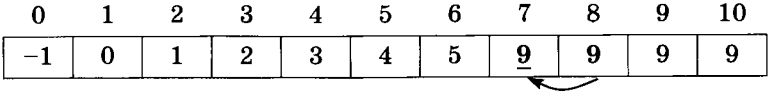
- $i = 10$:



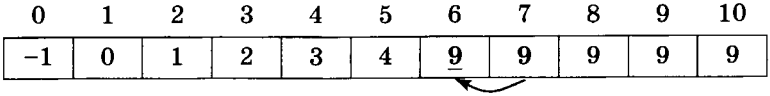
- $i = 9$:



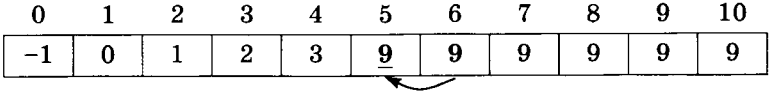
- $i = 8$:



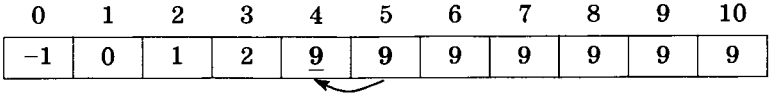
- $i = 7$:



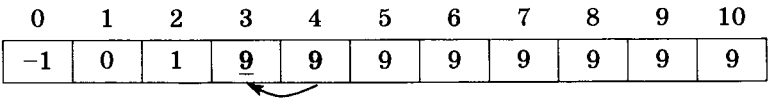
- $i = 6$:



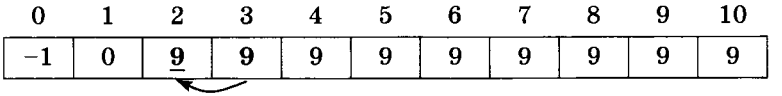
- $i = 5$:



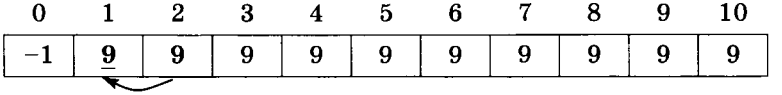
- $i = 4$:



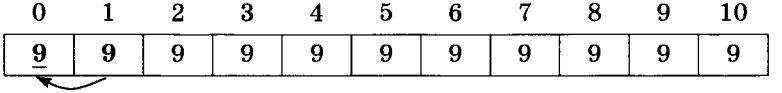
- $i = 3$:



- $i = 2$:



- $i = 1$:



Таким образом, в результате получается массив, в котором все элементы равны 9.

Ответ: массив 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 (вариант ответа №1).

Задача 3*. В программе используется одномерный целочисленный массив A с индексами от 0 до 10. Ниже представлен фрагмент программы, записанный на разных языках программирования, в котором значения элементов сначала задаются, а затем меняются.

Бейсик	Паскаль

`FOR i = 0 TO 10`
`A(i) = i`
`NEXT i`
`FOR i = 0 TO 10`
`A(10-i) = A(i)`
`A(i) = A(10-i)`
`NEXT i`
	`for i := 0 to 10 do`
	`  A[i] := i;`
	`for i := 0 to 10 do`
	`begin`
	`  A[10-i] := A[i];`
	`  A[i] := A[10-i];`
	`end;`

Си	Алгоритмический язык

`for (i = 0;i <= 10; i++)`
`  A[i]=i;`
`for (i = 0; i <= 10; i++)`
`{`
`  A[10-i] = A[i];`
`  A[i] = A[10-i];`
`}`
	`нц для i от 0 до 10`
	`  A[i] := i`
	`кц`
	`нц для i от 0 до 10`
	`  A[10-i] := A[i]`
	`  A[i] := A[10-i]`
	`кц`

Чему будут равны элементы этого массива после выполнения фрагмента программы?

1) 10 9 8 7 6 5 4 3 2 1 0
2) 0 1 2 3 4 5 6 7 8 9 10
3) 10 9 8 7 6 5 6 7 8 9 10
4) 0 1 2 3 4 5 4 3 2 1 0

Решение

1. Модель массива *A*:

0	1	2	3	4	5	6	7	8	9	10

2. Массив обрабатывается весь (диапазон изменения цикловой переменной совпадает с размером массива).

3. Заполнение массива: каждому элементу присваивается значение, равное его индексу:

0	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	10

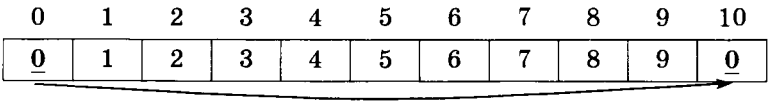
4. Изменение массива выполняется стоящей в теле второго цикла парой операторов:

```
A[10-i] := A[i];
A[i] := A[10-i];
```

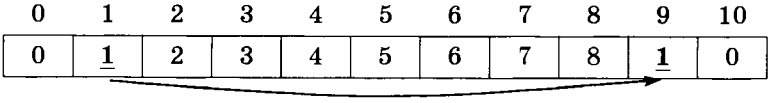
Эта алгоритмическая конструкция *похожа* на обмен местами двух элементов (i-го и $(10-i)$-го), но **не является обменом**, поскольку здесь не используется буферная переменная! Сначала в $(10-i)$-й элемент заносится значение i-го элемента (а прежнее значение теряется), а затем выполняется обратное присваивание, которое, очевидно, уже ничего не меняет.

Эта операция выполняется последовательно для всех значений i:

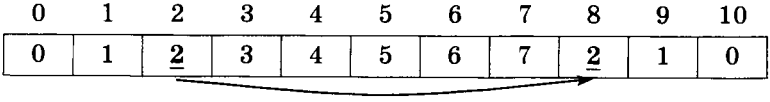
- $i = 0$:



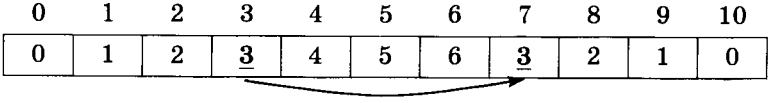
- $i = 1$:



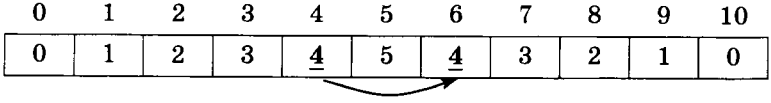
- $i = 2$:



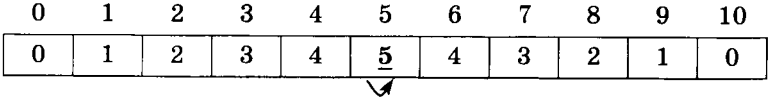
- $i = 3$:



- $i = 4$:

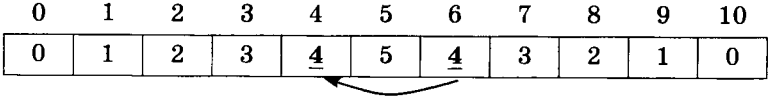


- $i = 5$:



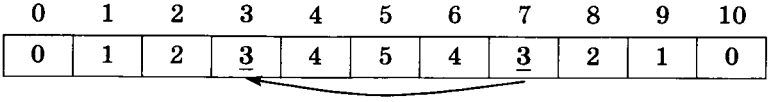
(При обмене элемента самого с собой никаких изменений не происходит.)

- $i = 6$:

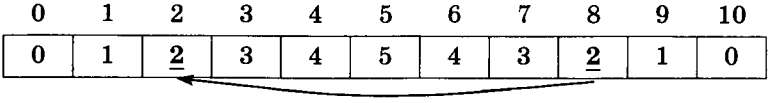


При дальнейших операциях массив не изменяется.

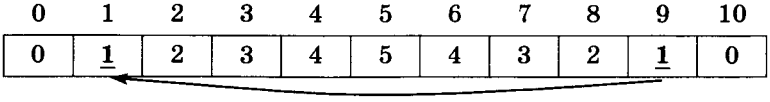
- $i = 7$:



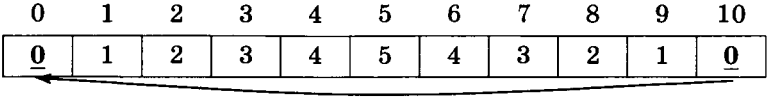
- $i = 8$:



- $i = 9$:



- $i = 10$:



Таким образом, в результате получается массив из элементов: 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0.

Ответ: массив 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0 (вариант ответа №4).

Задача 4*. Все элементы двумерного массива A размером 10×10 элементов первоначально были равны 0. Затем значения элементов меняются с помощью вложенного оператора цикла в представленном фрагменте программы (ниже представлена одна и та же программа, записанная на разных языках программирования).

Бейсик	Паскаль

`FOR n = 1 TO 4`
`FOR k = n TO 4`
`  A(n,k) = A(n,k) + 1`
`  A(k,n) = A(k,n) + 1`
`NEXT k`
`NEXT n`
	`for n := 1 to 4 do`
	`for k := n to 4 do`
	`begin`
	`  A[n,k] := A[n,k] + 1;`
	`  A[k,n] := A[k,n] + 1;`
	`end`

Алгоритмический язык

```
нц для n от 1 до 4
нц для k от n до 4
   A[n,k] := A[n,k] + 1
   A[k,n] := A[k,n] + 1
кц
кц
```

Сколько элементов массива в результате будут равны 1?

1) 0 2) 16 3) 12 4) 4

Решение

1. Модель двумерного массива:

n \ k	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

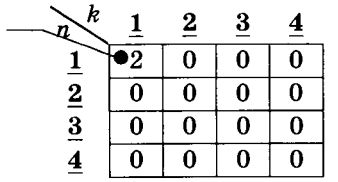
2. Обрабатываемая область:

n \ k	**1**	**2**	**3**	**4**	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

3. Поэлементно заполняется массив, отслеживая изменение индексных переменных (ниже приводится только нужный фрагмент модели массива; при его выделении нужно быть внимательным — в зависимости от записи индексов элемента возможен выход значений индексов за пределы изменения значений индексных переменных в цикле).

1)

```
A[n,k] =
= A[k,n]
```



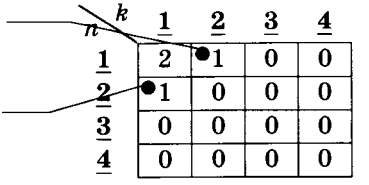
$n = 1,\ k = n = 1$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[1,1]`. Затем на единицу должно быть увеличено значение `A[k,n]`, т.е. `A[1,1]`. Поскольку это тот же самый элемент, его значение увеличивается на 1 ещё раз и становится равным 2. Это, будет происходить всегда, когда $n = k$.

2)

```
A[n,k]
```

```
A[k,n]
```



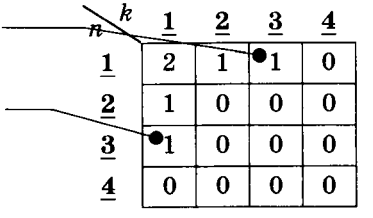
$n = 1,\ k = 2$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[1,2]`. Затем на единицу увеличено значение `A[k,n]`, т.е. `A[2,1]`.

3)

```
A[n,k]
```

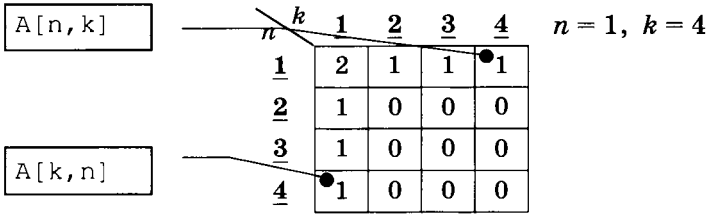
```
A[k,n]
```



$n = 1,\ k = 3$

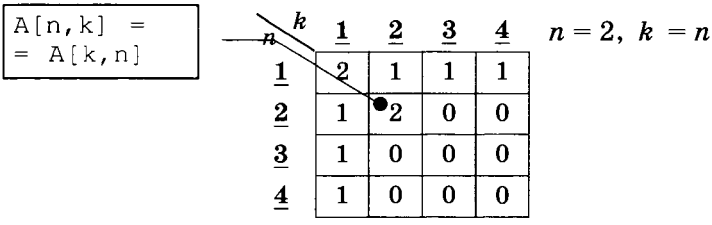
Вначале на единицу увеличено значение A[n,k], т.е. A[1,3]. Затем на единицу увеличено значение A[k,n], т.е. A[3,1].

4)



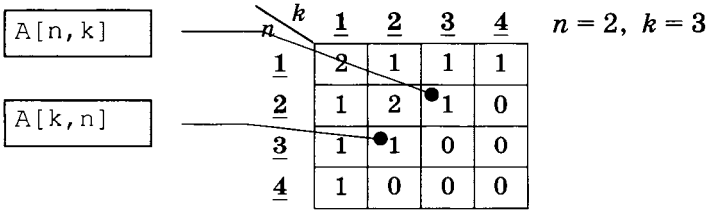
Вначале на единицу увеличено значение A[n,k], т.е. A[1,4]. Затем на единицу увеличено значение A[k,n], т.е. A[4,1].

5) внутренний цикл завершён, выполняется новый проход внешнего цикла и внутренний цикл запускается заново:



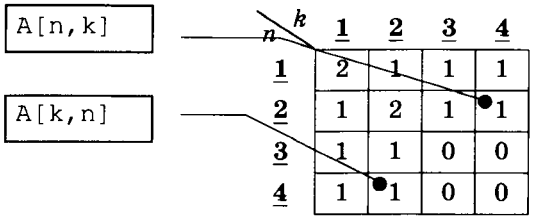
Вначале на единицу увеличено значение A[n,k], т.е. A[2,2]. Затем на единицу должно быть увеличено значение A[k,n], т.е. снова A[2,2].

6)



Вначале на единицу увеличено значение A[n,k], т.е. A[2,3]. Затем на единицу увеличено значение A[k,n], т.е. A[3,2].

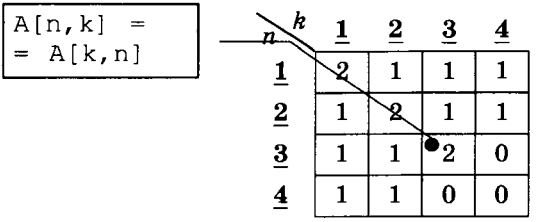
7)



$n = 2,\ k = 4$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[2,4]`. Затем на единицу увеличено значение `A[k,n]`, т.е. `A[4,2]`.

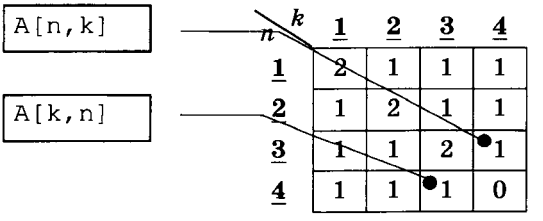
8) внутренний цикл завершён, выполняется новый проход внешнего цикла и внутренний цикл запускается заново:



$n = 3,\ k = n = 3$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[3,3]`. Затем на единицу должно быть увеличено значение `A[k,n]`, т.е. снова `A[3,3]`.

9)

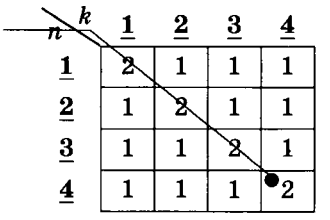


$n = 3,\ k = 4$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[3,4]`. Затем на единицу увеличено значение `A[k,n]`, т.е. `A[4,3]`.

10) внутренний цикл завершён, выполняется новый проход внешнего цикла и внутренний цикл запускается заново:

```
A[n,k] =
= A[k,n]
```



$n = 4,\ k = n = 4$

Вначале на единицу увеличено значение `A[n,k]`, т.е. `A[4,4]`. Затем на единицу должно быть увеличено значение `A[k,n]`, т.е. снова `A[4,4]`.

11) внутренний цикл завершён, внешний цикл также завершён.

4. В полученном массиве подсчитывается количество полученных единиц:

- всего ненулевых ячеек массива: $4 \cdot 4 = 16$ (остальные ячейки остались равны нулю и нас не интересуют);
- двойками заняты 4 ячейки;
- тогда единицы содержат $16 - 4 = 12$ ячеек.

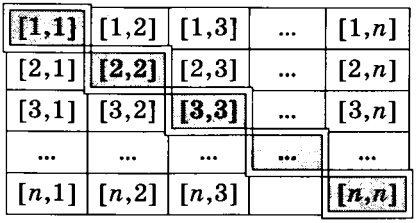
Ответ: 12 элементов (вариант ответа №3).

C2 Операции с массивами: задачи группы «С»

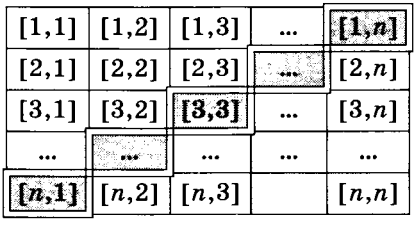
Конспект

Характерные элементы двумерного массива (матрицы)

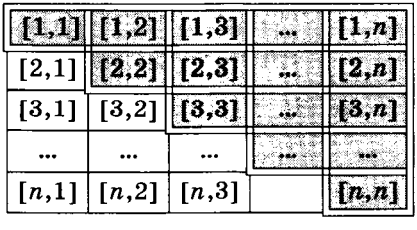
Главная диагональ матрицы — это совокупность её элементов, расположенных по диагонали от верхнего левого (первого) до нижнего правого (последнего) элемента. Эти элементы имеют равные значения индексов: [1,1], [2,2], ..., [n,n], где n — размер (порядок) матрицы.



Побочная диагональ матрицы — это совокупность её элементов, расположенных по диагонали от верхнего правого до нижнего левого элемента. Таким образом, побочная диагональ является «зеркально симметричной» по отношению к главной. Её элементы имеют следующие значения индексов: [1,*n*], [2,*n*-1], ..., [*n*,1], где *n* — размер (порядок) матрицы.



Верхняя матрица — это совокупность элементов, расположенных на главной диагонали и выше нее. Эти элементы имеют значения индексов: [*i*,*j*], где *i* меняется от 1 до *n* (размер матрицы), а *j* — меняется во вложенном цикле от *i* до *n*. ***Для верхней матрицы без главной диагонали*** *i* меняется от 1 до *n* (размер матрицы), а *j* — меняется во вложенном цикле от *i*+1 до *n*.



Нижняя матрица — это совокупность элементов, расположенных на главной диагонали и ниже нее. Эти элементы имеют значения индексов: [*i*,*j*], где *i* меняется

от 1 до n (размер матрицы), а j — меняется во вложенном цикле от 1 до i. ***Для нижней матрицы без главной диагонали*** i меняется от 2 до n (размер матрицы), а j — меняется во вложенном цикле от i-1 до n.

[1,1]	[1,2]	[1,3]	...	[1,n]
[2,1]	[2,2]	[2,3]	...	[2,n]
[3,1]	[3,2]	[3,3]	...	[3,n]
...	...	...	...	...
[n,1]	[n,2]	[n,3]		[n,n]

Заполнение массива

Производится в цикле (для многомерного массива — во вложенных циклах) поэлементно путём ввода каждого элемента с клавиатуры (оператором `read`) либо присваивания каждому элементу некоторого значения — константы (например, при обнулении массива) или вычисленного значения выражения.

Пример:

```
var Mas: array [1..10] of integer;
begin
for i := 1 to 10 do
  read(Mas[i]);
end.
```

Другой возможный вариант — указание значений элементов массива при его объявлении. Примеры:

1) обнуление одномерного массива:

```
var mas : array[1..4] of integer := (0,0,0,0);
```

2) присваивание начальных значений элементам двумерного массива:

```
var mas : array[1..2,1..5] of integer :=
                  ((1,2,3,4,5),(6,7,8,9,10));
```

Вывод массива на экран

Производится в цикле (для многомерного массива — во вложенных циклах) поэлементно путём вывода каждого элемента с клавиатуры (оператором `write`).

Для одномерного массива обычно используется оператор `write`, в котором, кроме обращения к *i*-му элементу массива, предусмотрен разделяющий пробел. В этом случае элементы массива выводятся в строку.

Пример:

```
var Mas: array [1..10] of integer;
. . . . .
for i := 1 to 10 do
  write(Mas[i],' ');
end.
```

Возможен альтернативный вариант, когда используется оператор `writeln`, в котором указывается значение индекса и значение элемента с этим индексом. В этом случае каждый элемент выводится в отдельной строке.

Пример:

```
var Mas: array [1..10] of integer;
. . . . .
for i := 1 to 10 do
  writeln(i,'-й элемент равен ', Mas[i]);
end.
```

Для двумерного массива вывод строк обычно производится в одну строку (оператор `write`), а по завершении вывода очередной строки отдельно добавляется пустой оператор `writeln` для перехода на следующую строку.

```
var Mas: array [1..10,1..10] of integer;
. . . . .
for i := 1 to 10 do begin
  for j := 1 to 10 do
    writeln(Mas[i],' ');
  writeln;
  end;
end.
```

Обмен местами элементов массива

Производится аналогично обмену значений двух обычных переменных (обычно — при помощи дополнительной «буферной» переменной).

Обработка элементов массива (определение максимума/минимума, вычисление суммы, произведения, среднего и пр.)

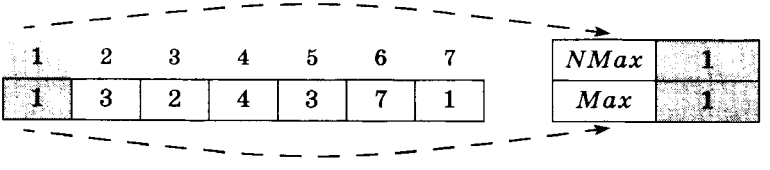
В цикле (для многомерного массива — во вложенных циклах) производится перебор элементов массива (полный или частичный — для фрагмента массива).

- **При поиске максимума/минимума** за предполагаемый максимум/минимум берется первый элемент массива либо константа, заведомо меньшая/большая любого элемента. Далее каждый очередной элемент массива сравнивается с предполагаемым максимумом/минимумом, и если этот элемент больше/меньше предполагаемого максимума/минимума, то значение этого элемента запоминается в качестве нового предполагаемого максимума/минимума. Дополнительно при этом в отдельной переменной (переменных) может перезапоминаться индекс (индексы) очередного предполагаемого максимума/минимума.

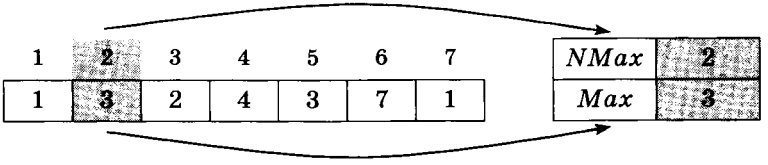
Поиск максимума	Поиск минимума

`Max := Mas[1];`
`NMax := 1;`
`for i := 2 to N do`
`  if Mas[i] > Max then`
`begin`
`    Max := Mas[i];`
`    NMax := i;`
`  end;`
`end;`
	`Min := Mas[1];`
	`NMin := 1;`
	`for i := 2 to N do`
	`  if Mas[i] < Min then`
	`begin`
	`    Min := Mas[i];`
	`    NMin := i;`
	`  end;`
	`end;`

Пример: поиск максимума в массиве (1,3,2,4,3,7,1).
1) первоначально:



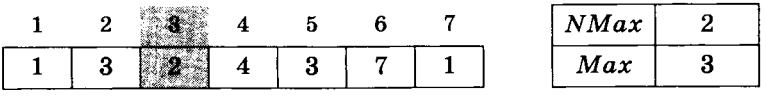
2) $i = 2$: $Mas[2] = 3$ больше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	2
Max	3

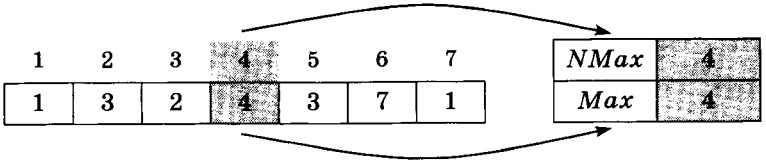
3) $i = 3$: $Mas[3] = 2$ меньше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	2
Max	3

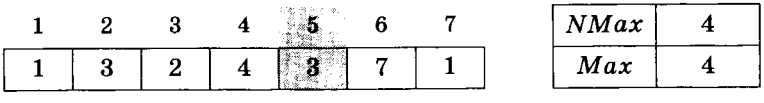
4) $i = 4$: $Mas[4] = 4$ больше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	4
Max	4

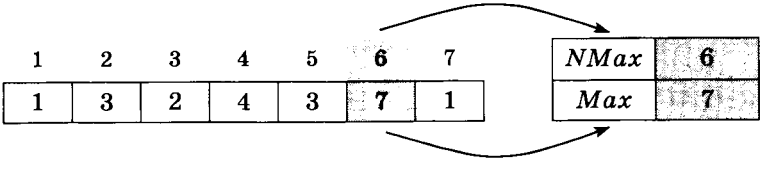
5) $i = 5$: $Mas[5] = 3$ меньше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	4
Max	4

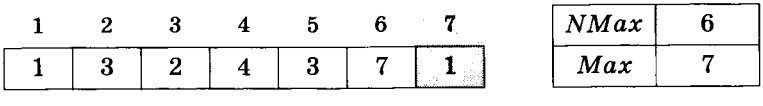
6) $i = 6$: $Mas[6] = 7$ больше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	6
Max	7

7) $i = 7$: $Mas[7] = 1$ меньше, чем Max:



1	2	3	4	5	6	7
1	3	2	4	3	7	1

NMax	6
Max	7

Результат:

NMax	6
Max	7

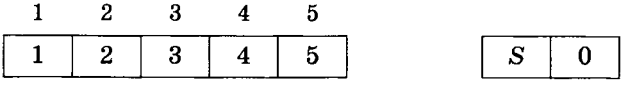
При вычислении суммы/произведения вначале переменной, выделенной для накопления суммы/произведения присваивается инициализационное значение (нуль — для суммы, единица — для произведения). Затем в цикле (либо вложенных циклах) выполняется перебор элементов массива и текущее значение суммы/произведения складывается/умножается на текущий элемент массива.

Вычисление суммы	Вычисление произведения

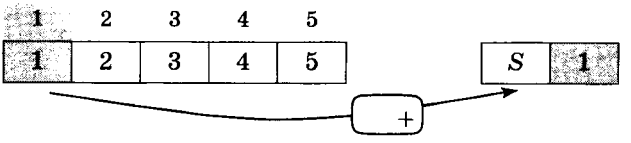
`S := 0;`
`for i := 1 to N do`
`  S := S + Mas[i];`
	`P := 1;`
	`for i := 1 to N do`
	`  P := P * Mas[i];`

Пример: вычисление суммы элементов массива (1,2,3,4,5).

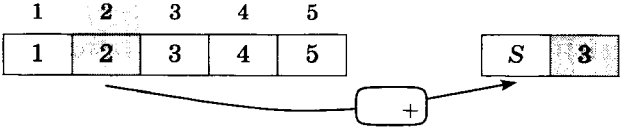
1) первоначально:



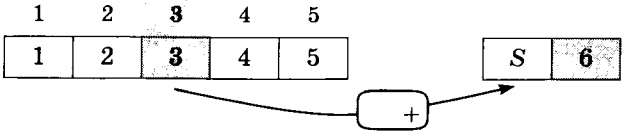
2) $i = 1$:



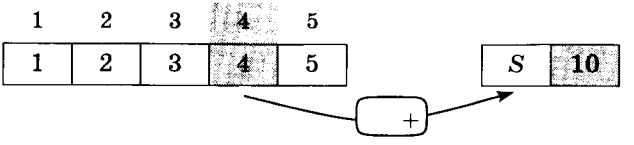
3) $i = 2$:



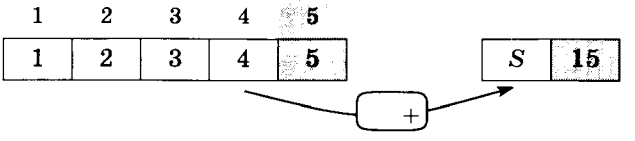
4) $i = 3$:



5) $i = 4$:



6) $i = 5$:



Результат:

S	15

- **При вычислении среднего значения** выполняется суммирование элементов массива, а затем полученная сумма делится на количество элементов в массиве.

```
S := 0;
for i := 1 to N do
  S := S + Mas[i];
Sredn := S / N;
```

- **При определении максимума/минимума, суммы, произведения, среднего значения элементов, удовлетворяющих заданному условию (например, только положительных)** дополнительно добавляется условный оператор с соответствующим условием, и требуемое действие (проверка и переприсваивание предполагаемого максимума/минимума, сложение, умножение) выполняется только при истинности этого условия. При вычислении среднего значения

также предусматривается отдельная переменная-счётчик, которая увеличивается на 1 каждый раз, когда к сумме добавляется очередной удовлетворяющий условию элемент массива, и после вычисления суммы она делится на значение этого счётчика (количество вошедших в сумму элементов).

В задаче на простой поиск минимума/максимума в качестве предполагаемого минимума/максимума проще всего брать первый элемент массива, а просмотр и сравнение их с предполагаемым минимумом/максимумом вести со второго элемента массива.

Если требуется поиск минимума/максимума среди элементов, удовлетворяющих заданным условиям, то формально брать в качестве предполагаемого минимума/максимума первый элемент уже нельзя: он может не удовлетворять заданным условиям.

Если из условий задачи известен возможный диапазон значений элементов массива, то в качестве предполагаемого *минимума* можно брать константу, заведомо *большую*, чем возможное самое большое значение элементов массива, удовлетворяющих заданному условию, а в качестве предполагаемого *максимума* брать константу, заведомо *меньшую*, чем возможное самое маленькое значение элементов массива, удовлетворяющих заданному условию.

Если из условий задачи невозможно определить диапазон допустимых значений элементов массива, то в качестве предполагаемого минимума/максимума может быть взята произвольная константа («заглушка»), заведомо *не удовлетворяющая* заданным условиям, а в операторе проверки условий переприсваивания предполагаемого минимума/максимума необходимо учесть необходимость замены этого «значения-заглушки» первым же элементом массива, удовлетворяющим условиям.

Пример 1: среднее арифметическое положительных элементов массива (1, −2, 3, −4, 6):

```
S := 0;
k := 0;
for i := 1 to N do
  if Mas[i] > 0 then begin
    S := S + Mas[i];
    k := k + 1;
  end;
Sredn := S / k;
```

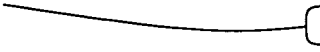
1) первоначально:

1	2	3	4	5
1	−2	3	−4	6

S	0
k	0

2) $i = 1$: $Mas[1] = 1$ — положительное число:

1	2	3	4	5
1	−2	3	−4	6



S	1
k	1

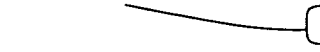
3) $i = 2$: $Mas[2] = -2$ — отрицательное число:

1	2	3	4	5
1	−2	3	−4	6

S	1
k	1

4) $i = 3$: $Mas[3] = 3$ — положительное число:

1	2	3	4	5
1	−2	3	−4	6



S	4
k	2

5) $i = 4$: $Mas[4] = -4$ — отрицательное число:

1	2	3	4	5
1	−2	3	−4	6

S	4
k	2

6) $i = 5$: $Mas[5] = 6$ — положительное число:

1	2	3	4	5
1	−2	3	−4	6

S	10
k	3

Результат:

S	10
k	3

$\Rightarrow Sredn = 10/3 = 3{,}3333$

Пример 2: минимум среди чётных положительных элементов массива (5, −4, −3, 6, 1, 4, 2) (известно, что значения элементов не превышают 100):

```
Min := 101;     // предполагаемый минимум больше
                // максимально возможного
                // значения элемента
for i := 1 to N do
  if (Mas[i] > 0) AND (Mas[i] mod 2 = 0) AND
     (Mas[i] < Min) then
        begin
          Min := Mas[i];
          NMin := i;
        end;
end;
```

1) первоначально:

1	2	3	4	5	6	7
5	−4	−3	6	1	4	2

NMin	–
Min	101

2) $i = 1$: $Mas[1] = 5$ — нечётное:

1	2	3	4	5	6	7
5	−4	−3	6	1	4	2

NMin	–
Min	101

3) $i = 2$: $Mas[2] = -4$ — чётное, но отрицательное:

1	**2**	3	4	5	6	7
5	**−4**	−3	6	1	4	2

NMin	–
Min	101

4) $i = 3$: $Mas[3] = -3$ — нечётное и отрицательное:

1	2	**3**	4	5	6	7
5	−4	**−3**	6	1	4	2

NMin	–
Min	101

5) $i = 4$: $Mas[4] = 6$ — чётное, положительное, меньше чем *Min*:

1	2	3	**4**	5	6	7
5	−4	−3	**6**	1	4	2

NMin	**4**
Min	**6**

6) $i = 5$: $Mas[5] = 1$ — нечётное:

1	2	3	4	5	6	7
5	−4	−3	6	1	4	2

NMin	4
Min	6

7) $i = 6$: $Mas[6] = 4$ — чётное, положительное, меньше чем *Min*:

1	2	3	4	5	6	7
5	−4	−3	6	1	4	2

NMax	6
Max	4

8) $i = 7$: $Mas[7] = 2$ — чётное, положительное, меньше чем *Min*:

1	2	3	4	5	6	7
5	−4	−3	6	1	4	2

NMax	7
Max	2

Результат:

NMin	7
Min	2

Пример 3: максимум среди нечётных положительных элементов массива (5, −4, −3, 7, 1, 3, 9), причём диапазон возможных значений элементов массива неизвестен:

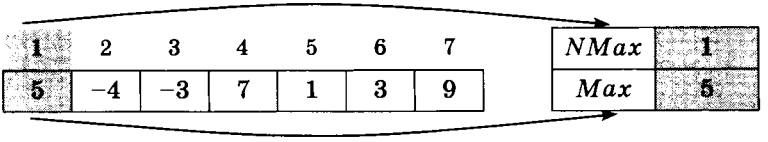
```
Max := 0; // поскольку требуется поиск
          // максимума среди положительных
          // элементов, нуль является
          // значением, заведомо меньшим любого
          // значения элемента массива,
          // удовлетворяющего условию
          // положительности
for i := 1 to N do
  if (Mas[i] > 0) AND (Mas[i] mod 2 <> 0) AND
     (Mas[i] > Max) then
        begin
          Max := Mas[i];
          NMax := i;
        end;
end;
```

1) первоначально:

1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

NMax	–
Max	0

2) $i = 1$: $Mas[1] = 5$ — нечётное, положительное, больше чем *Max*:



1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

NMax	1
Max	5

3) $i = 2$: $Mas[2] = -4$ — чётное:

1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

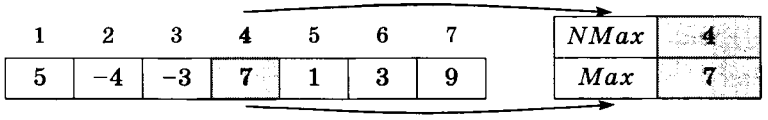
NMax	1
Max	5

4) $i = 3$: $Mas[3] = -3$ — нечётное, но отрицательное:

1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

NMax	1
Max	5

5) $i = 4$: $Mas[4] = 7$ — нечётное, положительное, больше чем *Max*:



1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

NMax	4
Max	7

6) $i = 5$: $Mas[5] = 1$ — нечётное, положительное, но меньше чем *Max*:

1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

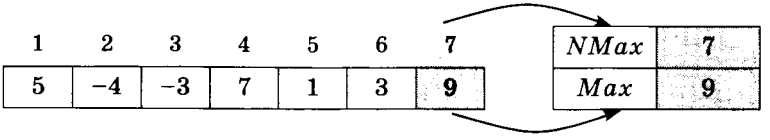
NMax	4
Max	7

7) $i = 6$: $Mas[6] = 3$ — нечётное, положительное, но меньше чем *Max*:

1	2	3	4	5	6	7
5	–4	–3	7	1	3	9

NMax	4
Max	7

8) $i = 7$: $Mas[7] = 9$ — чётное, положительное, больше чем Max:



1	2	3	4	5	6	7
5	−4	−3	7	1	3	**9**

NMax	7
Max	**9**

Результат:

NMin	7
Min	9

Пример 4: минимум среди положительных элементов массива (5, −6, −3, 9, 1, 3, 6), кратных трем, причём диапазон возможных значений элементов массива неизвестен:

Поскольку мы не знаем, какое возможно максимальное значение элементов массива, мы не можем указать, какое заведомо большее этого максимального значения число можно взять вместо предполагаемого минимума.

```
Min := 0;   // требуется поиск минимума среди
            // положительных элементов массива;
            // нуль не удовлетворяет этому
            // условию и потому может быть
            // выбран в качестве «значения-
            // заглушки» для исходного
            // присваивания предполагаемому
            // минимуму
for i := 1 to N do
  if (Mas[i] > 0) AND (Mas[i] mod 2 <> 0) AND
     ((Mas[i] < Min) OR (Min = 0)) then
            // переприсваивание Min выполняется
            // в двух случаях:
            // 1) если текущий элемент
            // удовлетворяет заданным условиям,
            // а предполагаемый минимум равен
            // «значению-заглушке»;
            // цель этой операции — подставить
            // в качестве предполагаемого
            // минимума первый же элемент
            // массива, удовлетворяющий
            // заданным условиям;
```

```
            // 2) если текущий элемент
            // удовлетворяет заданным условиям и
            // меньше предполагаемого минимума
        begin
          Min := Mas[i];
          NMin := i;
        end;
end;
```

1) первоначально:

1	2	3	4	5	6	7
5	−6	−3	9	1	3	6

NMin	–
Min	0

2) $i = 1$: $Mas[1] = 5$ — положительное, но не кратное 3:

1	2	3	4	5	6	7
5	−6	−3	9	1	3	6

NMin	–
Min	0

3) $i = 2$: $Mas[2] = -6$ — отрицательное:

1	**2**	3	4	5	6	7
5	**−6**	−3	9	1	3	6

NMin	–
Min	0

4) $i = 3$: $Mas[3] = -3$ — отрицательное:

1	2	**3**	4	5	6	7
5	−6	**−3**	9	1	3	6

NMin	–
Min	0

5) $i = 4$: $Mas[4] = 9$ — положительное, кратное 3; *Min* равно «значению-заглушке» — нулю:

1	2	3	**4**	5	6	7
5	−6	−3	**9**	1	3	6

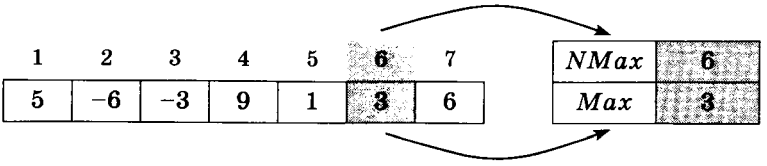
NMin	**4**
Min	**9**

6) $i = 5$: $Mas[5] = 1$ — положительное, но не кратное 3:

1	2	3	4	**5**	6	7
5	−6	−3	9	**1**	3	6

NMin	4
Min	9

7) $i = 6$: $Mas[6] = 3$ — положительное, кратное 3, меньше чем Min:



1	2	3	4	5	6	7
5	−6	−3	9	1	3	6

NMax	6
Max	3

8) $i = 7$: $Mas[7] = 6$ — положительное, кратное 3, но больше чем Min:

1	2	3	4	5	6	7
5	−6	−3	9	1	3	6

NMin	6
Min	3

Результат:

NMin	6
Min	3

Разбор типовых задач

Задача 1*. Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 100 — баллы учащихся выпускного класса за итоговый тест по информатике. Для получения положительной оценки за тест требовалось набрать не менее 20 баллов. Опишите на русском языке или на одном из языков программирования алгоритм, который позволяет найти и вывести минимальный балл среди учащихся, получивших за тест положительную оценку. Известно, что в классе хотя бы один учащийся получил за тест положительную оценку.

Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать часть из них.

Паскаль	Бейсик

const N = 30;
var
a: array [1..N] of
integer;
i, j, min: integer;
begin
for i := 1 to N do
readln(a[i]);
…
end.
	N = 30
	DIM A(N) AS INTEGER
	DIM I, J, MIN AS
	INTEGER
	FOR I = 1 TO N
	INPUT A(I)
	NEXT I
	…
	END

Си	**Естественный язык**

#include <stdio.h>
#define N 30
void main(void)
{int a[N];
int i, j, min;
for (i = 0; i < N; i++)
scanf("% d", &a[i]);
}
	Объявляем целочисленные переменные I, J, MIN. В цикле от 1 до 30 вводим элементы массива A с 1-го по 30-й.
	…

В качестве ответа вам необходимо привести фрагмент программы (или описание алгоритма на естественном языке), который должен находиться на месте многоточия. Вы можете записать решение также на другом языке программирования (укажите название и используемую версию языка программирования, например, Borland Pascal 7.0) или в виде блок-схемы. В этом случае вы должны использовать те же самые исходные данные и переменные, какие были предложены в условии (например, в образце, записанном на естественном языке).

Решение

Данная задача подобна поиску минимума в массиве, но с дополнительным условием: искать минимум надо только среди элементов, равных или превышающих 20.

Соответственно этому, в типовой алгоритм поиска минимума следует внести следующие изменения:

- поскольку в качестве предполагаемого минимума первоначально нельзя использовать первый элемент (он может оказаться не удовлетворяющим условию «20»), нужно в качестве первого значения предполагаемого минимума брать число, заведомо превышающее любое значение в массиве (в данном случае это может быть число 101);
- в оператор `if`, проверяющий, не является ли текущий элемент просматриваемого массива меньшим, чем предполагаемый минимум, надо добавить условие «текущий элемент больше или равен 20».

Полный текст программы:

```
const N = 30;
var a : array [1..N] of integer;
    i, min: integer; // переменная j не
                     // используется
begin
  for i := 1 to N do readln(a[i]);  // считали
                                    // массив
  min := 101;
  // в качестве исходного предполагаемого
  // минимума берется значение, превышающее
  // максимально возможное значение элемента
  // в массиве
  for i := 1 to N do // полный просмотр массива
    if (a[i] >= 20) and (a[i] < min) then
       min := a[i];
  // если текущий элемент удовлетворяет условию
  // (больше или равен 20) И этот элемент меньше
  // предполагаемого минимума, то этот элемент
  // запоминаем как новый предполагаемый
  // минимум
writeln(min); // вывод найденного минимума на
              // экран
end.
```

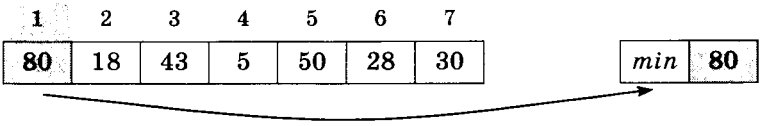
Пример для массива (80, 18, 43, 5, 50, 28, 30).

1) первоначально:

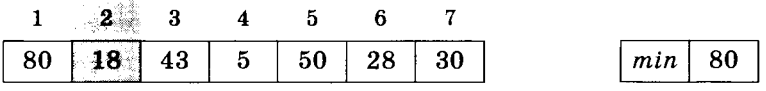
1	2	3	4	5	6	7
80	18	43	5	50	28	30

min	101

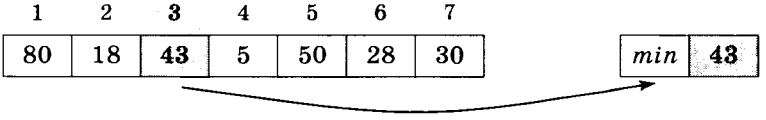
2) $i = 1$: $Mas[1] = 80$ — больше 20 и меньше, чем min:



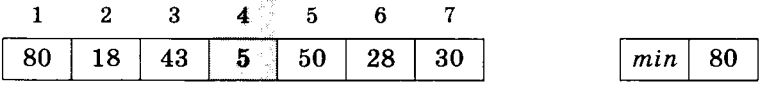
3) $i = 2$: $Mas[2] = 18$ — меньше 20:



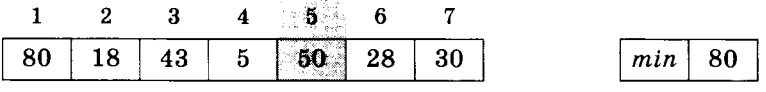
4) $i = 3$: $Mas[3] = 43$ — больше 20 и меньше, чем min:



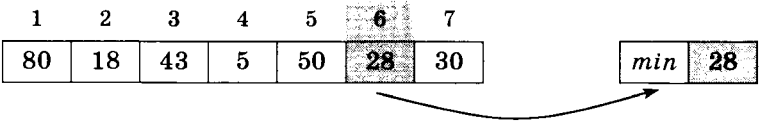
5) $i = 4$: $Mas[4] = 5$ — меньше 20:



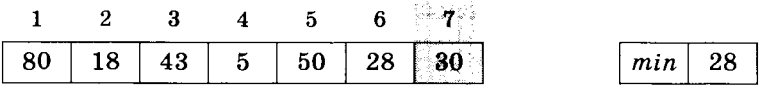
6) $i = 5$: $Mas[5] = 50$ — больше 20, но больше, чем min:



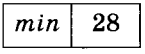
7) $i = 6$: $Mas[6] = 28$ — больше 20 и меньше, чем min:



8) $i = 7$: $Mas[7] = 30$ — больше 20, но больше, чем min:



Результат:



Задача 2*. Дан целочисленный массив из 30 элементов. Элементы массива могут принимать значения от 0 до 1000. Опишите на русском языке или на одном из языков программирования алгоритм, который позволяет подсчитать и вывести среднее арифметическое элементов массива, имеющих нечетное значение. Гарантируется, что в исходном массиве хотя бы один элемент имеет нечетное значение.

Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать часть из них.

Паскаль

```
const N = 30;
var a: array [1..N] of
    integer;
i, x, y: integer;
s: real;
begin
for i := 1 to N do
readln(a[i]);
...
end.
```

Бейсик

```
N = 30
DIM A(N) AS INTEGER
DIM I, X, Y AS INTEGER
DIM S AS SINGLE
FOR I = 1 TO N
INPUT A(I)
NEXT I
...
END
```

Си

```
#include <stdio.h>
#define N 30
void main(void)
{int a[N];
int i, x, y;
float s;
for (i = 0; i < N; i++)
scanf("%d", &a[i]);
...
}
```

Естественный язык

```
Объявляем массив A из
30 элементов.
Объявляем целочисленные
переменные I, X, Y.
Объявляем вещественную
переменную S.
В цикле от 1 до 30 вво-
дим элементы массива A
с 1-го по 30-й.
...
```

В качестве ответа вам необходимо привести фрагмент программы (или описание алгоритма на естественном языке), который должен находиться на месте многоточия. Вы можете записать решение также на другом

языке программирования (укажите название и используемую версию языка программирования, например, Borland Pascal 7.0) или в виде блок-схемы. В этом случае вы должны использовать переменные, аналогичные переменным, используемым в алгоритме, записанном на естественном языке, с учетом синтаксиса и особенностей используемого вами языка программирования.

Решение

Речь идёт о вычислении среднего арифметического элементов массива, удовлетворяющих заданному условию (здесь — нечётность). Для этого в цикле просмотра массива нужно подсчитать сумму только элементов, удовлетворяющих этому условию, и одновременно количество таких элементов.

Полный текст программы:

```
const N = 30;
var a : array [1..N] of integer;
    i, x, y: integer;
    s: real;
begin
  for i := 1 to N do readln(a[i]); // считали
                                   // массив
  x := 0;
  y := 0;
  // x — переменная для подсчета суммы
  // элементов, удовлетворяющих заданному
  // условию;
  // y — переменная для подсчета количества этих
  // элементов;
  // изначально обе переменные обнуляются
  for i := 1 to N do  // полный просмотр массива
    if (a[i] mod 2 = 1) then begin
  // если текущий элемент удовлетворяет условию
  // (нечетный), то
      x := x + a[i];
  // прибавляем этот элемент к накапливаемой
  // сумме
      y := y + 1;
  // и увеличиваем количество просуммированных
  // элементов на 1
  end;
```

```
  s := x / y;
  // по окончании цикла вычисляем среднее
  // арифметическое, деля полученную сумму
  // нечетных чисел на их количество
  writeln(s);  // вывод среднего арифметического
               // значения нечетных элементов
end.
```

Пример для массива (4, 65, 2, 3, 1).

1) первоначально:

1	2	3	4	5
4	65	2	3	1

x	0
y	0

2) $i = 1$: Mas[1] = 4 — чётное:

1	2	3	4	5
4	65	2	3	1

x	0
y	0

3) $i = 2$: Mas[2] = 65 — нечётное:

1	2	3	4	5
4	65	2	3	1

x	65
y	1

4) $i = 3$: Mas[3] = 2 — чётное:

1	2	3	4	5
4	65	2	3	1

x	65
y	1

5) $i = 4$: Mas[4] = 3 — нечётное:

1	2	3	4	5
4	65	2	3	1

x	68
y	2

6) $i = 5$: Mas[5] = 1 — нечётное:

1	2	3	4	5
4	65	2	3	1

x	69
y	3

Результат:

x	69
y	3

$\Rightarrow s = 69/3 = 23$

Задача 3*. Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от 0 до 1000. Опишите на русском языке или на одном из языков программирования алгоритм, позволяющий найти и вывести минимальное значение среди элементов массива, которые имеют чётное значение и не делятся на три. Гарантируется, что в исходном массиве есть хотя бы один элемент, значение которого чётно и не кратно трём.

Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но использовать все описанные переменные не обязательно.

Паскаль

```
const N = 20;
var a: array [1..N] of
    integer;
i, j, min: integer;
begin
for i := 1 to N do
readln(a[i]);
…
end.
```

Алгоритмический язык

```
алг
нач
цел N = 20
целтаб a[1:N]
цел i, j, MIN
нц для i от 1 до N
ввод a[i]
кц
...
кон
```

Бейсик

```
N = 20
DIM A(N) AS INTEGER
DIM I, J, MIN AS
    INTEGER
FOR I = 1 TO N
INPUT A(I)
NEXT I
…
END
```

Си

```
#include <stdio.h>
#define N 20
void main(void){
int a[N];
int i, j, min;
for (i = 0; i < N; i++)
scanf("% d", &a[i]);
…
}
```

Естественный язык

```
Объявляем массив A из 20 элементов.
Объявляем целочисленные переменные I, J, MIN.
В цикле от 1 до 20 вводим элементы массива A
с 1-го по 20-й.
…
```

В качестве ответа необходимо привести фрагмент программы (или описание алгоритма на естественном языке), который должен находиться на месте многоточия. Можно записать решение также на другом языке программирования (указав название и используемую версию языка программирования, например Borland Pascal 7.0) или в виде блок-схемы. В этом случае нужно использовать те же самые исходные данные и переменные, какие были предложены в условии (например, в образце, записанном на естественном языке).

Решение

Данная задача по смыслу решения аналогична задаче 1: здесь также необходимо искать минимум среди элементов массива, удовлетворяющих заданному условию (чётность и неделимость нацело на 3).

Изменения, которые нужно внести в типовой алгоритм поиска минимума в массиве:

- поскольку нельзя в качестве предполагаемого минимума первоначально использовать первый элемент (он может оказаться не удовлетворяющим заданному условию), в качестве первого значения предполагаемого минимума следует брать число, заведомо превышающее любое значение в массиве (в данном случае это может быть число 1001);
- в оператор `if`, проверяющий, не является ли текущий элемент просматриваемого массива меньшим, чем предполагаемый минимум, надо добавить условия «текущий элемент чётный» и «текущий элемент не делится нацело на 3».

Полный текст программы:

```
const N = 20;
var
   a: array [1..N] of integer;
   i, min: integer; // переменная j не
                    // используется
begin
  for i := 1 to N do readln(a[i]);     // считали
                                       // массив

min := 1001;
```

```
    // в качестве исходного предполагаемого
    // минимума берется значение, превышающее
    // максимально возможное значение элемента
    // в массиве
for i := 1 to N do     // полный просмотр массива
if (a[i] mod 2 = 0) and (a[i] mod 3 <> 0) and
(a[i] < min) then min := a[i];
    // если текущий элемент удовлетворяет
    // условиям: он четный И он не делится без
    // остатка на 3 И этот элемент меньше
    // предполагаемого минимума, то этот элемент
    // запоминаем как новый предполагаемый
    // минимум
writeln(min); // вывод найденного минимума на
              // экран
end.
```

Пример для массива (333, 80, 43, 60, 44, 3, 68).

1) первоначально:

1	2	3	4	5	6	7
333	80	43	60	44	3	68

min	**1001**

2) $i = 1$: $Mas[1] = 333$ — нечётное:

1	2	3	4	5	6	7
333	80	43	60	44	3	68

min	1001

3) $i = 2$: $Mas[2] = 80$ — чётное, не делится на 3, меньше, чем *min*:

1	**2**	3	4	5	6	7
333	**80**	43	60	44	3	68

min	**80**

4) $i = 3$: $Mas[3] = 43$ — нечётное:

1	2	**3**	4	5	6	7
333	80	**43**	60	44	3	68

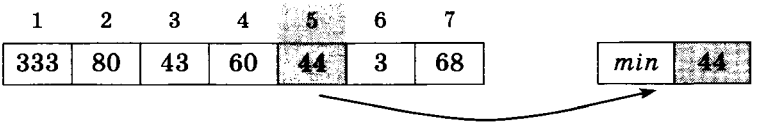
min	80

5) $i = 4$: $Mas[4] = 60$ — чётное, но делится на 3:

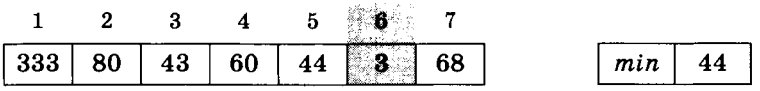
1	2	3	**4**	5	6	7
333	80	43	**60**	44	3	68

min	80

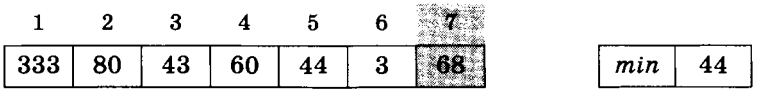
6) $i = 5$: $Mas[5] = 44$ — чётное, не делится на 3, меньше, чем min:



7) $i = 6$: $Mas[6] = 3$ — нечётное:



8) $i = 7$: $Mas[7] = 68$ — чётное, не делится на 3, но больше, чем min:



Результат:

min	44

Задача 4. Дан целочисленный массив из 20 элементов. Элементы массива могут принимать целые значения от 0 до 1000. Опишите на одном из языков программирования алгоритм, позволяющий найти и вывести порядковый номер минимального значения среди нечетных положительных элементов массива.

Гарантируется, что в исходном массиве есть хотя бы один элемент, значение которого нечетно и положительно. Если в массиве имеется несколько равных друг другу элементов, значение которых является минимальным среди нечётных положительных чисел, то нужно вывести номер последнего такого элемента (т.е. максимальный из найденных номеров элементов — минимумов).

Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но использовать все описанные переменные не обязательно.

Паскаль

```
const N = 20;
var
a: array [1..N] of
    integer;
i, j, min, nmin:
    integer;
begin
for i := 1 to N do
readln(a[i]);
…
end.
```

Алгоритмический язык

```
алг
нач
цел N = 20
целтаб a[1:N]
цел i, j, MIN, NMIN
нц для i от 1 до N
ввод a[i]
кц
...
кон
```

Бейсик

```
N = 20
DIM A(N) AS INTEGER
DIM I, J, MIN, NMIN AS
     INTEGER
FOR I = 1 TO N
INPUT A(I)
NEXT I
…
END
```

Си

```
#include <stdio.h>
#define N 20
void main(void){
int a[N];
int i, j, min, nmin;
for (i = 0; i < N; i++)
scanf("% d", &a[i]);
…
}
```

Естественный язык

```
Объявляем массив A из 20 элементов.
Объявляем целочисленные переменные I, J, MIN,
NMIN.
В цикле от 1 до 20 вводим элементы массива A
с 1-го по 20-й.
…
```

В качестве ответа необходимо привести фрагмент программы (или описание алгоритма на естественном языке), который должен находиться на месте многоточия.

Решение

Особенность этой задачи — в том, что при наличии в массиве *нескольких* минимальных элементов надо запомнить и вывести именно конкретный номер элемента — наибольший среди номеров таких элементов. Поэтому нужно принять меры, чтобы проигнорировать

значения минимумов, номера которых меньше последнего такого элемента.

Способ 1

В условии, проверяющем, является ли текущий элемент просматриваемого массива меньшим, чем предполагаемый минимум, надо использовать *знак нестрогого неравенства*. Тогда для встреченных далее значений, равных текущему минимуму, номер элемента *nmin* будет перезапоминаться. В итоге последним запомненным будет максимальный из номеров таких элементов.

```
const N = 20;
var a: array [1..N] of integer;
    i, j, min, nmin: integer;
begin
  for i := 1 to N do
    readln(a[i]);
  min := 1001;
  // значение, заведомо большее максимального
  // значения элементов
  for i := 1 to N do
    if (a[i] > 0) AND (a[i] mod 2 <> 0) AND
(a[i] <= min) then
      min := a[i];
      nmin := i;
    end;
  end;
  writeln('Номер последнего минимального
          элемента: ', nmin);
end.
```

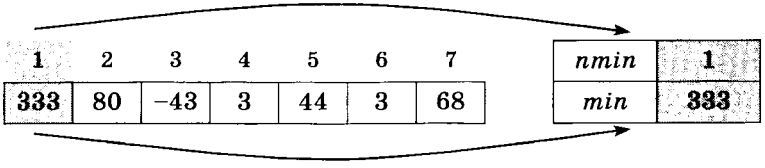
Пример для массива (333, 80, −43, 3, 44, 3, 68).

1) первоначально:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	–
min	1001

2) $i = 1$: $Mas[1] = 333$ — положительное, нечётное, меньше чем *min*:



1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	1
min	333

3) $i = 2$: $Mas[2] = 80$ — положительное, но чётное:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

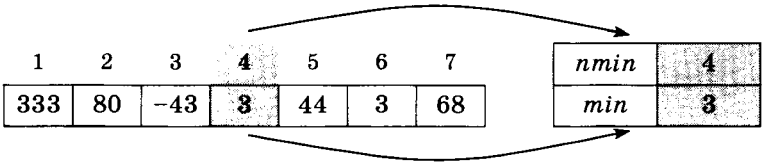
nmin	1
min	333

4) $i = 3$: $Mas[3] = -43$ — отрицательное:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	1
min	333

5) $i = 4$: $Mas[4] = 3$ — положительное, нечётное, меньше чем *min*:



1	2	3	4	5	6	7
333	80	−43	3	44	3	68

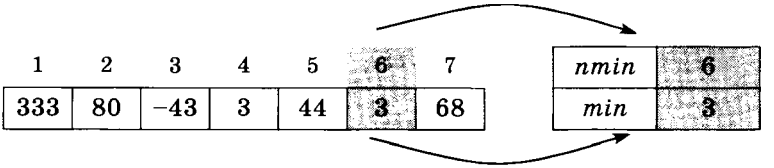
nmin	4
min	3

6) $i = 5$: $Mas[5] = 44$ — положительное, но чётное:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	4
min	3

7) $i = 6$: $Mas[6] = 3$ — положительное, нечётное, ***равное min***:



1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	6
min	3

8) $i = 7$: $Mas[7] = 68$ — положительное, но чётное:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	6
min	3

Результат:

nmin	6
min	3

Способ 2

Можно использовать знак строгого неравенства, но просмотр массива вести с конца. Тогда первый из встреченных равных друг другу минимумов будет запомнен, а все расположенные в массиве левее него такие минимумы будут проигнорированы (как равные, но не меньшие).

```
const N = 20;
var a: array [1..N] of integer;
    i, j, min, nmin: integer;
begin
  for i := 1 to N do
    readln(a[i]);
  min := 1001; // значение, заведомо большее
               // максимального значения
               // элементов
  for i := N downto 1 do
    if (a[i] > 0) AND (a[i] mod 2 <> 0) AND
       (a[i] < min) then
       min := a[i];
       nmin := i;
    end;
  end;
  writeln('Номер последнего минимального
           элемента: ', nmin);
end.
```

Пример для массива (333, 80, −43, 3, 44, 3, 68).

1) первоначально:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	–
min	1001

2) $i = 7$: $Mas[7] = 68$ — положительное, но чётное:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	–
min	1001

3) $i = 6$: $Mas[2] = 3$ — положительное, нечётное, меньше чем *min*:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	6
min	3

4) $i = 5$: $Mas[5] = 44$ — положительное, но чётное:

1	2	3	4	**5**	6	7
333	80	−43	3	**44**	3	68

nmin	6
min	3

5) $i = 4$: $Mas[4] = 3$ — положительное, нечётное, равное *min* (но не меньшее!):

1	2	3	**4**	5	6	7
333	80	−43	**3**	44	3	68

nmin	6
min	3

6) $i = 3$: $Mas[3] = -43$ — отрицательное:

1	2	**3**	4	5	6	7
333	80	**−43**	3	44	3	68

nmin	6
min	3

7) $i = 2$: $Mas[2] = 80$ — положительное, но чётное:

1	**2**	3	4	5	6	7
333	**80**	−43	3	44	3	68

nmin	6
min	3

8) $i = 1$: $Mas[1] = 333$ — положительное, нечётное, но большее чем *min*:

1	2	3	4	5	6	7
333	80	−43	3	44	3	68

nmin	6
min	3

Результат:

nmin	6
min	**3**

B6, B14 Процедуры и функции

Конспект

Подпрограмма

Подпрограмма — это поименованная или каким-либо иным образом обозначенная часть программы, которая может быть многократно вызвана из разных частей

основной программы для выполнения неких «типовых» вычислений, в том числе для различных исходных данных.

Идея здесь состоит в следующем. Предположим, что некоторая часть алгоритма по своей сути повторяется в программе несколько раз. Так, в комбинаторике при вычислении количества *сочетаний* из n элементов по k (подмножеств из k элементов, взятых произвольно из множества n элементов и различающихся хотя бы одним элементом без учёта порядка их следования) используется формула:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

В ней трижды надо вычислять факториал для трёх разных исходных значений. Тогда вычисление факториала путём выполняемых в цикле умножений можно оформить в виде подпрограммы, а затем обращаться к ней, каждый раз передавая этой подпрограмме требуемое исходное значение.

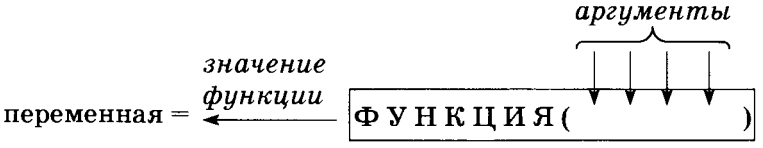
Иногда в виде подпрограмм выделяют фрагменты программного кода, выполняющие какие-либо типовые действия, — например, реализующие какие-то элементы интерфейса с пользователем. Это позволяет создавать отдельные *библиотеки подпрограмм*, облегчая этим работу программисту: ему уже не нужно отвлекаться на реализацию подобных «мелочей» и можно сосредоточиться непосредственно на сути решаемой вычислительной задачи, а библиотека готовых подпрограмм просто подключается к созданному листингу перед его компиляцией (или во время неё). А в современных ОС (например, в Windows) такие подпрограммы, уже откомпилированные в файлы DLL, являются основой функционирования как самой ОС, так и работающих в её среде прикладных программ и обеспечивают унификацию компонентов интерфейса.

Наконец, деление исходного сложного алгоритма на подпрограммы позволяет обеспечить его структуризацию, облегчить понимание и разработку программного

продукта, разделить исходную задачу на более простые подзадачи, реализуя тем самым *принцип нисходящего программирования* («от общего к частному»).

Функция

Это одна из двух возможных разновидностей подпрограмм. Её отличительная особенность состоит в том, что функция, принимая на вход любое количество исходных данных (аргументов), может возвращать только одно значение, которое передается как значение самой этой функции.



Для работы с **подпрограммами-функциями** в языке Паскаль (и аналогично — в большинстве других современных процедурных языков высокого уровня) требуется знание следующих особенностей.

1. Подпрограмма-функция должна быть *описана* в начале программы — после объявления используемых в ней *глобальных* констант, меток и переменных, но до собственно текста основной программы, заключенного в операторные скобки BEGIN и END (для наглядности эти слова, обрамляющие текст основной программы, записаны прописными буквами).

2. Описание подпрограммы-функции начинается со строки

```
Function <имя функции>(<арг1>, <арг2>, ... :
<тип1>; <арг3>, <арг4>, ... : <тип2>; ...) :
<тип результата>;
```

Здесь:

- `Function` — зарезервированное слово, указывающее транслятору, что далее идёт описание подпрограммы-функции;
- `<имя функции>` — идентификатор, выбираемый по тем же правилам, что и для имён переменных;

- сразу после имени функции в скобках записывается перечень передаваемых в эту функцию *формальных параметров* — имён переменных, которые далее будут использоваться при вычислениях, выполняемых в этой функции. При этом формальные параметры группируются через запятую в блоки одинаковых типов, а обозначения их типов записываются после списка параметров через двоеточие, как принято в Паскале при определении переменных; блоки параметров различного типа записываются через точку с запятой;
- после закрывающей скобки, также через двоеточие, записывается тип результата, который будет возвращать функция.

Пример:

`Function F(x : real) : real;` — объявляется подпрограмма-функция с именем *F*, которая принимает на вход одно действительное (тип `real`) число *x* и возвращает также действительное (`real`, записанное после скобок) значение результата.

3. После строки определения функции записывается её программный код. Правила его записи аналогичны правилам записи основной программы:

- сначала может присутствовать раздел описания *локальных* констант и переменных; эти константы и переменные действуют только в пределах данной функции и могут иметь такие же имена, что и локальные переменные, объявленные в других подпрограммах (транслятор их не путает);
- далее между операторными скобками `begin` и `end` записывается текст подпрограммы.

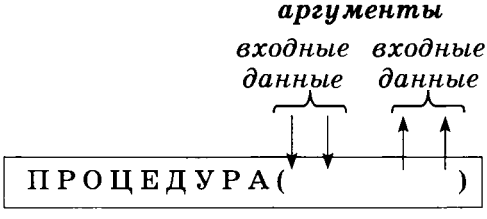
4. В конце текста подпрограммы значение, которое функция должна возвращать в качестве результата, нужно обязательно присвоить специальной переменной, имя которой совпадает с именем самой функции (причём отдельно объявлять эту переменную *не нужно*).

5. В тексте основной программы вызов функции производится следующим образом:

- записывается имя функции, а после него в скобках через запятую — список *фактических параметров* — константы, имена переменных, выражения, а также другие вызовы функций, значения которых нужно передать в функцию для выполнения вычислений. При этом количество и типы таких фактических параметров должны соответствовать указанным в объявлении функции формальным параметрам. Каждый фактический параметр при вызове функции записывается в соответствующий ему по порядку следования в списке формальный параметр и тем самым попадает в выполняемые функцией вычисления. При этом если для формального параметра указан соответствующий составной тип, объявленный в разделе `type` основной программы, то в подпрограмму может быть передан, например, массив целиком;
- поскольку функция возвращает значение, её запись (вместе с фактическими параметрами в скобках) должна быть либо включена в состав выражения, либо записана после имени переменной и следующего за ней оператора присваивания, либо указана как параметр в процедуре вывода на экран, записи в файл и пр. Иначе возвращённое функцией значение будет потеряно.

Процедура

Отличительная особенность **подпрограммы-процедуры** (или просто — **процедуры**) состоит в том, что она (в отличие от функции) может как принимать на вход, так и возвращать любое количество данных, причём все они записываются в качестве аргументов процедуры.



Для работы с процедурами в языке Паскаль (и в большинстве других языков высокого уровня) требуется знать следующее.

1. Процедура должна быть *описана* в начале программы — после объявления используемых в ней *глобальных* констант, меток и переменных, но до собственно текста основной программы, заключённого в операторные скобки BEGIN и END.

2. Описание подпрограммы-функции начинается со строки

```
Procedure <имя процедуры>(<арг1>, <арг2>, ... :
<тип1>; <арг3>, <арг4>, ... : <тип2>; ...) :
<тип результата>;
```

Здесь:

- Procedure — зарезервированное слово, указывающее транслятору, что далее идёт описание подпрограммы-процедуры;
- <имя процедуры> — идентификатор, выбираемый по тем же правилам, что и для имён переменных;
- сразу после имени функции в скобках записывается перечень передаваемых в эту функцию *формальных параметров* — имён переменных, которые будут использоваться при вычислениях, выполняемых в процедуре, и имён переменных, в которые процедура должна записать полученные результаты. Формальные параметры (как и в функциях) группируются через запятую в блоки одинаковых типов, а обозначения их типов записываются после списка параметров через двоеточие; блоки параметров различного типа записываются через точку с запятой;
- в процедуру, кроме обычных значений (числовых или символьных), можно передавать данные составных типов (массивы, записи и пр.). Для этого надо объявить такой составной тип, как *пользовательский* в разделе type в начале программы, далее объявить в разделе var соответствующий *экземпляр* данных как переменную этого составного типа, а далее при определении процедуры указать для соот-

ветствующих формальных параметров этот составной (пользовательский) тип.

Пример:

`Procedure Pr1(L : integer; var R : atype);` — объявляется процедура с именем `Pr1`, которая работает с целым (тип `integer`) числом `L` и массивом `R`, тип которого (`atype`) был ранее определён в разделе `type`:

```
type
    atype = array [1..L] of integer;
```

При этом параметры процедуры могут передаваться в нее как значения или как ссылки. В чем заключается различие между этими двумя способами передачи данных, мы рассмотрим чуть позже. Пока же отметим, что составные данные (в том числе массивы) обычно передаются как ссылки, а признаком передачи ссылки в процедуру является запись служебного слова `var` перед именем формального параметра (как и сделано в данной программе для передачи в процедуру массива `R`).

3. После строки определения процедуры записывается её программный код. Правила его записи аналогичны правилам записи основной программы:

- сначала может присутствовать раздел описания *локальных* констант и переменных, действующих только в пределах данной процедуры;
- далее между операторными скобками `begin` и `end` записывается текст подпрограммы.

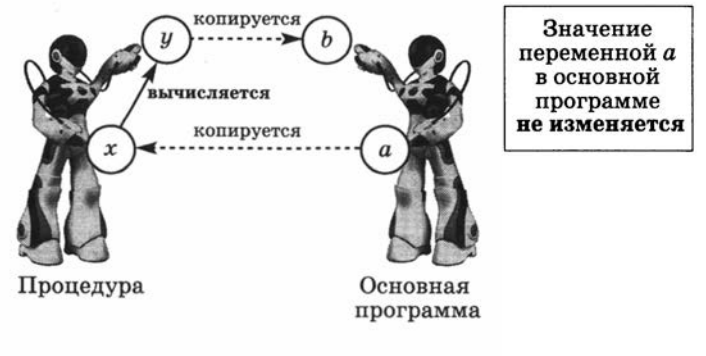
4. Результаты вычислений, которые процедура должна вернуть в основную программу, должны быть записаны в качестве значения соответствующего формального параметра.

5. В тексте основной программы вызов процедуры производится следующим образом:

- записывается имя процедуры, а после него в скобках через запятую — список *фактических параметров* — константы, имена переменных, выражения, а также другие вызовы процедур или функций, значения которых нужно передать в процедуру для выполнения вычислений. Переменные, в которые процедура должна вернуть результаты вычислений,

также записываются как её фактические параметры (и это должны быть переменные, объявленные в основной программе!). Количество и типы всех фактических параметров должны соответствовать указанным в объявлении процедуры формальным параметрам. При этом если параметр передаётся как значение, то это значение при вызове подпрограммы будет *скопировано* в соответствующий формальный параметр, так что если процедура, выполняя вычисления, изменит значение этого формального параметра, то это никак не повлияет на значение «фактической» переменной в основной программе. А вот если параметр передаётся как ссылка, то в процедуру попадёт информация о расположении в оперативной памяти компьютера самих исходных данных (скажем, массива), и тогда все изменения, которые процедура совершит с таким формальным параметром, будут отражены в самом исходном массиве. Иными словами, передавая массив (или другой экземпляр составных данных) в процедуру по ссылке, мы отдаём процедуре всю «власть» выполнять любые изменения с этим массивом. Соответственно, для возвращения результата изменений параметра, переданного как ссылка, не требуется предусматривать какой-то отдельный параметр: все эти изменения сразу производятся в исходном экземпляре данных.

Передача данных как значения



Передача данных как ссылки

Значение переменной M — экземпляра массива — процедура **изменяет** прямо в основной программе

- поскольку процедура возвращает результаты своей работы через свои параметры, она записывается как отдельная команда программы, а не как составная часть выражения или оператора присваивания.

Разбор типовых задач

Задача 1. Определить, какое число будет напечатано в результате работы следующей программы:

```
Program A14;
Uses crt;
Var d, a, b, t, M, R : real;
Function F(x : real) : real;
 begin
  F := (x + 2) * (4 - x);
 end;
BEGIN
 a := -2; b := 4;
 d := 0.1;
 t := a; M := a; R := F(a);
 while t <= b do
  begin
   if (F(t) > R) then
    begin
     M := t;
     R := F(t);
    end;
   t := t + d;
  end;
 write(M);
END.
```

Решение

В тексте программы (если не считать подключения модуля `crt` в разделе `uses`) вначале объявлен целый ряд переменных (раздел `var`).

Затем следует описание подпрограммы-функции с именем `F`, которой должно передаваться одно исходное значение — действительное число (формальный параметр `x`, который затем используется в качестве переменной в тексте подпрограммы) и которая должна возвращать действительное значение. Что именно вычисляется в этой подпрограмме, пока подробно не рассматривается.

После подпрограммы записан текст основной программы (между `BEGIN` и `END`).

Чтобы решить эту задачу нужно произвести полную трассировку используемых в программе переменных. Для этого составляется таблица трассировки. Следует особо отметить, что каждый раз, когда встречается в основной программе вызов функции, нужно переходить к тексту её описания и выполнять соответствующие команды до завершающего текст подпрограммы-функции слова `end`. При этом заданные в вызове функции фактические параметры автоматически переписываются в таблицу в качестве значений соответствующих им формальных параметров (столбец формального параметра, равно как и команды подпрограммы-функции, в таблице выделен серым фоном ячеек). Для наглядности значения переменных, изменяемые в результате действия текущей команды, выделены жирным шрифтом.

Оператор	*d*	*a*	*b*	*t*	*M*	*R*	*x*	F()	Примечание
`a := -2; b := 4; d := 0.1;`	**0.1**	**−2**	**4**						
`t := a; M := a;`	**0.1**	**−2**	**4**	**−2**	**−2**				
`R := F(a);`	**0.1**	**−2**	**4**	**−2**	**−2**		**−2**		Вызов функции
`F := (x + 2) * (4 - x);`	**0.1**	**−2**	**4**	**−2**	**−2**		**−2**	**0**	
`R := F(a);`	**0.1**	**−2**	**4**	**−2**	**−2**	**0**	**−2**		Значение `F` (результат выполнения функции) при выходе из функции в основную программу записывается в переменную `R`
`while t <= b do`	**0.1**	**−2**	**4** ($t \leqslant b$)	**-2**	**−2**	**0**	**−2**		Условие цикла истинно — цикл выполняется
`if (F(t) > R) then`	**0.1**	**−2**	**4**	**−2**	**−2**	**0**	**−2**		Вызов функции
`F := (x + 2) * (4 - x);`	**0.1**	**−2**	**4**	**−2**	**−2**	**0**	**−2**	**0**	
`if (F(t) > R) then`	**0.1**	**−2**	**4**	**−2**	**−2**	**0** ($F() = R$)	**−2**	**0**	Условие не выполнено — ветвь `then` пропускается
`t := t + d;`	**0.1**	**−2**	**4**	**−1.9**	**−2**	**0**	**−2**		
`while t <= b do`	**0.1**	**−2**	**4** ($t \leqslant b$)	**−1.9**	**−2**	**0**	**−2**		Условие цикла истинно — цикл выполняется
`if (F(t) > R) then`	**0.1**	**−2**	**4**	**−1.9**	**-2**	**0**	**−1.9**		Вызов функции

F := (x + 2) * (4 - x);	0.1	−2	4	−1.9	−2	0	−1.9	0.59	
if (F(t) > R) then	0.1	−2	4	−1.9	−2	0 $F() > R$	−1.9	0.59	Условие выполнено — выполняется ветвь then
M := t;	0.1	−2	4	−1.9	−1.9	0	−1.9		
R := F(t);	0.1	−2	4	−1.9	−1.9	0	−1.9		Вызов функции
F := (x + 2) * (4 - x);	0.1	−2	4	−1.9	−1.9	0	−1.9	0.59	
R := F(t);	0.1	−2	4	−1.9	−1.9	0.59	−1.9	0.59	
t := t + d;	0.1	−2	4	−1.8	−1.9	0.59	−1.9		
while t <= b do	0.1	−2	4 $t \leqslant b$	−1.8	−1.9	0.59	−1.9		Условие цикла истинно — цикл выполняется
if (F(t) > R) then	0.1	−2	4	−1.8	−1.9	0.59	−1.8		Вызов функции
F := (x + 2) * (4 - x);	0.1	−2	4	−1.8	−1.9	0.59	−1.8	1.16	
if (F(t) > R) then	0.1	−2	4	−1.8	−1.9	0.59 $F() > R$	−1.8	1.16	Условие выполнено — выполняется ветвь then
M := t;	0.1	−2	4	−1.8	−1.8	0.59	−1.8		
R := F(t);	0.1	−2	4	−1.8	−1.8	0.59	−1.8		Вызов функции
F := (x + 2) * (4 - x);	0.1	−2	4	−1.8	−1.8	0.59	−1.8	1.16	
R := F(t);	0.1	−2	4	−1.8	−1.8	1.16	−1.8	1.16	
t := t + d;	0.1	−2	4	−1.7	−1.8	1.16	−1.8		

Оператор	*d*	*a*	*b*	*t*	*M*	*R*	*x*	F()	Примечание
`while t <= b do`	0.1	−2	4 ($t \leqslant b$)	−1.7	−1.8	1.16	−1.8		Условие цикла истинно — цикл выполняется
`if (F(t) > R) then`	0.1	−2	4	−1.7	−1.8	1.16	−1.7		Вызов функции
`F := (x + 2) * (4 - x);`	0.1	−2	4	−1.7	−1.8	1.16	−1.7	1.71	
`if (F(t) > R) then`	0.1	−2	4	−1.7	−1.9	1.16 ($F() > R$)	−1.7	1.71	Условие выполнено — выполняется ветвь `then`
`M := t;`	0.1	−2	4	−1.7	−1.7	1.16	−1.7		
`R := F(t);`	0.1	−2	4	−1.7	−1.7	1.16	−1.7		Вызов функции
`F := (x + 2) * (4 - x);`	0.1	−2	4	−1.7	−1.7	1.16	−1.7	1.71	
`R := F(t);`	0.1	−2	4	−1.7	−1.7	1.71	−1.7	1.71	
`t := t + d;`	0.1	−2	4	−1.6	−1.7	1.71	−1.7		

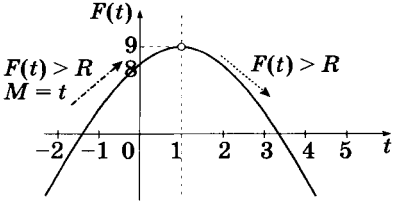
и т.д.

Выполнение полной трассировки такой программы — дело сложное и долгое. Однако можно выявить следующую закономерность: пока с увеличением значения t получаемое значение $F(t)$ также увеличивается (в результате чего это значение каждый раз оказывается больше предыдущего значения $F(t)$, запомненного в переменной R) текущее значение t заносится в интересующую переменную M.

Если выполнить перемножение скобок в выражении, записанном в составе функции F, то получается квадратное уравнение:

$(x + 2) \cdot (4 - x) = 4x + 8 - x^2 - 2x = -x^2 + 2x + 8$.

Поскольку перед x^2 стоит знак минуса, ветви параболы, соответствующей этому уравнению, будут направлены вниз. Вычисления производятся по левой ветви этой параболы снизу вверх. Очевидно, что рано или поздно они «доберутся» до вершины параболы, после чего с ростом t начнётся «движение» по её второй ветви уже сверху вниз. Тогда условие $F(t) > R$ перестанет выполняться (это произойдёт, когда будет пройдена вершина параболы и при переходе к следующему же после этого значению t), а значит, прекратится и запись текущих значений t в переменную M. Так будет до самого конца цикла изменения t.



Остаётся найти координату вершины параболы t и проследить работу программы вблизи этого значения данной переменной. Для этого можно использовать тот факт, что в вершине параболы значение производной равно нулю: $-2x + 2 = 0$. Тогда $x = 1$. Значит, искомое значение переменной t равно 1. Трассировка продолжается с него.

Оператор	*d*	*a*	*b*	*t*	*M*	*R*	*x*	F()	Примечание
`while t <= b do`	0.1	−2	4	1		8.99			Условие цикла истинно — цикл выполняется; значения переменных *M* и *x* мы пока не знаем; значение *R* равно *F*() от предыдущего значения *t* (0,9)
			$t \leqslant b$						
`if (F(t) > R) then`	0.1	−2	4	1		8.99	1		Вызов функции
`F := (x + 2) * (4 - x);`	0.1	−2	4	1		8.99	1	9	
`if (F(t) > R) then`	0.1	−2	4	1		8.99	1	9	Условие выполнено — выполняется ветвь `then`
						$F() < R$			
`M := t;`	0.1	−2	4	1	1	8.99	1		
`R := F(t);`	0.1	−2	4	1	1	8.99	1		Вызов функции
`F := (x + 2) * (4 - x);`	0.1	−2	4	1	1	8.99	1	9	
`R := F(t);`	0.1	−2	4	1	1	9	1	9	
`t := t + d;`	0.1	−2	4	1.1	1	9	1		
`while t <= b do`	0.1	−2	4	1.1	1	9	1		Условие цикла истинно — цикл выполняется
			$t \leqslant b$						
`if (F(t) > R) then`	0.1	−2	4	1.1	1	9	1.1		Вызов функции
`F := (x + 2) * (4 - x);`	0.1	−2	4	1.1	1	9	1.1	8.99	
`if (F(t) > R) then`	0.1	−2	4	1.1	1	9	1	8.99	Условие **не выполнено** — ветвь `then` пропускается. **И так будет до конца цикла изменения *t***
						$F() < R$			

Итак, последнее возможное изменение значения переменной M уже произошло, и при этом M стала равна 1.

Ответ: в результате работы данной программы будет выведено число **1** (вариант ответа №1).

Задача 2*. Определите, какое число будет напечатано в результате работы следующей программы (для вашего удобства программа представлена на четырёх языках):

Бейсик

```
Module A14
   Sub Main()
    Dim d, a, b, t, M, R
As Double
     a = -3 : b := 3
     d = 0.1
     t = a : M = a : R = F(a)
      While t < b
          If F(t) < R Then
       M = t
       R = F(t)
       End If
       t = t + d
      End While
           Console.Write(M)
          End Sub

Function F(ByVal x As Double)
As Double
Return (x - 1) * (x - 3)
        End Function

End Module
```

Паскаль

```
Program A14;
Uses crt;
Var d,a,b,t,M,R: real;
Function F(x : real): real;
   begin
     F := (x - 1) * (x - 3);
   end;
BEGIN
   a := -3; b := 3;
   d := 0.1;
   t := a; M := a;
   R := F(a);
   while t < b do
     begin
       if (F(t) < R) then
          begin
             M := t;
             R := F(t);
          end;
        t := t + d;
     end;
   write(M);
END.
```

Окончание табл.

Си	Алгоритмический язык

`#include <stdio.h>`
`double F(double x)`
`{`
`return (x - 1) * (x - 3);`
`}`
`void main()`
`{`
`double d, a, b, t, M, R;`
`  a = -3; b = 3;`
`  d = 0.1;`
`  t = a; M = a; R = F(a);`
`  while (t < b) {`
`  if ( F(t) < R ) {`
`  M = t; R = F(t);`
`  }`
`  t = t + d;`
`  }`
`printf("%f", M);`
`}`
	алг A14
	нач
	вещ d, a, b, t, M, R
	a := -3; b := 3
	d := 0.1
	t := a; M := a; R := F(a)
	нц пока t < b
	если F(t) < R
	то
	M := t; R := F(t)
	все
	t := t + d
	кц
	вывод M
	кон
	алг вещ F(вещ x)
	нач
	знач := (x - 1) * (x - 3)
	кон

1) −1 2) 2 3) −3 4) 24

Решение

Видя сходство предлагаемого текста программы с разобранным в предыдущей задаче, можно упростить решение.

Из текста программы извлекается «ключевая» информация об обрабатываемом графике функции:

- диапазон изменения аргумента функции $[a, b]$ — от −3 до 3;
- шаг изменения аргумента функции $d - 0{,}1$;
- формула, определяющая функцию: $F = (x - 1) \cdot (x - 3)$.

Раскрыв в записи этой функции скобки, получается квадратичная функциея $(x^2 - 4x + 4)$, график которой представляет собой параболу. При этом «нули» этой функции (точки пересечения её графика с осью абсцисс) нетрудно найти из исходной записи функции:
$(x - 1) \cdot (x - 3) = 0 \Rightarrow x_1 = 1, x_2 = 3$.

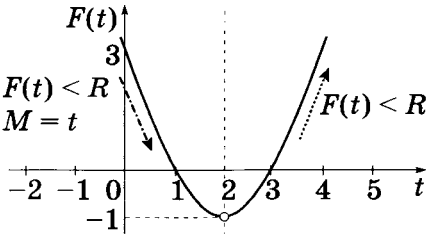
Чтобы определить направление ветвей параболы, нужно определить знаки значений заданной функции в трёх точках, одна из которых расположена внутри интервала между найденными корнями, а две другие — вне этого интервала:

- $x = 0 \Rightarrow F = (0 - 1) \cdot (0 - 3) = 3$ (знак «+»);
- $x = 2 \Rightarrow F = (2 - 1) \cdot (2 - 3) = -1$ (знак «−»);
- $x = 4 \Rightarrow F = (4 - 1) \cdot (4 - 3) = 3$ (знак «+»).

Таким образом, ветви параболы направлены вверх.

Вершина параболы (точка её минимума) определяется приравниванием нулю производной заданной функции:

$F = (x - 1) \cdot (x - 3) \Rightarrow F' = 2x - 4 \Rightarrow 2x - 4 = 0 \Rightarrow$
$\Rightarrow x = 2 \Rightarrow F = (2 - 1) \cdot (2 - 3) = -1.$



Имеется фрагмент текста программы:

```
if (F(t) < R) then
  begin
    M := t;
    R := F(t);
  end;
```

До каких пор будет производиться перезапись значения переменной M?

В предыдущей задаче аналогичный фрагмент листинга содержал условие `F(t) > R`, и подобная перезапись значений M происходила, пока последующее значение функции оказывалось больше, чем запомненное в переменной R предыдущее, т.е. происходил подъём по

графику до точки максимума. В нашем же случае условие обратное: `F(t) < R`. Оно показывает, что изменение значений *M* будет происходить, пока последующее значение функции будет меньше предыдущего. Это соответствует спуску до минимальной точки параболы.

Тогда по аналогии с предыдущей задачей можно предположить, что последним перезаписанным значением переменной *M* будет значение *t*, соответствующее точке минимума параболы, т.е. 2.

Ответ: в результате работы данной программы будет выведено число 2 (вариант ответа №2).

В подобных задачах, видя их аналогию, можно упростить решение:

- проанализировать вид графика заданной функции;
- определить, выполняется ли подъём по графику до максимальной точки или спуск до минимальной;
- найти эту максимальную или минимальную точку и записать в качестве ответа соответствующее ей значение аргумента (*x* или *t*).

Однако подобный способ решения — достаточно рискованный, поскольку опирается на предположение, что в условии задачи изменён только вид функции и, возможно, диапазон изменения аргумента. Если составители заданий ЕГЭ поменяют сам алгоритм, то такое «упрощённое» решение будет неверным. Поэтому на реальном ЕГЭ лучше дополнительно провести решение с полной трассировкой программы, показанное в задаче 1.

При решении таких задач, анализируя вид графика заданной функции, следует не забывать проверять, находится ли точка её минимума (максимума) в заданном интервале изменения абсцисс! В противном случае остановка перезаписи значения *M* произойдёт на границе интервала «на пути» к минимуму/максимуму, поэтому ответом будет значение аргумента на соответствующей границе исходного интервала.

Задача 3. Что будет напечатано в результате выполнения этой программы?

```
Program Task;
Uses crt;
const L = 4;
type
     atype = array [1..L] of integer;
Var R : atype;
     N, p : integer;

Procedure Pr1(L : integer; var R : atype );
 var i,n,t : integer;
begin
 for i := 1 to L do
 begin
  t := (R[i] div 2) * 4;
  R[i] := t mod 5;
 end;
end;

Function F1 (L : integer; R: atype) : integer;
 Var N, i, T : integer;
begin
 N := 1;
 T := 1;
 for i:=1 to L do
 begin
  N := N * R[i] + T;
  T := T + 2;
 end;
 F1 := N;
end;

BEGIN
 R[1] := 6; R[2] := 10; R[3] := 7; R[4] := 3;
 Pr1(L,R);
 N := F1(L,R);
 write(N);
 writeln;
END.
```

Решение

Зная принципы программирования процедур и отличия процедур от функций, анализируется предлагаемый в задаче программный код:

`Program Task;`	Начало основной программы
`Uses crt;`	
`const L = 4;`	*L* — длина массива
`type`	Определение составного пользовательского типа (целочисленного массива из 4 элементов)
`atype = array [1..L] of integer;`	
`Var R : atype;`	Определение экземпляра такого массива
`N, p : integer;`	
`Procedure Pr1(L : integer; var R : atype );`	Описание процедуры. Ей передаётся значение переменной *L* (как значение) и массив *R* (как ссылка)
`var i,n,t : integer;`	
`begin`	
`for i := 1 to L do`	Процедура непосредственно меняет значения элементов массива *R*, «принадлежащего» основной программе
`begin`	
`t := (R[i] div 2) * 4;`	
`R[i] := t mod 5;`	
`end;`	
`end;`	
`Function F1 (L : integer; R: atype) : integer;`	Определение подпрограммы-функции, которой также передаётся переменная *L* и массив *R* (оба параметра — как значения)
`Var N, i, T : integer;`	
`begin`	
`N := 1;`	
`T := 1;`	
`for i := 1 to L do`	
`begin`	
`N := N * R[i] + T;`	
`T := T + 2;`	
`end;`	Вычисленное значение *N* будет возвращено в основную программу как значение функции
`F1 := N;`	
`end;`	
`BEGIN`	Основная программа
`R[1] := 6; R[2] := 10;`	
`R[3] := 7; R[4] := 3;`	
`Pr1(L,R);`	Вызов процедуры (она изменит массив)
`N := F1(L,R);`	Вызов функции
`write(N);`	Печатается значение, возвращённое функцией
`writeln;`	
`END.`	

«Трассируются» переменные в этой программе.

1. *Основная программа:* группа присваиваний `R[1] := 6; R[2] := 10; R[3] := 7; R[4] := 3;` — формируется массив `R` = **[6, 10, 7, 3]**.

2. *Основная программа:* команда `Pr1(L, R);` — вызов процедуры, ранее описанной как

```
Procedure Pr1(L : integer; var R : atype );.
```

Ей в качестве параметров передаётся значение переменной `L` и ссылка на массив `R`, значит, элементы этого массива данная процедура будет менять напрямую.

3. *Процедура* `Pr1`: обратившись к её программному коду, выполняется трассировка в виде таблицы:

Передано: $L = 4$, $R[] = [6, 10, 7, 3]$

i	t	$R[]$
1	(R[1] div 2)*4 = (6 div 2)*4 = = 3 * 4 = **12**	t mod 5 = 12 mod 5 = **2**
2	(R[2] div 2)*4 = (10 div 2)*4 = = 5 * 4 = **20**	t mod 5 = 20 mod 5 = **0**
3	(R[3] div 2)*4 = (7 div 2)*4 = = 3 * 4 = **12**	t mod 5 = 12 mod 5 = **2**
4	(R[4] div 2)*4 = (3 div 2)*4 = = 1 * 4 = **4**	t mod 5 = 4 mod 5 = **4**

Таким образом, после завершения работы процедуры `Pr1` массив `R` (в основной программе!) будет иметь значения элементов: **[2, 0, 2, 4]**.

4. *Основная программа:* команда `N:=F1(L,R);` — вызов функции, ранее описанной как

```
Function F1 (L : integer; R: atype) :
                                  integer;.
```

Ей в качестве параметров передаются значение переменной `L` и массив `R` (как значения), а вернуть она должна целое число.

5. *Функция* `F1`: обратившись к её программному коду, выполняется трассировка в виде таблицы:

Передано: $L = 4$, $R[] = [2, 0, 2, 4]$

i	*N*	*T*
	1	1
1	$N*R[1] + T = 1*2 + 1 = 3$	$T + 2 = 1 + 2 = 3$
2	$N*R[2] + T = 3*0 + 3 = 3$	$T + 2 = 3 + 2 = 5$
3	$N*R[3] + T = 3*2 + 5 = 11$	$T + 2 = 5 + 2 = 7$
4	$N*R[4] + T = 11*4 + 7 = (51)$	$T + 2 = 7 + 2 = 9$

Функция возвращает значение `N` (которое присваивается имени этой функции — `F1 := N;`), равное 51. Именно это значение будет выведено на экран последующей командой `write(N);` и является ответом в данной задаче.

Ответ: **51.**

Задача 4*. Что будет напечатано в результате выполнения этой программы?

```
Program Task;
Uses crt;
const L = 5;
type
     atype = array [1..L] of integer;
Var R : atype;
     N, p : integer;

Procedure Multiply3_2(L, p : integer; var R : atype);
    var i,n,t : integer;
begin
p := 0;
for i := 1 to L do
 begin
  t := 2 * R[i] + p;
  R[i] := (t)mod(3);
  p := (t)div(3);
 end;
end;
```

```
Function Calc3 (L : integer; R: atype) : integer;
var
    N, i, T : integer;
begin
   N := 0;
   T := 1;
   for i := 1 to L do
    begin
     N := N + T * R[i];
     T := T * 3;
    end;
   Calc3 := N;
end;

BEGIN
   R[1] := 2; R[2] := 2; R[3] := 0;
   R[4] := 1; R[5] := 0;
   Multiply3_2(L, p, R);
   if (p > 0) then
    begin
     write('Переполнение');
     halt;
    end;
   N := Calc3(L,R);
   write(N);
   writeln;
END.
```

Решение

По аналогии с предыдущей задачей, можно выделить три «ключевых» блока: это *основная программа*, в которой задаётся исходный массив и откуда вызываются процедура и функция; *процедура*, которая меняет исходный массив, и *функция*, которая по этому массиву вычисляет некоторое числовое значение.

1) *задание массива:*

```
R[1]:=2; R[2]:=2; R[3]:=0; R[4]:=1; R[5]:=0;
```

Следовательно, массив `R` = **[2, 2, 0, 1, 0]**.

2) *обработка массива в процедуре:*

Передано: $L = 5$, $R[] = [2, 2, 0, 1, 0]$

i	*t*	*R*[]	*p*
–	–	–	0
1	2*R[i]+p = 2*R[1] +p = = 2* 2 + 0 = **4**	(t)mod(3)= = 4 mod 3 = **1**	(t)div(3)= 4 div 3 = **1**
2	2*R[i]+p = 2*R[2] +p = = 2* 2 + **1** = **5**	(t)mod(3)= = 5 mod 3 = **2**	(t)div(3)= 5 div 3 = **1**
3	2*R[i]+p = 2*R[3] +p = = 2* 0 + **1** = **1**	(t)mod(3)= = 1 mod 3 = **1**	(t)div(3)= 1 div 3 = **0**
4	2*R[i]+p = 2*R[4] +p = = 2* **1** + 0 = **2**	(t)mod(3)= = 2 mod 3 = **2**	(t)div(3)= 2 div 3 = **0**
5	2*R[i]+p = 2*R[5] +p = = 2* 0 + 0 = **0**	(t)mod(3)= = 0 mod 3 = **0**	(t)div(3)= 0 div 3 = **0**

Таким образом, после завершения работы процедуры `Pr1` массив `R` (в основной программе) будет иметь значения элементов: [1, 2, 1, 2, 0].

Кроме того, возвращается значение *p*, равное нулю. Условие в основной программе: `p > 0` ложно, поэтому сообщение о «переполнении» выдано не будет, а будет вызвана функция.

3) *обработка массива в функции:*

Передано: $L = 5$, $R[] = [1, 2, 1, 2, 0]$

i	*N*	*T*
–	0	1
1	N+T*R[i] = 0 + 1*R[1] = = 0 + 1*1 = **1**	T*3 = 1 * 3 = **3**
2	N+T*R[i] = 1 + 3*R[2] = = 1 + 3*2 = **7**	T*3 = 3 * 3 = **9**
3	N+T*R[i] = 7 + 9*R[3] = = 7 + 9*1 = **16**	T*3 = 9 * 3 = **27**
4	N+T*R[i] = 16 + 27*R[4] = = 16 + 27*2 = **70**	T*3 = 27 * 3 = **81**
5	N+T*R[i] = 70 + 81*R[5] = = 70 + 81*0 = **70**	T*3 = 81 * 3 = **243**

Функция возвращает значение N (которое присваивается имени этой функции — `Calc3 := N;`), равное 70. Именно это значение будет выведено на экран последующей командой `write(N);` и является ответом в данной задаче.

Ответ: **70.**

C1 Задачи на пересечение областей

Конспект

Связь между булевой логикой, теорией множеств и алгеброй функций

1. **Области**, выделяемые графиками функций на координатной плоскости, могут рассматриваться как **множества точек** этой плоскости (как замкнутые, так и открытые, если они не ограничены графиками со всех сторон).

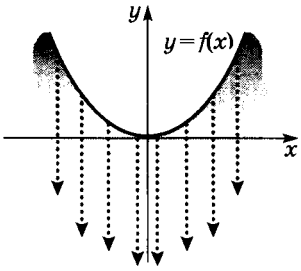
2. Пересекающиеся множества точек на плоскости могут рассматриваться как аналог кругов Эйлера-Венна: для таких областей аналогичным образом выполняются логические операции **НЕ** (NOT), **И** (AND) и **ИЛИ** (OR), определяющие, соответственно:

- операция **И** (AND) — пересечение областей (т.е. множество точек плоскости, принадлежащих обеим соединяемым этой операцией областям);
- операция **ИЛИ** (OR) — объединение областей (т.е. множество точек плоскости, принадлежащих любой из соединяемых этой операцией областей, в том числе их пересечению);
- операция **НЕ** (NOT) — множество точек плоскости, не принадлежащих области.

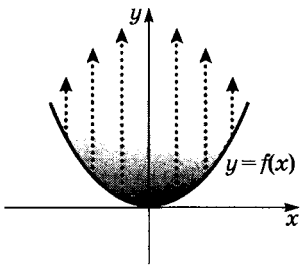
3. Границы областей определяются соответствующими графиками. При этом:

- один график функции обычно определяет соответствующую открытую область (полуплоскость, в том числе криволинейную); при этом условие, определяющее эту область, представляет собой неравенство вида $y \leqslant F(x)$, $y \geqslant F(x)$, $x \leqslant F^{-1}(y)$ или $x \geqslant F^{-1}(y)$, где F^{-1} — обратная функция по отношению к F; указан-

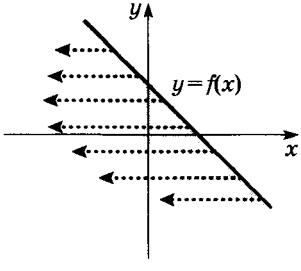
ное неравенство определяет условие принадлежности заданной точки данной области;



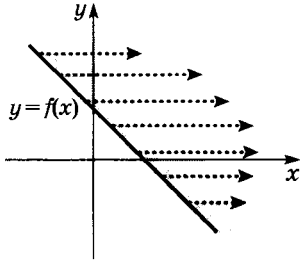
Всё, что ***ниже*** графика, определяется условием $y \leqslant F(x)$



Всё, что ***выше*** графика, определяется условием $y \geqslant F(x)$



Всё, что ***левее*** графика, определяется условием $x \leqslant F^{-1}(y)$



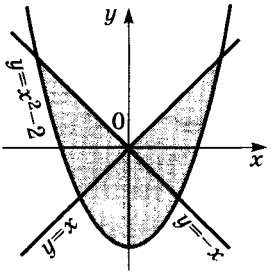
Всё, что ***правее*** графика, определяется условием $x \geqslant F^{-1}(y)$

- один график функции может определять замкнутую область, если функция имеет вид $F(x,y) = const$ (пример — $x^2 + y^2 = 1$ — окружность единичного радиуса), в этом случае знак $\leqslant$ или $\geqslant$, записанный вместо знака равенства в уравнении функции, определяет соответственно внутреннюю или наружную область относительно замкнутой кривой; указанное неравенство определяет условие принадлежности заданной точки данной области;
- замкнутая область, образуемая пересекающимися графиками функций, определяется как логическое

пересечение (**И**, AND) или объединение (**ИЛИ**, OR) отдельных элементарных областей (замкнутых или открытых); условие принадлежности точки такой области определяется как сложное логическое выражение, образованное элементарными условиями сравнения (условиями принадлежности точки каждой из элементарных областей) и указанными логическими операциями: **И** (AND) — для пересечения элементарных областей, **ИЛИ** (OR) — для их объединения.

Разбор типовых задач

Задача 1*. Требовалось написать программу, при выполнении которой с клавиатуры считываются координаты точки на плоскости (x, y — действительные числа) и определяется принадлежность этой точки заданной заштрихованной области (включая границы).



Программист торопился и написал программу неправильно.

Программа на Паскале	Программа на Бейсике

var x, y: real;
begin
readln(x, y);
if y <= x then
if y <= -x then
if y >= x * x - 2 then
write('принадлежит')
else
write('не принадлежит')
end.
	INPUT x, y
	IF y <= x THEN
	IF y <= -x THEN
	IF y >= x * x - 2 THEN
	PRINT "принадлежит"
	ELSE
	PRINT "не принадлежит"
	ENDIF
	ENDIF
	ENDIF
	END

Программа на СИ

```
void main(void)
{ float x,y;
scanf("%f%f",&x,&y);
if (y <= x)
if (y <= -x)
if (y >= x * x - 2)
printf("принадлежит");
else
printf("не принадлежит");
}
```

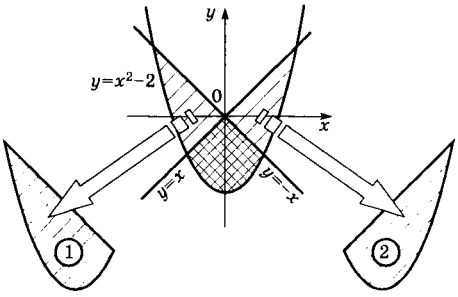
Последовательно выполните следующее:

1) приведите пример таких чисел x, y, при которых программа неправильно решает поставленную задачу;

2) укажите, как нужно доработать программу, чтобы не было случаев её неправильной работы. (Это можно сделать несколькими способами, поэтому можно указать любой правильный способ доработки исходной программы.)

Решение

Первое, что надо научиться делать, — это выделять на рисунке подобласти (в том числе прилегающие друг к другу или перекрывающиеся), границы которых соответствуют линиям графиков, — причём желательно, чтобы было минимальным как количество таких подобластей, так и количество графиков, участвующих в построении каждой подобласти. В данном случае можно выделить две таких подобласти, одна из которых ограничена параболой и прямой $y = -x$, а вторая — параболой и прямой $y = x$:



Второй важный момент касается построения условий, описывающих каждую выделенную «подобласть».

Вспомнив из курса геометрии, как выполняется пересечение полуплоскостей (пересечение двух полуплоскостей, в том числе с криволинейными границами, определяемыми указанными графиками функций):

- всё, что расположено **ниже** линии графика, соответствует условию «$y \leqslant f(x)$» (где $f(x)$ — функция, по которой построен график);

- всё, что расположено **выше** линии графика, соответствует условию «$y \geqslant f(x)$»;
- всё, что расположено **левее** линии графика, соответствует условию «$x \leqslant f^{-1}(y)$» (где $f^{-1}(y)$ — функция, обратная заданной для графика; для её получения достаточно в уравнении графика выразить x через y);
- всё, что расположено **правее** линии графика, соответствует условию «$x \geqslant f^{-1}(y)$»

Нестрогость операций сравнения следует из того, что согласно условию задачи точки на линиях границ области (графиков) включаются в эту область.

Область (в данном случае — подобласть) представляет собой **пересечение** указанных полуплоскостей. Очевидно, что каждая точка, принадлежащая этой области, должна одновременно принадлежать **И** первой полуплоскости, **И** второй, — следовательно, условие, определяющее эту область, можно получить из ранее выведенных «элементарных» условий, используя логическую операцию **И**. Здесь можно провести аналогию с кругами Эйлера.

Возвращаясь к задаче:

1. «Подобласть» ① ограничена графиком параболы ($y = x^2 - 2$) и прямой $y = -x$. Рассматривается всё, что расположено **выше** параболы (первое «элементарное» условие: $y \geqslant x^2 - 2$), и всё, что расположено **ниже** прямой (второе «элементарное» условие: $y \leqslant -x$). Тогда данная подобласть определяется логическим условием:

$$y \geqslant x^2 - 2 \text{ AND } y \leqslant -x$$

2. «Подобласть» ② ограничена графиком параболы ($y = x^2 - 2$) и прямой $y = x$. Рассматривается всё, что расположено **выше** параболы (первое «элементарное» условие: $y \geqslant x^2 - 2$), и всё, что расположено **ниже** прямой (второе «элементарное» условие: $y \leqslant x$). Тогда данная подобласть определяется логическим условием:

$$y \geqslant x^2 - 2 \text{ AND } y \leqslant x$$

Третье замечание касается объединения выделенных подобластей в одну требуемую в условии задачи область. Причём, эти подобласти могут прилегать друг к другу или перекрываться (как в данном случае).

Для записи условия, обозначающего принадлежность точки **любой** из выделенных подобластей, нужно соединить записи условий, определяющих каждую подобласть, при помощи операции **ИЛИ**:

$$(y \geqslant x^2 - 2 \text{ AND } y \leqslant -x) \text{ OR } (y \geqslant x^2 - 2 \text{ AND } y \leqslant x)$$

Это и есть решение задачи, — то самое условие, которое нужно проверять в программе и записать в **единственном** операторе `if`. Однако, в задаче также требуется найти ошибку в приведённом листинге, и указать пример точек, для которых приведённая в условии задачи программа будет работать неправильно. Поэтому далее следует перейти к анализу указанного в условии листинга (на примере программы на Паскале).

```
var x, y: real;
begin
readln(x,y);
if y <= x then
if y <= -x then
if y >= x * x - 2 then
write('принадлежит')
else
write('не принадлежит')
end.
```

Если цепочка из нескольких операторов `if … then` завершается записью `else`, то эта ветвь `else` *всегда* относится к последнему из стоящих перед ней операторов `if`.

Второе важное правило, которое с очевидностью следует из анализа работы программы с такой цепочкой последовательно записанных вложенных друг в друга операторов `if`, — что такая запись эквивалентна соединению условий, записанных в этих операторах `if`, через логическую операцию **И**.

Учитывая всё это, можно записать условие, запрограммированное в приведённом выше листинге:

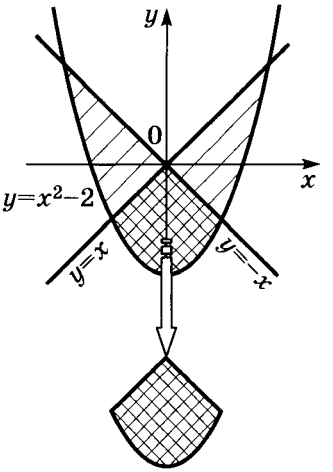
$$(y \leqslant x)\ \text{AND}\ (y \leqslant -x)\ \text{AND}\ (y \geqslant x^2 - 2)$$

Сравнивая два полученных условия (сформулированное из анализа чертежа и «извлечённое» из листинга программы), сразу бросается в глаза отсутствие во втором условии операции OR. Именно в этом и заключается ошибка листинга: зная, что соединение «элементарных» условий через **И** более «строго», чем через **ИЛИ**, нетрудно догадаться, что тем самым в программе «потеряны» какие-то части заданной области. Чтобы понять, какие именно, можно расшифровать второе (ошибочное) условие в виде чертежа, используя рассуждения, обратные тем, которые применялись при составлении условия по чертежу:

- условие «$y \leqslant f(x)$» (где $f(x)$ — функция, по которой построен график) соответствует всему, что расположено **ниже** линии графика;
- условие «$y \geqslant f(x)$» соответствует всему, что расположено **выше** линии графика;
- условие «$x \leqslant f^{-1}(y)$» (где $f^{-1}(y)$ — функция, обратная заданной для графика; для её получения достаточно в уравнении графика выразить x через y) соответствует всему, что расположено **левее** линии графика;
- условие «$x \geqslant f^{-1}(y)$» соответствует всему, что расположено **правее** линии графика;
- соединение этих «элементарных» условий через AND (т.е. через операцию И) соответствует пересечению соответствующих полуплоскостей, а через OR (операцию ИЛИ) — объединению областей (в том числе с их взаимным наложением).

Зная всё это, можно определить, что в программе было заложено условие, обозначающее принадлежность заданной точки области, которая расположена одновременно **выше** графика параболы и **ниже** обеих прямых.

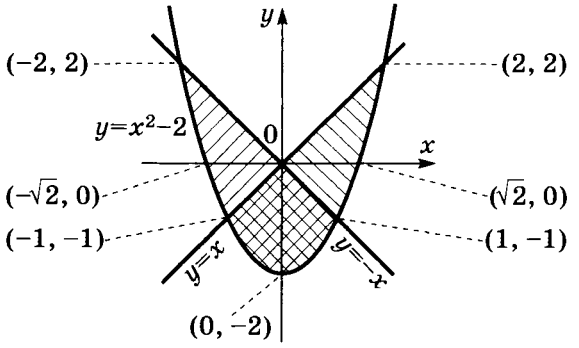
То есть речь идёт о подобласти, которая на чертеже выделена перекрестной штриховкой:



Отсюда, в качестве примера точек, для которых приведённая в условии задачи программа будет давать неправильный ответ, можно брать любые «удобные» точки из боковых частей области, которые на чертеже заштрихованы наклонными линиями. Например, это могут быть точки, расположенные на оси X: (1, 0) или (−1, 0). Или, как указано в ответе к данной задаче в материалах демонстрационного варианта ЕГЭ, точка (2, 2). Если решить уравнение $x^2 - 2 = 0$, чтобы определить координаты точек пересечения параболы с осью X, а также решить системы уравнений:

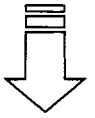
$$\begin{cases} y = x^2 - 2; \\ y = x, \end{cases} \text{ и } \begin{cases} y = x^2 - 2; \\ y = -x, \end{cases}$$

чтобы определить координаты точек пересечения заданных прямых с графиком параболы, то можно соответствующим образом «разметить» чертёж и убедиться, что точка (2, 2) находится как раз на пересечении графиков и потому попадает в правую боковую часть областей, которые ошибочная программа «упускает из вида».



Тем самым получен ответ на первый вопрос задачи. Чтобы ответить на второй вопрос (о необходимой доработке программы), достаточно переписать в виде оператора `if` выведенное ранее условие принадлежности точки заданной области:

```
(y >= x² - 2  AND  y <= -x)  OR
(y >= x² - 2  AND  y <= x)
```



```
var x, y: real;
begin
readln(x,y);
if ((y >= x * x - 2) and (y <= -x)) or
  ((y >= x * x - 2) and (y <= x)) then
    write('принадлежит')
else
write('не принадлежит')
end.
```

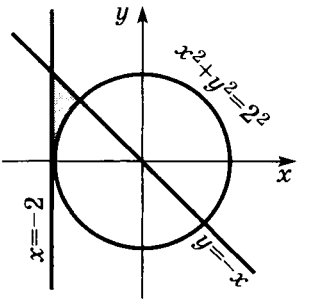
Доработка, приведённая в качестве примера правильного ответа в демонстрационном варианте ЕГЭ:

Возможная доработка (Паскаль, разбиение области на две части прямой x = 0):

```
if (y>=x*x-2) and (y<=x) and (x>=0) or (x<=0)
and (y<=-x) and (y>=x*x-2) then
     write('принадлежит')
else
write('не принадлежит')
```

менее рациональна, чем сформулированная в данном решении, поскольку разработчики заданий ЕГЭ в своей записи использовали ещё одну границу при разбиении подобластей — ось Y, которая записывается уравнением $x = 0$.

Задача 2*. Требовалось написать программу, при выполнении которой с клавиатуры считываются координаты точки на плоскости (x, y — действительные числа) и определяется принадлежность этой точки заданной заштрихованной области (включая границы).



Программист торопился и написал программу неправильно.

Программа на Паскале

```
var x,y: real;
begin
readln(x,y);
if x * x + y * y >= 4 then
if x >= -2 then
if y <= -x then
write('принадлежит')
else
write('не принадлежит')
end.
```

Программа на Бейсике

```
INPUT x, y
IF x * x + y * y >= 4 THEN
IF x >= -2 THEN
IF y <= -x THEN
PRINT "принадлежит"
ELSE
PRINT "не принадлежит"
ENDIF
ENDIF
ENDIF
END
```

Программа на СИ

```
void main(void)
{ float x, y;
scanf("% f % f",&x,&y);
if (x * x + y * y >= 4)
if (x >= -2)
if (y <= -x)
printf("принадлежит");
else
printf("не принадлежит");
}
```

Последовательно выполните следующее:

1) Приведите пример таких чисел x, y, при которых программа неправильно решает поставленную задачу.

2) Укажите, как нужно доработать программу, чтобы не было случаев её неправильной работы. (Это можно сделать несколькими способами, поэтому можно указать любой способ доработки исходной программы.)

Решение

1. Область образована пересечением полуплоскостей:

- ниже прямой $y = -x$;
- правее прямой $x = -2$;
- вне (т.е. «выше») окружности $x^2 + y^2 = 2^2$;
- выше оси X (т.е. прямой $y = 0$).

2. Запись условия, определяющего принадлежность точки этой области:

$$y \leqslant -x \text{ AND } x \geqslant -2 \text{ AND } x * x + y * y \geqslant 4 \text{ AND } y \geqslant 0$$

3. Запись условия, запрограммированного в приведённой в условии задачи программе:

$$x * x + y * y \geqslant 4 \text{ AND } x \geqslant -2 \text{ AND } y \leqslant -x$$

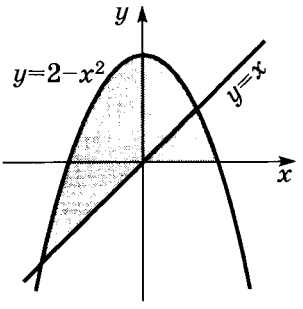
Очевидно, что ошибка в программе заключается в неучтённом ограничении «выше оси X».

4. Пример точки, для которой программа даёт ошибочный ответ (ошибочно относит эту точку к заданной области): (0, −3) — точка, лежащая вне окружности, правее прямой $x = -2$, ниже прямой $y = -x$, но не попадающей в указанную область. Пригодна в качестве ответа и точка (−1, −3).

5. Пример исправления программы:

```
var x, y: real;
begin
readln(x,y);
if (y <= -x) and (x >= -2) and
   (x * x + y * y >= 4) and (y >= 0) then
write('принадлежит')
else
write('не принадлежит')
end.
```

Задача 3*. Требовалось написать программу, при выполнении которой с клавиатуры считываются координаты точки на плоскости (x, y — действительные числа) и определяется принадлежность этой точки заданной закрашенной области (включая границы). Программист торопился и написал программу неправильно.



Паскаль

```
var x, y: real;
begin
 readln(x,y);
 if y >= x then
   if y >= 0 then
     if y <= 2 - x * x
then

write('принадлежит')
  else
    write('не принад-
          лежит')
end.
```

Бейсик

```
INPUT x, y
IF y >= x THEN
 IF y >= 0 THEN
  IF y <= 2 - x * x THEN
   PRINT "принадлежит"
  ELSE
   PRINT "не принадлежит"
  ENDIF
 ENDIF
ENDIF
END
```

Си

```
void main(void){
 float x, y;
 scanf("% f % f",&x,&y);
 if (y >= x)
   if (y >= 0)
     if (y <= 2 - x * x)

printf("принадлежит");
 else
  printf("не принад-
         лежит");
```

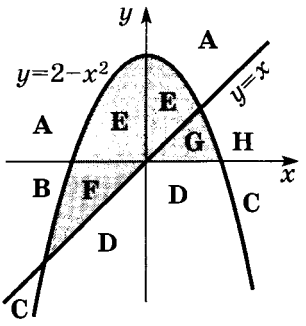
Алгоритмический язык

```
алг
нач
 вещ x, y
 ввод x, y
 если y >= x то
   если y >= 0 то
   если y <= 2 - x * x то
      вывод 'принадлежит'
      иначе
      вывод 'не принад-
             лежит'
      все
     все
  все
кон
```

Последовательно выполните следующее.

1. Перерисуйте и заполните таблицу, которая показывает, как работает программа при аргументах, принадлежащих различным областям (A, B, C, D, E, F, G и H).

Точки, лежащие на границах областей, отдельно не рассматривать.



Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A					
B					
C					
D					
E					
F					
G					
H					

В столбцах условий укажите «да», если условие выполнится, «нет», если условие не выполнится, «—» (прочерк), если условие не будет проверяться, «не изв.», если программа ведёт себя по-разному для разных значений, принадлежащих данной области. В столбце «Программа выведет» укажите, что программа выведет на экран. Если программа ничего не выводит, то напишите «—» (прочерк). Если для разных значений, принадлежащих области, будут выведены разные тексты, напишите «не изв». В последнем столбце укажите «да» или «нет».

2. Укажите, как нужно доработать программу, чтобы не было случаев её неправильной работы. (Это мож-

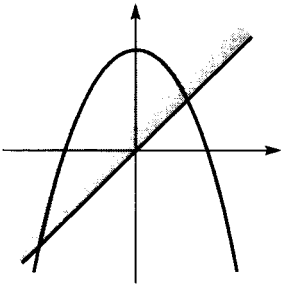
но сделать несколькими способами, достаточно указать любой способ доработки исходной программы.)

Решение

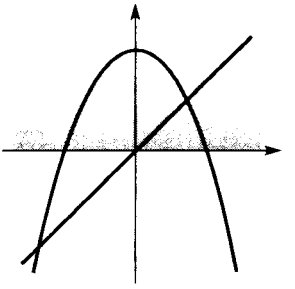
Эта новая модификация задач С1, впервые появившаяся на ЕГЭ в 2011/12 учебном году, решается практически полностью аналогично предыдущим. Заполнение же таблицы не только не усложняет решение, но даже помогает найти правильное условие для определения принадлежности точки указанной области. Необходим только навык внимательного заполнения таблицы.

1. Область можно определить как объединение двух подобластей:

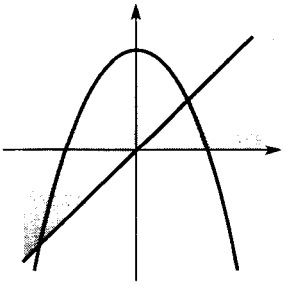
- выше графика $y = x$;
- выше оси $y = 0$,
- с последующим пересечением с полуплоскостью ниже графика $y = 2 - x^2$.



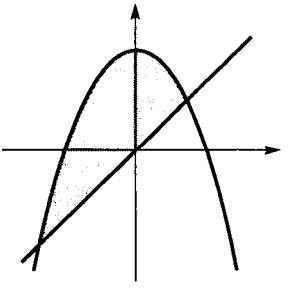
Подобласть ABEF



Подобласть AEGH



Область ABEFGH



Область EFG

2. Запись условия, определяющего принадлежность точки этой области.

1) Условие принадлежности точки подобласти ABEF:

```
(y >= x)
```

2) Условие принадлежности точки подобласти AEGH:

```
(y >= 0)
```

3) Условие принадлежности точки объединённой области ABEFGH:

```
(y >= x) OR (y >= 0)
```

4) Условие принадлежности точки области EFG:

```
((y >= x) OR (y >= 0)) AND (y <= 2 - x^2)
```

3. Заполнение таблицы надо производить построчно, глядя на рисунок с размеченными областями. При этом каждая строка заполняется слева направо, и если до какого-то условия выполнение программы просто не доходит, то в соответствующей ячейке ставится прочерк. В предпоследнем столбце записывается, что программа выведет на экран, — судя по содержимому срабатывающего оператора `write`. В последнем столбце надо записать свое заключение о том, соответствует ли выданное программой сообщение реальному положению дел.

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A					
B					
C					
D					
E					
F					
G					
H					

1) *Область A.*

Условие `y >= x` выполняется (область A расположена выше графика). Тогда в соответствующей ячейке пишется «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A	Да				

Условие `y >= 0` выполняется (область A расположена выше оси *X*). Тогда в следующей ячейке пишется «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A	Да	Да			

Условие `y <= 2 - x`^{2} не выполняется (область A расположена выше графика параболы). Тогда в следующей ячейке пишется «нет»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A	Да	Да	Нет		

Соответственно, срабатывает ветвь `else` третьего оператора `if`, и программа выводит «не принадлежит». Это правильное определение, так как данная подобласть не входит в заштрихованную область. Поэтому в последней ячейке записывается «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A	Да	Да	Нет	Не принадлежит	Да

2) *Область B.*

Условие `y >= x` выполняется (область B расположена выше графика). Тогда в соответствующей ячейке пишется «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
B	Да				

Условие `y >= 0` не выполняется (область B расположена ниже оси X). Тогда в следующей ячейке пишется «нет»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
B	Да	Нет			

Поскольку третье условие размещено в ветви `then` второго оператора `if`, которая не будет срабатывать (ведь условие, записанное во втором операторе `if`, ложно), в третьей графе ставится прочерк:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
B	Да	Нет			

По правилам языка Паскаль, ветвь `else` принадлежит последнему оператору `if` в их цепочке. Поскольку он не выполняется, программа не выведет ничего (ставится прочерк в предпоследней ячейке):

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
B	Да	Нет	—	—	

Правильно ли обрабатывается эта область? Очевидно, нет, так как программа должна была бы для неё вывести текст «не принадлежит». Поэтому в последней ячейке записывается «нет»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
B	Да	Нет	—	—	Нет

3) *Область C.*

Аналогичным образом заполняется строка для области C, учитывая, что для неё не выполняется первое условие:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
C	Нет	—	—	—	Нет

4) *Область D.* Для неё также не выполняется первое условие:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
D	Нет	—	—	—	Нет

5) *Область E.* Для неё выполняются все три условия, поэтому срабатывает ветвь `then` последнего оператора `if` и выводится «Принадлежит». Это правильное заключение, поэтому в последней графе записывается «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
E	Да	Да	Да	Принадлежит	Да

6) *Область F*. Для неё не выполняется второе условие, на экран ничего не выводится. Это неправильное заключение, в последней графе записывается «нет»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
F	Да	Нет	—	—	Нет

7) *Область G*. Для неё выполняются все три условия, поэтому срабатывает ветвь `then` последнего оператора `if` и выводится «Принадлежит». Это правильное заключение, в последней графе записывается «да»:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
G	Да	Да	Да	Принадлежит	Да

8) *Область H*. Для неё не выполняется первое условие:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
H	Нет	—	—	—	Нет

В результате заполненная таблица имеет вид:

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
A	Да	Да	Нет	Не принадлежит	Да
B	Да	Нет	–	–	Нет
C	Нет	–	–	–	Нет
D	Нет	–	–	–	Нет

Окончание табл.

Область	Условие 1 ($y \geqslant x$)	Условие 2 ($y \geqslant 0$)	Условие 3 ($y \leqslant 2-x*x$)	Программа выведет	Область обрабатывается верно
E	Да	Да	Да	Принадлежит	Да
F	Да	Нет	–	–	Нет
G	Да	Да	Да	Принадлежит	Да
H	Нет	–	–	–	Нет

4. В отличие от предыдущих задач, здесь не требуется указывать конкретную ошибку в записи условий в исходной программе и пример точки, которую программа обрабатывает неправильно. Поэтому можно текст программы не анализировать вовсе.

5. Пример исправления программы, как и раньше, можно получить, записав в одном операторе `if` ранее выведенное условие принадлежности точки заштрихованной области:

```
var x, y: real;
begin
 readln(x,y);
 if ((y >= x) OR (y >= 0)) AND (y <= 2 - x * x)
     then write('принадлежит')
      else write('не принадлежит')
end.
```

C4 Задачи на анализ и обработку данных

Конспект

Решение задач типа C4 требует не только умения применять типовые алгоритмы обработки данных (поиск минимума/максимума, вычисление среднего значения, определение количества элементов массива, удов-

летворяющих заданному условию, сортировка и пр.), но и умений анализировать условие задачи, выделять «ключевые» данные (и определять, какие данные являются излишними и не требуют запоминания), применять нестандартные приёмы ввода данных, формировать индексные массивы и массивы-счётчики и т.д.

Данное занятие посвящено разбору основных приёмов программирования, используемых при решении подобных задач.

Пропуск текстовых данных из потока символов

Данные на входе программы могут представлять собой строку, содержащую несколько чисел или строковых констант, разделённых пробелом.

При чтении данных из такой входной строки оператором `read` (`readln`) чтение данных должно производиться слева направо с начала и до конца.

Если какие-то данные не нужны для работы программы, то для их пропуска можно использовать следующие приёмы:

а) для пропуска числовых данных — считать ненужные данные в отдельно объявленную «буферную» переменную соответствующего типа, которая в программе далее не используется (при чтении следующего ненужного данного используется та же переменная).

Пример:

```
...

// входная строка: <день> <месяц> <год>, где день
// и месяц не нужны

var Year : integer;   // год
    Musor : integer; // переменная для чтения
                     // ненужных данных

...

read(Musor);   // считали день
read(Musor);   // считали в ту же переменную
               // месяц
readln(Year); // считали год, переменная Year
               // идет на обработку
```

б) для пропуска текстовых данных — поскольку их длина заранее не известна, а считывание в переменную типа `string` загружает в неё всю входную строку, нужно в цикле считывать их входной строки отдельные символы, включая пробел, завершающий очередное данное. Для пропуска ещё одного такого данного повторно строится такой же цикл.

Пример:

```
...

// входная строка: <фамилия> <имя> <возраст>,
// где фамилия и имя не нужны

var Age : integer;   // возраст
    Musor : char;   // переменная для чтения
                    // ненужного символа

...

{Пропуск фамилии}
repeat
  read(Musor)
until Musor=' ';
// считывается очередной ненужный символ, пока
// не окажется, что был считан пробел

{Пропуск имени}
repeat
  read(Musor)
until Musor=' ';
// считывается очередной ненужный символ, пока
// не окажется, что был считан пробел

{Чтение возраста}
readln(Age);   // считали возраст, переменная
               // Age идет на обработку
```

Массивы-счётчики

При определении количества элементов массива, соответствующих заданному условию (например, количества чётных элементов или количества элементов, равных максимальному), использовалась переменная-счётчик, которая увеличивалась на 1 при обнаружении подходящего элемента.

При необходимости подсчёта количеств нескольких объектов используется *массив-счётчик* размера, соответствующего количеству объектов. При обнаружении в потоке входных данных объекта, соответствующего условию, нужно увеличить на 1 значение ячейки массива-счётчика, соответствующей данному объекту.

Пример.

Для каждого из 8 филиалов фирмы (пронумерованных от 1 до 8) вразброс вводятся записи из трёх чисел:
`<номер филиала> <номер месяца> <прибыль/убыток>`
где прибыль выражена положительной суммой в рублях, а убыток — отрицательной. Завершение входных данных — номер филиала, равный нулю.

Требуется для каждого филиала подсчитать количество месяцев с убытком.

Решение.

1. Выделяется массив-счётчик. Количество филиалов равно 8, поэтому массив-счётчик должен состоять из 8 элементов с индексами от 1 до 8:

```
var Kol : array[1..8] of integer;
```

2. Считывается очередная строка данных (три числа в переменные `Filial`, `Mes` и `Summ`).

3. Строится цикл с предусловием:

```
while (Filial <> 0) do begin
```

4. В цикле обрабатываются входные данные:

- если обнаружен убыток, т.е. `Summ` < 0, то значение ячейки массива-счётчика, соответствующей номеру филиала, увеличиваем на 1.

```
if (Summ < 0) then Kol[Filial] := Kol[Filial] + 1;
```

5. Считывается очередная порция данных (три числа в переменные `Filial`, `Mes` и `Summ`) — если номер филиала не равен нулю, то выполняется очередной проход цикла.


Такое построение цикла позволяет отследить ситуацию, когда входных данных нет (т.е. сразу вводится завершающая строка с нулевым номером филиала).

6. После завершения цикла массив-счётчик, содержащий количества убыточных месяцев для каждого филиала, может быть дополнительно обработан (напри-

мер, может быть выполнен поиск номера филиала — номера ячейки этого массива, имеющего максимальное количество убыточных месяцев).

 При формировании массива-счётчика нужно не забывать, что в качестве индексов его элементов в Паскале могут использоваться не только произвольные диапазоны целых чисел, но и диапазоны символов, например букв алфавита.

Индексные массивы

Возможна ситуация, когда в качестве индексов массива-счётчика невозможно (или не рационально) использовать значения, считываемые из входных данных в качестве идентификатора объектов, например:

- если вводятся словесные названия месяцев, для которых нужно выполнять подсчёт (строковые данные нельзя использовать как индексы элементов массива);
- если номера объектов идут не подряд. Например, известно, что вводятся номера школ, для которых обрабатываются какие-то данные: 215, 386, 512, 1167, 3160, тогда при создании массива-счётчика с полным диапазоном индексов от 215 до 3160 подавляющее большинство его элементов, кроме пяти с перечисленными индексами, будут не использованы.

В этих случаях применяется *индексный массив*:

1) массив-счётчик объявляется по количеству обрабатываемых объектов (например, в случае с месяцами — на 12 элементов, в случае со школами — на 5 элементов);

2) отдельно объявляется массив того же размера, который содержит обозначения соответствующих объектов (для месяцев — их словесные названия, для школ — их номера);

3) при считывании входных данных соответствующее обозначение объекта ищется в индексном массиве (путём его просмотра с начала до конца); номер ячейки индексного массива, содержащей искомое обозначение объекта, запоминается;

4) если данный объект соответствует требуемому условию, то в массиве-счётчике увеличивается на 1 элемент, индекс которого равен запомненному индексу элемента в индексном массиве, содержащего обозначение этого объекта.

 Связь индексного массива и массива-счётчика осуществляется через общий индекс элемента.

Если при выводе полученных данных требуется выводить обозначения объектов, то они берутся из индексного массива.

 Проверить, входит ли символ в заданное множество символов, позволяет оператор `in`. Запись вида `<символ> in [<множество>]` истинна, если символ входит в данное множество. При этом требуемое множество в квадратных скобках записывается как перечисление через запятую отдельных символов или интервалов символов. Примеры:

- множество знаков препинания: `['.', ',', '!', '?', ';', ':']`;
- множество латинских букв: `['A'..'Z', 'a'..'z']`.

 Получить число из его символьной записи (строки цифр) можно:

- при помощи стандартных функций:
  - `copy(<строка>, <начальная позиция>, <длина>)` (извлекает из строки подстроку заданной длины, начинающуюся с символа с заданной начальной позицией),
  - `LeftStr(<строка>, <длина>)` (извлекает из строки подстроку заданной длины слева),
  - `RightStr(<строка>, <длина>)` (извлекает из строки подстроку заданной длины справа),
  - `StrToInt(<строковая запись числа>)` (преобразует запись целого числа в виде строки цифр в само число);
- путём обработки кодов символов цифр: код символа позволяет получить стандартная функция `ord(<символ>)`, при этом разность кода символа заданной цифры и кода символа цифры «нуль» равна числовому значению заданной цифры (см. таблицу кодировки ASCII). Тогда можно получить число, отдельно обрабатывая каждую его цифру указанным способом и суммируя полученные числовые значения цифр с учётом их разрядов (умножением на 10 в степени, равной номеру разряда).

Проверка равенства вещественных значений

Как известно, вещественные значения в памяти компьютера представлены приближенно. Поэтому, например, вычисленное значение `2.0 + 2.0` в общем случае может оказаться не равно вычисленному значению `2.0 * 2.0` (из-за погрешностей вычислений). Из-за этого использовать при сравнении вещественных значений оператор равенства не совсем корректно. Вместо этого производится сравнение модуля разности сравниваемых величин с некоторым малым значением, определяемым точностью вычислений. В данном случае в качестве такого малого значения можно использовать число `0,0001`.

Разбор типовых задач

Задача 1*. На автозаправочных станциях (АЗС) продаётся бензин с маркировкой 92, 95 и 98. В городе N был проведен мониторинг цены бензина на различных АЗС.

Напишите эффективную по времени работы и по используемой памяти программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая будет определять для каждого вида бензина, сколько АЗС продают его дешевле всего. На вход программе в первой строке подаётся число данных о стоимости бензина. В каждой из последующих *N* строк находится информация в следующем формате:

```
<Компания> <Улица> <Марка> <Цена>
```

где `<Компания>` — строка, состоящая не более, чем из 20 символов без пробелов, `<Улица>` — строка, состоящая не более, чем из 20 символов без пробелов, `<Марка>` — одно из чисел — 92, 95 или 98, `<Цена>` — целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках. `<Компания>` и `<Улица>`, `<Улица>` и `<Марка>`, а также `<Марка>` и `<Цена>` разделены ровно одним пробелом. Пример входной строки:

```
Синойл Цветочная 95 2250
```

Программа должна выводить через пробел 3 числа — количество АЗС, продающих дешевле всего 92-й, 95-й и 98-й бензин соответственно. Если бензин какой-то марки нигде не продавался, то следует вывести 0. Пример выходных данных:

```
12 1 0
```

Решение

1. Необходимо вычислять минимальную цену на бензин каждого вида и количество таких значений. Это можно делать сразу при обработке введённых данных без организации массива-счётчика.

Вместо этого удобно определить два массива (взаимосвязанных через индексы элементов), в элементах которых будут храниться соответственно текущее минимальное значение цены бензина соответствующей марки (массив `min`) и количество элементов, равных этому минимуму (массив `ans`).

```
var min, ans: array[92..98] of integer;
```

При этом принципы работы с этими массивами аналогичны работе с массивом-счётчиком: считанный номер марки бензина используется как индекс соответствующего элемента в массиве. (Массивы объявлены на 7 элементов каждый, при этом в каждом используется только по три элемента.)

2. Входные данные: сначала отдельно считывается количество строк N, а затем в цикле с параметром считываются сами строки.

При этом строковые данные `<Компания>` и `<Улица>` — лишние, их нужно пропустить путём посимвольного чтения до завершающего пробела включительно.

Номер бензина и цена (целые числа) считываются в соответствующие переменные.

3. Обработка считанных данных: считанное значение номера марки бензина используется как индекс и определяет, какая пара переменных (предполагаемый минимум и счётчик значений, равных ему) разбирается в данном случае. Считанное значение цены обрабатывается в типовом алгоритме поиска минимума с подсчётом количества значений, равных минимуму, аналогично элементу массива:

1) цена, по условию, не превышает 3000, поэтому при начальной инициализации ячеек массива `min` в качестве предполагаемых минимумов задаётся константа, заведомо большая максимально возможного значения

цены (число 3001 или большее); счётчики количества значений, равных минимальному (массив ans), обнуляются.

Все эти инициализационные действия производятся в цикле (индексы массивов меняются от 92 до 98) ***до*** начала цикла чтения входных строк.

2) после чтения номера марки бензина k и цены b выполняется проверка: если b < min[k], то:

- значение b запоминается как новое значение предполагаемого минимума min[k];
- значение счётчика ans[k] устанавливается в 1 (т.е. одно такое значение уже найдено);

В зависимости от номера марки бензина, работаем с соответствующей парой ячеек «минимум — счётчик».

3) иначе если значение b равно текущему значению минимума для данной марки бензина min[k], то значение счётчика ans[k] увеличивается на 1.

Если бензина какой-то марки не было вообще, то значение соответствующего счётчика не изменится (останется равным нулю).

По завершении обработки входных данных в ячейках ans[92], ans[95] и ans[98] имеются искомые количества встреченных значений цен, равных минимальным (для каждой из трёх этих марок), либо нули.

4. Вывод данных: на экран выводятся полученные значения ans[92], ans[95] и ans[98].

Полный текст программы:

```
program C4;
var min, ans: array[92..98] of integer;
    c: char;
    i, k, N, b: integer;
begin
  for i := 92 to 98 do // инициализация
                       // начальных значений
   begin               // предполагаемых
                       // минимумов и счетчиков
    min[i] := 3001;
    ans[i] := 0;
   end;
```

```
  readln(N);          // считано количество строк
  for i := 1 to N do   // чтение строк входных
                       // данных
    begin
     repeat
       read(c);
     until c=' ';    // пропуск названия компании
     repeat
       read(c);
     until c = ' ';    // пропуск названия улицы
     readln(k,b);      // чтение номера марки
                       // бензина и его цены

     // поиск минимума с подсчетом количества
     // элементов, равных этому минимуму
     if b < min[k] then
      begin
       min[k] := b;
       ans[k] := 1
      end else
     if min[k] = b then ans[k] := ans[k] + 1;
  end;

  // вывод результатов
  writeln(ans[92],' ', ans[95],' ', ans[98])
end.
```

Задача 2*. На вход программе подаётся набор символов, заканчивающийся точкой (в программе на языке Бейсик символы можно вводить по одному в строке, пока не будет введена точка, или считывать данные из файла). Напишите эффективную, в том числе и по используемой памяти, программу (укажите используемую версию языка программирования, например, Borland Pascal 7.0), которая сначала будет определять, есть ли в этом наборе символы, соответствующие десятичным цифрам. Если такие символы есть, то можно ли переставить их так, чтобы полученное число было симметричным (читалось одинаково как слева направо, так и справа налево). Ведущих нулей в числе быть не должно, исключение — число 0, запись которого содержит ровно один нуль.

Если требуемое число составить невозможно, то программа должна вывести на экран слово «NO». А если возможно, то в первой строке следует вывести слово «YES», а во второй — искомое симметричное число. Если таких чисел несколько, то программа должна выводить максимальное из них. Например, пусть на вход подаются следующие символы:

```
Do not 911 to 09 do.
```

В данном случае программа должна вывести

```
YES
91019
```

Решение

1. Необходимо вычислять количества каждой из цифр от 0 до 9, поэтому объявляется массив-счётчик с индексами элементов от `'0'` до `'9'`.

```
var a: array['0'..'9'] of integer;
```


Внимание! Индексы элементов массива — это символы `'1' .. '9'`, а не числа `1 .. 9` (либо при использовании в качестве индексов именно чисел нужно считанные из входной строки символы цифр преобразовывать в их числовые значения).

Поскольку в качестве индексов массива-счётчика используются сами считываемые обозначения объектов, использовать индексный массив не требуется.

2. Входные данные: считываются посимвольно до завершающей точки (цикл с предусловием).

3. Обработка считанных данных: если считанный символ представляет собой цифру, то в массиве-счётчике увеличивается на 1 значение элемента с индексом, равным этому символу цифры.

4. Обработка полученного массива-счётчика.

Построение максимального по величине симметричного числа возможно в трёх случаях:

- если количества всех цифр четны;
- если есть только одна цифра, количество которой нечётно (цепочка этих цифр будет в полученном симметричном числе стоять в середине);

- если не имеется никаких цифр, кроме нулей, то симметричное число можно построить только если этот нуль — единственный.

Исходя из анализа этих условий решается задача[1]:

1) просматривая массив-счётчик, подсчитывается в нём (в переменной *k*) количество цифр, встречающихся нечётное число раз. При этом в переменной `c_odd` запоминается символ цифры, которая последней встретилась нечётное число раз, а в переменной *s* независимо от чётности количества каждой цифры подсчитывается суммарное количество всех цифр (оно понадобится позже для проверки другого условия).

```
k := 0;
for c := '0' to '9' do
 begin
  if a[c] mod 2 = 1 then
   begin
    k := k + 1;
    c_odd := c
   end;
  s := s + a[c];
 end;
```

2) проверяется условие: количество нулей равно 1, а количества других цифр равны нулю (что понятно из равенства общей суммы количеств всех цифр всё той же единице). Если это истинно, то выводится слово «YES» и число 0.

```
if (a['0'] = 1) and (s = 1) then
 begin
  writeln('YES');
  writeln('0');
 end else
```

3) иначе проверяется условие: если нули — не единственные цифры и при этом `k = 0` или `k = 1`, то выводится слово «YES» и конструируется число; иначе выводится слово «NO»:

---

[1] В демонстрационном варианте ЕГЭ за 2011 год приведено другое решение. Предлагаемый здесь вариант решения более «прозрачен» для понимания смысла алгоритма.

```
  if (s <> a['0']) and ((k = 0) or (k = 1))
then
   begin
    writeln('YES');
    <вывод симметричного числа>
   end else writeln('NO');
```

4) симметричное число конструируется так:

- перебираются цифры с 9 до 0 (цикл с параметром, изменяемым в обратном порядке — от '9' до '0'):

```
for c := '9' downto '0' do
```

- выводится цепочку таких цифр (цифра — текущее значение параметра цикла), длина которой равна половине всего количества таких цифр (если это количество нечётно, то дробная часть отбрасывается, т.е. используется целочисленное деление):

Способ 1: с использованием стандартной функции	Способ 2: без использования стандартной функции
`write(StringOfChar(c, a[c] div 2));`	`for i := 1 to a[c] div 2 do write(c);`

- по завершении цикла перебора цифр от 9 до 0, если `k = 1`, выводится одна цифра, количество которой было нечётно (она запомнена в переменной `c_odd`):

```
if k = 1 then write(c_odd);
```

- перебирается цифры с 0 до 9:

```
for c := '0' to '9' do
```

- выводится цепочка таких цифр (цифра — текущее значение параметра цикла), длина которой равна половине всего количества таких цифр (если это количество нечётно, то дробная часть отбрасывается, т.е. используется целочисленное деление):

Способ 1: с использованием стандартной функции	Способ 2: без использования стандартной функции
`write(StringOfChar(c, a[c] div 2));`	`for i := 1 to a[c] div 2 do write(c);`

Полный текст программы:

```
program C4;
var a: array['0'..'9'] of integer;
     c, c_odd: char;
     i, k, s: integer;
begin
   for c := '0' to '9' do a[c] := 0;
// обнуляется массив-счетчик

   read(c);    // чтение символов из строки до
               // точки
   while c <> '.' do
    begin
     if c in ['0' .. '9'] then a[c] := a[c] + 1;
     // если считанный символ — цифра,
     // то соответствующий элемент
     // массива-счетчика увеличивается на 1
     read(c);
   end;

// подсчет количества цифр, встречающихся
// нечетное число раз
  k := 0; s := 0;          // начальные обнуления
   for c := '0' to '9' do   // просмотр каждой
                            // цифры
    begin
     if a[c] mod 2 = 1 then
// если ее количество нечетно, то
      begin
       k := k + 1;      // увеличить счетчик
       c_odd := c       // и запомнить эту цифру
      end;
     s := s + a[c];
     // в любом случае подсчет суммарного
     // количества всех цифр
    end;

// проверка первого условия (единственный нуль)
   if (a['0'] = 1) and (s = 1) then
    begin
     writeln('YES');
     writeln('0');
    end else
```

```
  // иначе проверка второго условия
  // (не более одной цифры встречается нечетное
  // число раз, притом нули — не единственные
  // цифры)
  if (s <> a['0']) and ((k = 0) or (k = 1)) then
    begin
      writeln('YES');

      // если условие соблюдается,
      // то конструируется число
      for c := '9' downto '0' do
        write(StringOfChar(c, a[c] div 2));
        // первая половина
      if k = 1 then write(c_odd);
      // средний символ
      for c := '0' to '9' do
        write(StringOfChar(c, a[c] div 2));
        // вторая половина

        // иначе ответ — «NO»
    end else writeln('NO');

end.
```

Задача 3*. В командных олимпиадах по программированию для решения предлагается не больше 11 задач. Команда может решать предложенные задачи в любом порядке. Подготовленные решения команда посылает в единую проверяющую систему соревнований.

Вам предлагается написать эффективную, в том числе по используемой памяти, программу, которая будет статистически обрабатывать пришедшие запросы, чтобы определить наиболее популярные задачи. Следует учитывать, что количество запросов в списке может быть очень велико, так как многие соревнования проходят с использованием Интернет.

Перед текстом программы кратко опишите используемый вами алгоритм решения задачи.

На вход программе в первой строке подаётся количество пришедших запросов N. В каждой из последующих N строк записано название задачи в виде текстовой строки. Длина строки не превосходит 100 символов, на-

звание может содержать буквы, цифры, пробелы и знаки препинания.

Пример входных данных:

```
6
A+B
Крестики-Нолики
Прямоугольник
Простой делитель
A+B
Простой делитель
```

Программа должна вывести список из трёх наиболее популярных задач с указанием количества запросов по ним. Если в запросах упоминаются менее трёх задач, то выведите информацию об имеющихся задачах. Если несколько задач имеют ту же частоту встречаемости, что и третья по частоте встречаемости задача, их тоже нужно вывести.

Пример выходных данных для приведённого выше примера входных данных:

```
A+B 2
Простой делитель 2
Крестики-Нолики 1
Прямоугольник 1
```

Решение

1. Требуется подсчёт количества каждой из задач. Всего задач может быть 11 видов. Следовательно, необходим массив-счётчик на 11 ячеек. Поскольку названия задач не могут являться индексами массива-счётчика, требуется также индексный массив из 11 ячеек, который должен хранить названия этих задач.

```
Var Count: array[1..11] of integer;
// массив-счетчик
    Names: array[1..11] of string;
    // индексный массив
```

Однако есть дополнительная сложность: названия задач, которые нужно запоминать, и даже реальное количество их разновидностей неизвестны. Поэтому требуется *динамически формировать индексный массив.*

2. Входные данные: вначале отдельно считывается количество строк, а затем в цикле с параметром вводятся строки — названия задач.

3. Обработка входных данных:

1) в индексном массиве путём просмотра с его начала количества заполненных его элементов (это количество изначально равно нулю) ищется элемент, равный считанному названию задачи;

```
Num := 0;
// количество заполненных элементов
// индексного массива первоначально равно нулю
. . .
  j := 1; // просмотр индексного массива
          // с начала
  while (j <= Num) and (s <> Names[j])
  do j := j + 1;
  // переходим к следующему элементу
  // индексного массива, пока или не будут
  // просмотрены все его заполненные элементы,
  // или в нем не будет найдено считанное
  // название задачи
```

2) если такой элемент найден, то элемент массива-счётчика с соответствующим индексом (*j*) увеличивается на 1;

```
    if j <= Num then
// значит, просмотр завершен раньше, чем
// просмотрены все заполненные элементы, т.е.
// в ячейке с индексом j найдено требуемое
// название задачи
      Count[j] := Count[j] + 1
```

3) иначе (если такой элемент не найден) индексный массив наращивается на один элемент:

```
    else begin    // иначе
```

- в текущую (незанятую) ячейку индексного массива с индексом *j* записывается считанное новое название задачи;

```
      Names[j] := s;
```

- в соответствующую ячейку (с индексом *j*) массива-счётчика записывается единица (одна такая задача уже встречена);

```
      Count[j] := 1;
```

- количество заполненных элементов индексного массива увеличивается на 1.

```
    Num := Num + 1
  end
```

Результат: заполненные по всем встреченным задачам индексный массив с названиями задач и массив-счётчик с их количествами.

4. Обработка массива-счётчика: требуется определить три вида задач, для которых количества будут наибольшими. Для этого проще всего отсортировать по убыванию массив-счётчик (и синхронно с ним — индексный массив). Используется метод «пузырька» (попарное сравнение элементов и при необходимости их обмен местами); для упрощения решения задачи — с полным просмотром (без контроля досрочного прерывания сортировки, если массив уже отсортирован).

```
for i := Num downto 2 do
  for j := 2 to i do
    if Count[j-1] < Count[j] then
     begin
      t := Count[j]; Count[j] := Count[j-1];
      Count[j-1] := t;
      s := Names[j]; Names[j] := Names[j-1];
      Names[j-1] := s;
     end;
```

 Внимание! При перестановке элементов в массиве-счётчике нужно одновременно переставлять соответствующие им элементы индексного массива, чтобы сохранить соответствие этих массивов.

5. Вывод данных:

1) если найдено только три вида задач или меньше, то надо выводить информацию обо всех них (тогда в переменной *j* запоминается номер последней запомненной задачи), иначе (если видов задач больше) — только о трёх первых видах (тогда в переменной *j* запоминается номер 3);

2) массив-счётчик просматривается с начала и до тех пор, пока не исчерпано количество заполненных его элементов (*Num*) и пока количество задач текущего

вида не меньше количества задач того вида, что в отсортированных массивах (счётчике и индексном) расположен в месте под номером *j*. При этом, если различных задач меньше трёх, то будет выведена информация обо всех них, а если их больше трёх, то будет выведена информация о трёх первых по количеству видах задач, а также о следующих задачах (четвёртой и т.д.), если их количество совпадает с количеством задач третьего вида.

```
  if Num >= 3 then j := 3 else j := Num;
  i := 1;
 while (i <= Num) and (Count[i] >= Count[j]) do
   begin
    WriteLn(Names[i], ' ', Count[i]);
    i := i + 1;
   end
```

Полный текст программы:

```
Program C4;
Var Count: array[1..11] of integer; // массив-
                                    // счетчик
    Names: array[1..11] of string; // индексный
                                   // массив
    n, Num, i, j, t: integer;
    s: string;
Begin
  Num := 0;
 // изначально в индексном массиве не заполнено
  // ни одного элемента
 ReadLn(N);          // считано количество строк
 for i := 1 to N do  // чтение строк входных
                     // данных
   begin
   ReadLn(S);      // считано очередное название
                   // задачи

    // считанное название задачи ищется
    // в индексном массиве:
    j := 1;
    while (j <= Num) and (s <> Names[j]) do
    j := j + 1;
    if j <= Num then
    // если задача найдена в j-й ячейке,
```

```
        Count[j] := Count[j] + 1
//   то  увеличиваем  соответствующий  счетчик
     else begin    // иначе название задачи
                   // добавляется в конце
                   // индексного массива
                   // (в ячейку с индексом j)
        Names[j] := s;
        Count[j] := 1;
        Num := Num + 1
     end
    end;

  // сортировка массива-счётчика по убыванию и
  // синхронно с ним - индексного массива
  // (методом «пузырька»)
  for i := Num downto 2 do
    for j := 2 to i do
      if Count[j-1] < Count[j] then
       begin
        t := Count[j]; Count[j] := Count[j-1];
        Count[j-1] := t;
        s := Names[j]; Names[j] := Names[j-1];
        Names[j-1] := s;
       end;

  // вывод результата
  if Num >= 3 then j := 3 else j := Num;
  i := 1;
 while (i <= Num) and (Count[i] >= Count[j]) do
   begin
    WriteLn(Names[i], ' ', Count[i]);
    i := i + 1;
   end
end.
```

Раздел 11. **Теория игр**

С3 Анализ выигрышных ходов

Конспект

Теория игр — это раздел прикладной математики, посвящённый изучению математических методов поиска оптимальных стратегий в играх. При этом под «играми» понимаются не только собственно игры как соревнования соперников с целью развлечения, но и вообще любые процессы, в которых участвуют две или более сторон, ведущих борьбу за реализацию своих интересов. Каждая из этих сторон имеет свою цель и использует некоторую стратегию, которая может приводить к выигрышу или проигрышу в зависимости от действий соперников. Игра рассматривается как определённый математический объект, который включает игроков, набор стратегий для каждого игрока и выигрыши («платежи») игроков для каждой комбинации стратегий. Основные признаки игры как математической модели ситуации следующие:

- несколько участников;
- неопределённость поведения участников, поскольку у каждого из них возможно несколько вариантов действий;
- конфликт интересов участников;
- взаимосвязанность поведения участников, так как результат игры для каждого из них зависит от поведения всех участников;
- правила поведения, известные всем участникам.

Методы теории игр находят широкое применение в экономике, политологии, психологии, конфликтологии, биологии (при исследовании поведения животных и теории эволюции), в кибернетике и исследованиях в сфере искусственного интеллекта. Теория игр помогает выбирать наилучшие стратегии с учётом имеющейся информации о других участниках, их ресурсах и возможных действиях.

В теории игр используются две основные формы записи:

- в виде платёжной матрицы (представление игры в нормальной, или стратегической форме);
- в виде ориентированного дерева (представление игры в экстенсивной, или расширенной форме).

Платёжная матрица представляет собой двумерную матрицу, размерность которой определяется количеством возможных стратегий каждого игрока. При этом, например, строки матрицы соответствуют первому игроку, а столбцы — второму игроку. На пересечении строк и столбцов платёжной матрицы записываются выигрыши (положительные числа) и/или потери (отрицательные числа) каждого игрока при выборе им соответствующей стратегии.

Пример нормальной формы игры для двух игроков, у каждого из которых возможно по две стратегии:

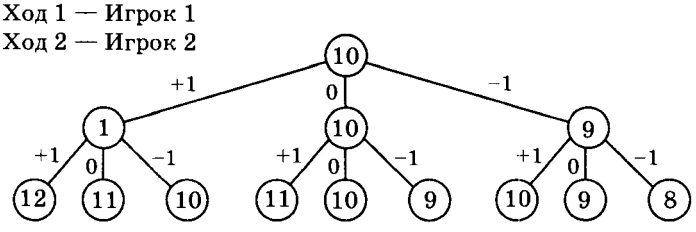
		Игрок 1	
		Стратегия 1	**Стратегия 2**
Игрок 2	**Стратегия 1**	(1, 0)	(−1, −1)
	Стратегия 2	(1, 1)	(0, 1)

В данной игре использование стратегии 1 обоими игроками даёт выигрыш в 1 очко (условную единицу) игроку 1, не давая выигрыша или проигрыша игроку 2. Использование обоими игроками стратегии 2, наоборот, даёт выигрыш в 1 очко игроку 2, не давая выигрыша или проигрыша игроку 1. Если игрок 1 выбирает стратегию 1, а игрок 2 — стратегию 2, то это обеспечивает выигрыши в 1 очко каждому. Выбор же игроком 1 стратегии 2, а игроком 2 — стратегии 1 невыгоден обоим игрокам (даёт им проигрыш в 1 очко). Анализ такой платёжной матрицы позволяет каждому игроку определить наиболее выгодный для него стиль поведения в игре, избегая явно проигрышных ситуаций и стремясь к выигрышным.

Ориентированное дерево игры представляет собой древовидный граф, корень которого соответствует исходным данным, вершины соответствуют текущим ситуациям в игре, количество ветвей от каждой вершины определяется количеством возможных стратегий каждого играющего, а уровни вершин соответствуют попеременным ходам игроков. Дерево строится до тех пор, пока все его ветви не будут завершены ситуацией, в которой достигнут выигрыш того или иного игрока.

Такой способ представления игры обладает высокой наглядностью, поэтому широко используется в решении задач по теории игр.

Пример фрагмента дерева игры для двух игроков, которые на каждом своём ходе могут выбирать один из трёх вариантов действий (прибавить 1 к исходному числу, вычесть из него 1 или оставить число без изменения; рядом с вершинами надписано текущее значение числа):



Другая форма представления информации о ходе игры — табличная, она может быть более удобной, если каждый ход может быть реализован большим числом разных вариантов и построение графа затруднено. Например, приведённому выше графу соответствует следующий фрагмент таблицы:

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1
10	11 («+1»)	11 («+1»)	
		10 («0»)	
		9 («−1»)	

Окончание табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1
	10 («0»)	11 («+1»)	
		10 («0»)	
		9 («−1»)	
	9 («−1»)	11 («+1»)	
		10 («0»)	
		9 («−1»)	

Анализ выигрышных ходов производится исходя из того, что ***оба*** игрока располагают всей необходимой информацией об игре, способны выполнить грамотный анализ оптимальной стратегии поведения в ней (т.е. «не ошибаются») и стремятся к выигрышу. Это означает, что при решении задач, связанных с анализом дерева игры, нужно «по очереди играть за каждого игрока», каждый раз выбирая наиболее выгодный для этого игрока вариант хода. Соответственно, выигрыш или проигрыш того или иного игрока в рассматриваемых в таких задачах играх однозначно зависит только от правил игры и от того, кто из игроков начинает игру («ходит первым»).

Суть анализа (игрок 1 — это игрок, который делает первый ход начиная игру):

- если в дереве игры имеется *хотя бы один путь, который ведёт от корня дерева к концевым вершинам с выигрышем игрока 1, и на этом пути нет никаких выигрышных ходов игрока 2*, то при правильной стратегии поведения в данной игре *всегда* будет выигрывать игрок, делающий первый ход, а указанный путь отмечает выигрышный для него первый ход;
- если в дереве игры *для каждого хода игрока 1 встречается хотя бы один путь, ведущий к выигрышу игрока 2*, то при правильной стратегии поведения в данной игре *всегда* будет выигрывать игрок 2, делающий второй ход, а указанный путь отмечает выигрышный для него первый ход.

Если в нескольких ветвях дерева игры получаются одни и те же значения игровых параметров, то можно пронумеровать эти значения параметров и строить для каждого из них отдельное поддерево дальнейшего хода игры.

Графическое изображение дерева игры может быть достаточно громоздким. Можно представить его в виде таблицы, которая строится слева направо; при этом соблюдается договоренность, что верхний, средний и нижний результаты очередного хода всегда соответствуют первой, второй и третьей возможным стратегиям хода игрока.

Разбор типовых задач

Задача 1*. Два игрока играют в следующую игру. На координатной плоскости стоит фишка. В начале игры фишка находится в точке с координатами $(-2, -1)$. Игроки ходят по очереди. Ход состоит в том, что игрок перемещает фишку из точки с координатами (x, y) в одну из трёх точек: $(x + 3, y)$, $(x, y + 4)$, $(x + 2, y + 2)$. Игра заканчивается, как только расстояние от фишки до начала координат превысит число 9. Выигрывает игрок, который сделал последний ход. Кто выигрывает при безошибочной игре — игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

Решение

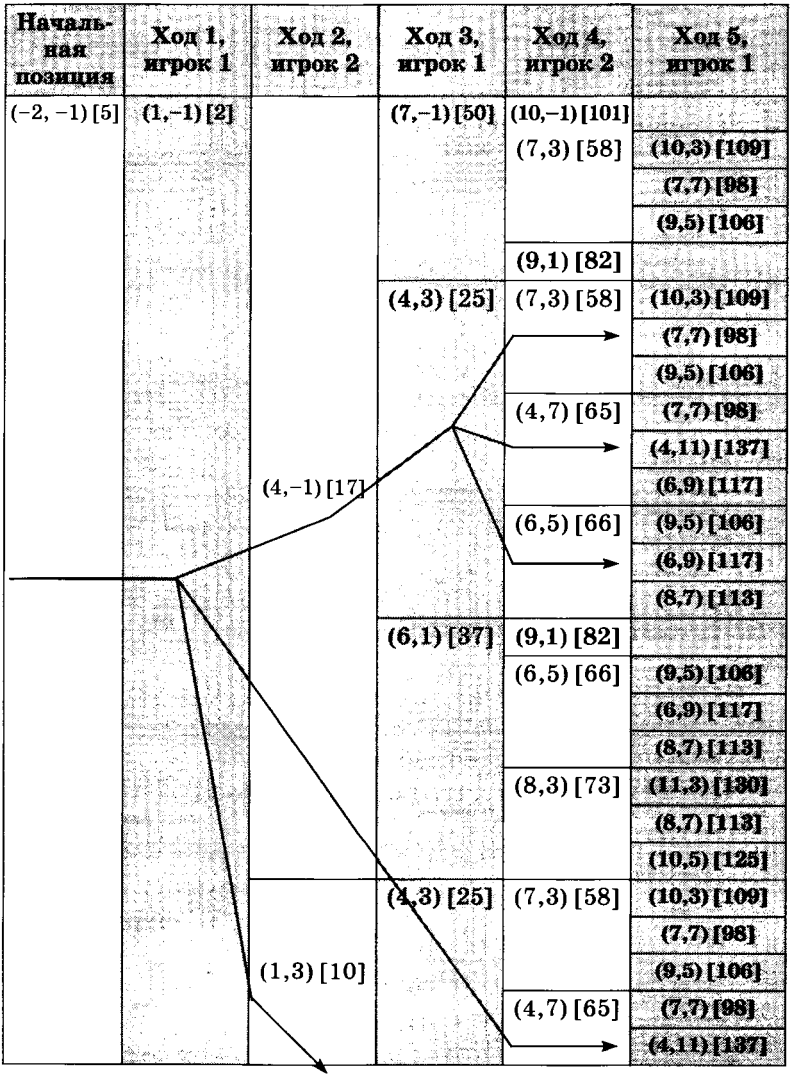
Речь идёт об игре для двух игроков, каждый из которых может в качестве хода выбрать одну из трёх возможных стратегий.

Условие завершения игры — расстояние между фишкой (точкой с текущими координатами x и y) и началом координат (точкой 0, 0) больше 9. Это расстояние можно вычислить по формуле:

$$d = \sqrt{(x_2 - x_1) + (y_2 - y_1)^2};$$

в данном случае — $d = \sqrt{x^2 + y^2}$. Чтобы не вычислять квадратный корень, удобнее заменить это условие на эквивалентное ему: «игра заканчивается, как только *квадрат расстояния* от фишки до начала координат превысит число $9^2 = 81$». При этом выигрывает игрок, который первым достиг условия завершения игры.

Строится дерево игры в виде таблицы. В круглых скобках — координаты фишки; в квадратных скобках — квадрат расстояния между фишкой и точкой (0, 0); построение ведётся до 5-го хода, когда не остаётся ни одной ветви дерева, которую можно продолжить очередным ходом:



Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(6,9) [117]
				(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
			(1,7) [50]	(4,7) [65]	**(7,7) [98]**
					(4,11) [137]
					(6,9) [117]
				(1,11) [122]	
				(3,9) [90]	
			(3,5) [34]	(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
		(3,1) [10]	**(6,1) [37]**	(9,1) [82]	
				(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
				(8,3) [73]	**(11,3) [130]**
					(8,7) [113]
					(10,5) [125]
			(3,5) [34]	(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
			(5,3) [34]	(8,3) [73]	**(11,3) [130]**

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(8,7) [113]
					(10,5) [125]
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
				(7,5) [74]	**(10,5) [125]**
					(7,9) [130]
					(9,7) [130]
	(−2,3) [13]		**(4,3) [25]**	(7,3) [58]	**(10,3) [109]**
					(7,7) [98]
					(9,5) [106]
				(4,7) [65]	**(7,7) [98]**
					(4,11) [137]
					(6,9) [117]
				(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
			(1,7) [50]	(4,7) [65]	**(7,7) [98]**
		(1,3) [10]			**(4,11) [137]**
					(6,9) [117]
				(1,11) [122]	
				(3,9) [90]	
			(3,5) [34]	(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
		(−2,7) [53]	**(1,7) [50]**	(4,7) [65]	**(7,7) [98]**
					(4,11) [137]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(6,9) [117]
				(1,11) [122]	
				(3,9) [90]	
			(–2,11) [125]		
			(0,9) [81]	(3,9) [90]	
				(0,13) [169]	
				(2,11) [125]	
			(3,5) [34]	(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
		(0,5) [25]	**(0,9) [81]**	(3,9) [90]	
				(0,13) [169]	
				(2,11) [125]	
			(2,7) 53	(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
				(2,11) [125]	
				(4,9) [97]	
	(0,1) [1]		**(6,1) [37]**	(9,1) [82]	
				(6,5) [66]	**(9,5) [106]**
					(6,9) [117]
					(8,7) [113]
		(3,1) [10]		(8,3) [73]	**(11,3) [130]**
					(8,7) [113]
					(10,5) [125]
			(3,5) [34]	(6,5) [66]	**(9,5) [106]**
					(6,9) [117]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	(8,7) [113]
					(5,11) [146]
					(7,9) [130]
			(5,3) [34]	(8,3) [73]	(11,3) [130]
					(8,7) [113]
					(10,5) [125]
				(5,7) [74]	(8,7) [113]
					(5,11) [146]
					(7,9) [130]
				(7,5) [74]	(10,5) [125]
					(7,9) [130]
					(9,7) [130]
			(3,5) [34]	(6,5) [66]	(9,5) [106]
					(6,9) [117]
					(8,7) [113]
				(3,9) [90]	
				(5,7) [74]	(8,7) [113]
					(5,11) [146]
					(7,9) [130]
		(0,5) [25]	(0,9) [81]	(3,9) [90]	
				(0,13) [169]	
				(2,11) [125]	
			(2,7) [53]	(5,7) [74]	(8,7) [113]
					(5,11) [146]
					(7,9) [130]
				(2,11) [125]	
				(4,9) [97]	
			(5,3) [34]	(8,3) [73]	(11,3) [130]
					(8,7) [113]

Окончание табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(10,5) [125]
				(5,7) [74]	**(8,7) [113]**
					(5,11) [146]
					(7,9) [130]
				(7,5) [74]	**(10,5) [125]**
					(7,9) [130]
					(9,7) [130]
			(2,7) [53]	(5,7) [74]	**(8,7) [113]**
	(2,3) [13]				**(5,11) [146]**
					(7,9) [130]
				(2,11) [125]	
				(4,9) [97]	
			(4,5) [41]	(7,5) [74]	**(10,5) [125]**
					(7,9) [130]
					(9,7) [130]
				(4,9) [97]	
				(6,7) [85]	

После анализа таблицы выясняется, что если игрок 1 своим первым ходом выберет (1, −1), то далее при *любом* ходе 2 игрока 2 игрок 1 может выбрать такой ход 3, что игрок 2 никаким ходом 4 не достигнет выигрыша (тогда как игрок 1 ходом 5 выиграет). А вот при выборе игроком 1 первого хода (−2, 3) или (0, 1) игрок 2 может сделать такой ход 2 (в частности, (0, 5) — красная пунктирная стрелка в таблице), что при любом ходе 3 игрока 1 игрок 2 имеет хотя бы один выигрышный ход 4. Поэтому:

1) игрок 1 должен первым ходом выбрать только (1, −1);
2) игрок 1 (при условии выбора этого хода и безошибочной игры) гарантированно выиграет.

Ответ: выигрывает первый игрок, своим первым ходом он должен поставить фишку в точке с координа-

тами (1, −1). Доказательством является неполное дерево игры в виде таблицы:

Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
(1,−1)	(3,1)	(5,3)	(8,3)	(11,3)
			(5,7)	(8,7)
			(7,5)	(10,5)
	(1,3)	(4,3)	(7,3)	(10,3)
			(4,7)	(7,7)
			(6,5)	(9,5)
	(4,−1)	(4,3)	То же	

Из этой таблицы видно, что при любом ходе второго игрока у первого имеется ход, приводящий к выигрышу.

Задача 2*. Два игрока играют в следующую игру. Перед ними лежат две кучки камней, в первой из которых 3, а во второй 4 камня. У каждого игрока неограниченно много камней. Игроки ходят по очереди. Ход состоит в том, что игрок или удваивает число камней в какой-то кучке или добавляет 4 камня в какую-то кучку. Игрок, после хода которого общее число камней в двух кучках становится больше 25, **проигрывает**. Кто выигрывает при безошибочной игре обоих игроков — игрок, делающий первый ход, или игрок, делающий второй ход? Каким должен быть первый ход выигрывающего игрока? Ответ обоснуйте.

Решение

Эта задача решается аналогично предыдущей.

Речь идёт об игре для двух игроков, каждый из которых может в качестве хода выбрать одну из четырёх возможных стратегий:

Кучки	Стратегия 1	Стратегия 2	Стратегия 3	Стратегия 4
X	$2 \cdot X$	X	$X + 4$	X
Y	Y	$2 \cdot Y$	Y	$Y + 4$

Условие завершения игры — сумма ($X + Y$) больше 25, причём игрок, достигший этого, проигрывает (т.е. цель игроков — **не достигнуть** завершения игры, или, что то же самое, — заставить соперника завершить игру).

Построим дерево игры в виде таблицы (в круглых скобках — количество камней в кучках — (X, Y), в квадратных скобках — суммарное количество камней в обеих кучках):

Начальная позиция	**Ход 1, игрок 1**	**Ход 2, игрок 2**	**Ход 3, игрок 1**	**Ход 4, игрок 2**	**Ход 5, игрок 1**
(3,4) [7]	(6,4) [10]	**(12,4) [16]**	(24,4) [28]		
			(12,8) [20]	**(24,8) [32]**	
				(12,16) [28]	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
			(16,4) [20]	**(32,4) [36]**	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
				(20,4) [24]	(40,4) [44]
					(20,8) [28]
					(24,4) [28]
					(20,8) [28]
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(16,12) [28]
			(12,8) [20]	(24,8) [32]	
				(12,16) [28]	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
		(6,8) [14]	(12,8) [20]	(24,8) [32]	
				(12,16) [28]	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
			(6,16) [22]	(12,16) [28]	
				(6,32) [38]	
				(10,16) [26]	
				(6,20) [26]	
			(10,8) [18]	(20,8) [28]	
				(10,16) [26]	
				(14,8) [22]	(28,8) [36]
					(14,16) [30]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(18,8) [26]
					(14,12) [26]
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
			(6,12) [18]	(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
		(10,4) [14]	(20,4) [24]	**(40,4) [44]**	
				(20,8) [28]	
				(24,4) [28]	
				(20,8) [28]	
			(10,8) [18]	**(20,8) [28]**	
				(10,16) [26]	
				(14,8) [22]	(28,8) [36]
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(10,12) [22]	(20,12) [32]
					(10,24) [34]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(14,12) [26]
					(10,16) [26]
				(28,4) [32]	
				(14,8) [22]	**(28,8) [36]**
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(18,4) [22]	**(36,4) [40]**
					(18,8) [26]
					(22,4) [26]
					(18,8) [26]
				(14,8) [22]	**(28,8) [36]**
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(20,8) [28]	
				(10,16) [26]	
				(14,8) [22]	**(28,8) [36]**
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(10,12) [22]	**(20,12) [32]**
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
				(24,8) [32]	
				(12,16) [28]	
				(16,8) [24]	**(32,8) [40]**
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
			(6,16) [22]	**(12,16) [28]**	
				(6,32) [38]	
				(10,16) [26]	
				(6,20) [26]	
			(10,8) [18]	**(20,8) [28]**	
				(10,16) [26]	
				(14,8) [22]	(28,8) [36]
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
			(6,12) [18]	**(12,12) [24]**	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(24,8) [32]	
				(12,16) [28]	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
			(12,8) [20]	**(12,12) [24]**	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(12,16) [28]	
				(6,32) [38]	
			(6,16) [22]	**(10,16) [26]**	
				(6,20) [26]	
				(20,8) [28]	
	(3,8) [11]	**(6,8) [14]**		**(10,16) [26]**	
				(14,8) [22]	(28,8) [36]
					(14,16) [30]
			(10,8) [18]		(18,8) [26]
					(14,12) [26]
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
			(6,12) [18]		(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(14,12) [26]
					(10,16) [26]
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
		(3,16) [19]	(6,16) [22]	**(12,16) [28]**	
				(6,32) [38]	
				(10,16) [26]	
				(6,20) [26]	
			(3,32) [35]		
			(7,16) [23]	**(14,16) [30]**	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
			(3,20) [23]	**(6,20) [26]**	
				(3,40) [43]	
				(7,20) [27]	
				(3,24) [27]	
		(7,8) [15]	(14,8) [22]	**(28,8) [36]**	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(7,16) [23]	**(14,16) [30]**	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
			(11,8) [19]	**(22,8) [30]**	
				(11,16) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(19,8) [27]
					(15,12) [27]
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(14,12) [26]	
				(7,24) [31]	
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27])
			(7,12) [19]		(11,16) [27]
				(7,16) [23]	(14,16) [30]
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
					(24,12) [36]
				(12,12) [24]	(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
		(3,12) [15]	(6,12) [18]		(14,12) [26]
					(10,16) [26]
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
			(3,24) [27]		
			(7,12) [19]	**(14,12) [26]**	

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(7,24) [31]	
				(11,12) [23]	**(22,12) [34]**
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(7,16) [23]	**(14,16) [30]**
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
			(3,16) [19]	**(6,16) [22]**	**(12,16) [28]**
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
				(3,32) [35]	
				(7,16) [23]	**(14,16) [30]**
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
				(3,20) [23]	**(6,20) [26]**
					(3,40) [43]
					(7,20) [27]
					(3,24) [27]
	(7,4) [11]	**(14,4) [18]**	**(28,4) [32]**		
			(14,8) [22]	**(28,8) [36]**	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(18,4) [22]	**(36,4) [40]**	
				(18,8) [26]	
				(22,4) [26]	
				(18,8) [26]	

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
			(14,8) [22]	**(28,8) [36]**	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(14,8) [22]	**(28,8) [36]**	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(7,16) [23]	**(14,16) [30]**	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
		(7,8) [15]	(11,8) [19]	**(22,8) [30]**	
				(11,16) [27]	
				(15,8) [23]	**(30,8) [38]**
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
				(11,12) [23]	**(22,12) [34]**
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
			(7,12) [19]	**(14,12) [26]**	
				(7,24) [31]	
				(11,12) [23]	**(22,12) [34]**
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(7,16) [23]	**(14,16) [30]**
					(7,32) [39]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(11,16) [27]
					(7,20) [27]
		(11,4) [15]	(22,4) [26]		
			(11,8) [19]	**(22,8) [30]**	
				(11,16) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
			(15,4) [19]	**(30,4) [34]**	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
				(19,8) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
			(11,8) [19]	**(22,8) [30]**	
				(11,16) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
				(11,12) [23]	(22,12) [34]
					(11,24) [35]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
					(15,12) [27]
					(11,16) [27]
		(7,8) [15]	(14,8) [22]	(28,8) [36]	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(7,16) [23]	(14,16) [30]	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
			(11,8) [19]	(22,8) [30]	
				(11,16) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
			(7,12) [19]	(14,12) [26]	
				(7,24) [31]	
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(7,16) [23]	(14,16) [30]
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
	(3,8) [11]	(6,8) [14]	(12,8) [20]	(24,8) [32]	

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(12,16) [28]	
				(16,8) [24]	(32,8) [40]
					(16,16) [32]
					(20,8) [28]
					(16,12) [28]
				(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
			(6,16) [22]	(12,16) [28]	
				(6,32) [38]	
				(10,16) [26]	
				(6,20) [26]	
			(10,8) [18]	(20,8) [28]	
				(10,16) [26]	
				(14,8) [22]	(28,8) [36]
					(14,16) [30]
					(18,8) [26]
					(14,12) [26]
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
			(6,12) [18]	(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
		(3,16) [19]	(6,16) [22]	(12,16) [28]	
				(6,32) [38]	
				(10,16) [26]	
				(6,20) [26]	
			(3,32) [35]		
			(7,16) [23]	(14,16) [30]	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
			(3,20) [23]	(6,20) [26]	
				(3,40) [43]	
				(7,20) [27]	
				(3,24) [27]	
		(7,8) [15]	(14,8) [22]	(28,8) [36]	
				(14,16) [30]	
				(28,8) [26]	
				(14,12) [26]	
			(7,16) [23]	(14,16) [30]	
				(7,32) [39]	
				(11,16) [27]	
				(7,20) [27]	
			(11,8) [19]	(22,8) [30]	
				(11,16) [27]	
				(15,8) [23]	(30,8) [38]
					(15,16) [31]
					(19,8) [27]
					(15,12) [27]

Продолжение табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
			(7,12) [19]	**(14,12) [26]**	
				(7,24) [31]	
				(11,12) [23]	(22,12) [34]
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(7,16) [23]	(14,16) [30]
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
		(3,12) [15]	(6,12) [18]	(12,12) [24]	(24,12) [36]
					(12,24) [36]
					(16,12) [28]
					(12,16) [28]
				(6,24) [30]	
				(10,12) [22]	(20,12) [32]
					(10,24) [34]
					(14,12) [26]
					(10,16) [26]
				(6,16) [22]	(12,16) [28]
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
			(3,24) [27]		
			(7,12) [19]	**(14,12) [26]**	

Окончание табл.

Начальная позиция	Ход 1, игрок 1	Ход 2, игрок 2	Ход 3, игрок 1	Ход 4, игрок 2	Ход 5, игрок 1
				(7,24) [31]	
				(11,12) [23]	**(22,12) [34]**
					(11,24) [35]
					(15,12) [27]
					(11,16) [27]
				(7,16) [23]	**(14,16) [30]**
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
			(3,16) [19]	**(6,16) [22]**	**(12,16) [28]**
					(6,32) [38]
					(10,16) [26]
					(6,20) [26]
				(3,32) [35]	
				(7,16) [23]	**(14,16) [30]**
					(7,32) [39]
					(11,16) [27]
					(7,20) [27]
				(3,20) [23]	**(6,20) [26]**
					(3,40) [43]
					(7,20) [27]
					(3,24) [27]

Проанализировав построенное дерево игры, можно увидеть, что при *любом* ходе игрока 1 игрок 2 имеет возможность выбрать такой ход, чтобы на пятом ходу игрок 1 свёл игру к проигрышу. Поэтому игрок 2 при безошибочной игре выиграет.

Ответ: выигрывает второй игрок. Доказательством является неполное дерево игры в виде таблицы:

Начальная позиция	Ход 1, игрок 1 (все варианты хода)	Ход 2, игрок 2 (выигрышный ход)	Ход 3, игрок 1 (все варианты хода, кроме непосредственно проигрышных)	Ход 4, игрок 2 (выигрышные ходы)	Ход 5, игрок 1
(3,4)	(6,4)	(12,4)	(12,8)	(16,8)	Любой следующий ход первого игрока — проигрышный
				(12,12)	
			(16,4)	(20,4)	
				(16,8)	
	(3,8)	(3,12)	(6,12)	(10,12)	
				(6,16)	
				(12,12)	
			(3,16)	(7,16)	
				(3,20)	
				(6,16)	
			(7,12)	(11,12)	
				(7,16)	
	(7,4)	(11,4)	(15,4)	(19,4)	
				(15,8)	
			(11,8)	(15,8)	
				(11,12)	

Из этой таблицы видно, что при любом ходе первого игрока у второго имеется ход, не приводящий к проигрышу, за которым следует проигрыш первого игрока.