О. Б. Богомолова

ИНФОРМАТИКА

СПРАВОЧНИК

для подготовки

к ЕГЭ

О. Б. Богомолова

ИНФОРМАТИКА

СПРАВОЧНИК

ДЛЯ ПОДГОТОВКИ

К ЕГЭ

УДК 373:002
ББК 32.81я721
Б74

Богомолова, Ольга Борисовна.
Б74 Информатика : справочник для подготовки к ЕГЭ / О.Б. Богомолова. — 491 ,[5] с .

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

ISBN 978-5-17-127491-7

Содержание

Предисловие 5

Раздел 1. **Информация. Измерение информации. Кодирование информации**

Измерение количества информации 7

Равномерные и неравномерные двоичные коды . . . 23

Передача информации по коммуникационным каналам 34

Раздел 2. **Моделирование и компьютерный эксперимент**

Задачи на графах 44

Раздел 3. **Системы счисления**

Двоичная, восьмеричная, шестнадцатеричная системы счисления. Арифметика в указанных системах счисления 60

Задачи на кодирование, решаемые с применением недесятичных систем счисления 81

Раздел 4. **Основы логики**

Таблицы истинности. Законы алгебры логики. Задачи, решаемые с использованием таблиц истинности 92

Решение систем логических уравнений 122

Раздел 5. **Элементы теории алгоритмов**

Анализ работы автомата, формирующего число по заданным правилам 149

Исполнители: Робот, Чертёжник, Редактор 155

Числовые исполнители 177

Раздел 6. **Архитектура компьютеров и компьютерных сетей**

Файловая система ПК 192

Основные принципы функционирования сети Интернет. Протокол TCP/IP 201

Раздел 7. **Технология обработки звуковой и графической информации**

Определение объёма и скорости передачи цифровой мультимедиа-информации 213

Раздел 8. **Обработка числовой информации**

Электронные таблицы. Ссылки. Формулы 223

Раздел 9. **Технологии поиска и хранения информации**

Базы данных. Сортировка данных. Запросы в базах данных 234

Поиск информации в сети Интернет. Поисковые запросы 257

Раздел 10. **Программирование**

Условный оператор. Циклы 272

Циклы: анализ алгоритмов 287

Операции с массивами: анализ программ 318

Процедуры и функции 380

Задачи на исправление ошибок в программах 432

Задачи на анализ и обработку данных 443

Раздел 11. **Теория игр**

Анализ выигрышных ходов 462

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 организаций среднего (полного) общего образования и вступительных экзаменов по соответствующим общеобразовательным предметам — для образовательных организаций среднего и высшего профессионального образования.

Информатика 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 растёт. 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 расчёт 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.

Каждый раздел справочника по каждой изучаемой теме включает конспект теоретического материала и разбор решений ряда типовых заданий ЕГЭ за 2–3 последних года.

Предлагаемый вашему вниманию справочник — результат многолетней педагогической практики автора. Структура справочника основана на результатах анализа тематики заданий ЕГЭ за последние несколько лет. 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.

Использование справочника рекомендуется в сочетании с дополнительным решением вариантов заданий ЕГЭ.

В связи с возможными изменениями в формате и количестве заданий рекомендуем в процессе подготовки к экзамену обращаться к материалам сайта официального разработчика экзаменационных заданий — Федерального института педагогических измерений: www.fipi.ru.

Раздел 1. Информация. Измерение информации. Кодирование информации

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 несовершении) 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:

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. $p_1 = p_2 = \ldots = p_N = p$, 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

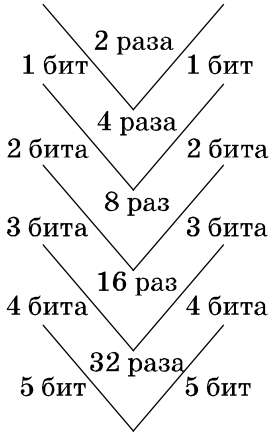
Для N равновероятных событий вероятность одного отдельного события $p = 1/N$. С учётом этого формула Хартли может быть преобразована в соотношение:

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

В этом случае вычисление количества информации можно производить по таблице со стр. 8, 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 таблице определяется, что $I = 6$ битов.

«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 ***bi**nary 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 битов), 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 **бо́льшую** 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 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.

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. 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 в байтах, который занимает хранение 60 паролей.

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 ($12 \cdot 2 = 24$ различных знака). Итого мощность используемого алфавита составляет: $10 + 12 \cdot 2 = 34$ 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: $I = 6$ битов на один знак алфавита.

3. Длина пароля (длина сообщения, кодируемого с использованием рассмотренного алфавита) равна 11 символам ($K = 11$). Тогда согласно формуле $I_S = I \cdot K$ 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.

Задача 2. В школьной компьютерной сети каждому учащемуся выдаётся пароль, состоящий из 15 восьмеричных цифр. При этом символы кодируют одинаковым минимально возможным количеством битов. В базе данных, в которой хранятся сведения о паролях, для каждого пользователя отводится одинаковое минимально возможное целое количество байтов. Кроме самого пароля, для каждого пользователя в базе данных также хранится дополнительная информация, занимающая целое количество байтов (одинаковое для всех пользователей). Для хранения сведений о 20 учащихся потребовалось 320 байтов.

Сколько байтов занимает дополнительная информация об одном пользователе? (В ответе нужно указать только целое число — количество байтов.)

Решение

Данная задача аналогична предыдущей. Отличие заключается только в наличии в записи о пользователе дополнительных байтов (количество которых и нужно найти).

1. Алфавит состоит из 8 символов (восемь восьмеричных цифр от 0 до 7). Следовательно, для хранения одного такого символа достаточно 3 бита ($2^3 = 8$).

2. В пароле содержится 15 символов, тогда для хранения собственно пароля требуется $15 \cdot 3 = 45$ битов.

3. На 20 пользователей требуется 320 байтов. Значит, на одного пользователя отводится $320/20 = 16$ байтов.

4. Дополнительная информация занимает целое количество байтов. Тогда на собственно пароль тоже должно отводиться минимально возможное количество целых байтов — а именно 6 байтов ($6 \cdot 8 = 48$ битов, а $5 \cdot 8 = 40$ битов — слишком мало).

5. Тогда получим уравнение: $16 = 6 + X$, откуда $X = 10$ байтов отводится на дополнительную информацию.

Ответ: 10.

Задача 3. Вес выпускаемых деталей может меняться от 250 до 310 граммов. Компьютер при маркировке деталей записывает данные о каждой из них в базу данных, присваивая деталям порядковые номера в пределах рабочей смены — от 1 до 2000. Для хранения номера детали и веса детали в граммах программа использует минимально возможные количества битов.

Определите информационный объём в битах массива из 1500 записей о промаркированных деталях.

Решение

Эта задача даже проще, чем предыдущие, но содержит небольшую хитрость.

1. Для хранения номера детали (от 1 до 2000) требуется как минимум 11 битов ($2^{11} = 2048$, а 10 битов мало, так как $2^{10} = 1024$).

2. Для хранения веса детали (от 250 до 310 граммов) могло бы потребоваться целых 9 битов (числа до 512). Но мы же можем в нашей программе заранее заложить «базовую» константу 250 и в базе данных хранить не сам вес, а разницу между действительным весом и этой «базовой» константой! То есть при занесении информации в базу данных мы вычитаем из веса детали число 250 и записываем в память эту разность, а при последующей выдаче результатов по запросу — наоборот, прибавляем «базовую» константу 250 к хранящемуся значению.

Тогда запоминаемые значения веса будут меняться уже от 0 до 60 (310 − 250), а для них достаточно всего 6 битов ($2^6 = 64$). Итого — целых 3 бита экономии на каждой детали!

3. Тогда одна запись о детали имеет объём: 11 бит номера + 6 битов веса = 17 бит.

4. Для хранения 1500 таких записей потребуется $1500 \cdot 17 = 25\,500$ битов.

Ответ: 25 500.

Задача 4. При регистрации на сайте для каждого пользователя формируется учётная запись и выдаётся

пароль. Длина пароля составляет 12 символов, при этом в пароле могут использоваться **строчные и заглавные латинские буквы, хотя бы три цифры и хотя бы два символа** из набора: @, #, $,%, *, & и ~. Для хранения каждого 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. Кроме собственно пароля, учётная запись содержит некоторую дополнительную информацию, объём которой составляет целое число байтов.

Известно, что для хранения всех учётных записей 40 пользователей потребовалось 1200 байтов.

Определите объём памяти в байтах, выделенный для хранения дополнительной информации в учётной записи одного пользователя. (В ответе необходимо записать только число — количество байтов.)

Для справки: латинский алфавит содержит 26 букв.

Указания к решению:

Что изменилось в условии? Если раньше пароль мог состоять из произвольного набора знаков (букв, цифр, дополнительных символов), то теперь оговорено, что в пароле обязательно должно быть ***не менее указанного*** количества цифр и ***не менее указанного*** количества дополнительных символов. Однако точное количество цифр и дополнительных символов мы не знаем, а в условии по-прежнему записано, что каждый символ занимает в памяти ***одинаковое*** и минимально возможное количество битов.

Поэтому мы по-прежнему должны учитывать, что наш алфавит (набор символов) для пароля включает в себя ***все*** возможные знаки и имеет мощность 69 знаков. А дальнейший ход решения абсолютно тот же, что и в предыдущих задачах.

Ответ: 19.

Задача 5. При регистрации на сайте для каждого пользователя формируется учётная запись и выдаётся пароль. Длина пароля составляет 12 символов, при этом в пароле могут использоваться строчные и заглавные латинские буквы, **ровно три** цифры и **ровно два** символа из набора: @, #, $, %, *, & и ~. Для хранения каждого пароля на компьютере отводится минимально возможное и одинаковое целое количество байтов, при этом используется посимвольное кодирование и все символы **кодируются минимально возможным** количеством битов. Кроме собственно пароля, учётная запись содержит некоторую дополнительную информацию, объём которой составляет целое число байтов.

Известно, что для хранения всех учётных записей 40 пользователей потребовалось 1200 байтов.

Определите объём памяти в байтах, выделенный для хранения дополнительной информации в учётной записи одного пользователя. (В ответе необходимо записать только число — количество байтов.)

Для справки: латинский алфавит содержит 26 букв.

Решение

Изменение условия, казалось бы, совсем пустячное. Вместо «хотя бы» записано «**ровно**» и исчезло указание о кодировании символов *одинаковым* количеством битов. Но теперь решать задачу нужно совершенно иначе!

1.1. Мы точно знаем, что три символа пароля из двенадцати — это цифры. Цифр всего 10 (имеется в виду десятичная система счисления), поэтому для хранения одной цифры достаточно **4 битов** ($2^4 = 16$, а $2^3 = 8$ и этого недостаточно).

1.2. Мы также точно знаем, что два символа пароля берутся из 7-символьного набора дополнительных знаков. Для хранения такого символа достаточно **3 битов** ($2^3 = 8$).

1.3. Остальные символы пароля (их $12 - 3 - 2 = 7$) — это буквы. Мощность латинского алфавита, включающего и строчные, и заглавные буквы, составляет

$26 + 26 = 52$ символа. Для их хранения требуется **6 битов** ($2^6 = 64$, а $2^5 = 32$, этого мало).

2. Тогда для хранения всего 12-символьного пароля нам понадобится:

$$\underbrace{3 \times 4}_{\text{цифры}} + \underbrace{2 \times 3}_{\text{доп. символы}} + \underbrace{7 \times 6}_{\text{буквы}} = 60 \text{ битов.}$$

Этим 60 битам соответствует **8 байтов** ($8 \times 8 = 64$).

Дальнейший ход решения аналогичен предыдущим задачам.

3. Если для хранения учётных записей 40 пользователей потребовалось 1200 байтов, то для хранения учётной записи одного пользователя требуется $1200 / 40 =$ **= 30 байтов.**

4. Тогда если пароль занимает 8 байтов, а вся учётная запись — 30 байтов, то на хранение дополнительной информации о пользователе расходуется $30 - 8 =$ **= 22 байта.**

Ответ: 22.

Задача 6. При регистрации на сайте для каждого пользователя формируется учётная запись и выдаётся пароль. Длина пароля составляет 12 символов, при этом в пароле могут использоваться **строчные и заглавные латинские буквы, хотя бы три цифры и хотя бы два символа** из набора: @, #, $, %, *, & и ~. Для хранения каждого пароля на компьютере отводится минимально возможное и одинаковое целое количество байтов, при этом используется посимвольное кодирование и все символы кодируются минимально возможным количеством битов **(возможно, неодинаковым)**. Кроме собственно пароля, учётная запись содержит некоторую дополнительную информацию, объём которой составляет целое число байтов.

Известно, что для хранения всех учётных записей 40 пользователей потребовалось 1200 байтов.

Определите объём памяти в байтах, выделенный для хранения дополнительной информации в учётной записи одного пользователя. (В ответе необходимо записать только число — количество байтов.)

Для справки: латинский алфавит содержит 26 букв.

Решение

Отличие данного условия от задачи 2 только в явном указании, что для хранения символов может использоваться ***неодинаковое*** количество битов. Но точное количество цифр и дополнительных знаков (как было в задаче 3) мы не знаем. Как решать?

Главная хитрость здесь в том, что мы знаем о гарантированном наличии в пароле указанных количеств цифр и дополнительных символов. Именно на них можно сэкономить (как делалось в задаче 3).

1.1. Мы знаем, что три символа пароля — это точно цифры. Цифр всего 10 (имеется в виду десятичная система счисления), поэтому для хранения одной цифры достаточно **4 битов** ($2^4 = 16$, а $2^3 = 8$ и этого недостаточно).

1.2. Мы также знаем, что два символа пароля точно берутся из 7-символьного набора дополнительных знаков. Для хранения такого символа достаточно **3 битов** ($2^3 = 8$).

1.3. Остальные символы пароля (их $12 - 3 - 2 = 7$) могут быть ***любыми*** — буквами, цифрами или дополнительными символами. Мощность такого объединённого алфавита (как мы вычислили в задачах 1 и 2) составляет 69 символов. Для их хранения требуется **7 битов** ($2^7 = 128$, а $2^6 = 64$, этого мало).

2. Тогда для хранения всего 12-символьного пароля нам понадобится:

$$\underbrace{3 \times 4}_{\text{цифры}} + \underbrace{2 \times 3}_{\substack{\text{доп.} \\ \text{символы}}} + \underbrace{7 \times 7}_{\substack{\text{все} \\ \text{остальное}}} = 67 \text{ битов.}$$

Этим 67 битам соответствует **9 байтов** ($9 \times 8 = 72$).

3. Если для хранения учётных записей 40 пользователей потребовалось 1200 байтов, то для хранения учётной записи одного пользователя требуется 1200 / 40 = = **30 байтов**.

4. Тогда если пароль занимает 9 байтов, а вся учётная запись — 30 байтов, то на хранение дополнительной информации о пользователе расходуется 30 − 9 = = **21 байт**.

Ответ: 21.

Итак, нужно очень внимательно читать условия задач.

• Если сказано, что каждый символ кодируется **одинаковым** минимально необходимым количеством битов, то в любом случае надо вычислять **общий** объём используемого алфавита и найденное по нему «универсальное» количество битов на символ умножать на всю длину пароля.

• Если не сказано, что символы кодируются одинаковым числом битов или тем более указано, что количество битов на символ, **возможно**, **неодинаково**, то надо отдельно просчитать количество битов на символ для цифр, дополнительных знаков и т.д., для которых в условии указано «не менее», «хотя бы» или «ровно». А пароль нужно разделять на соответствующие фрагменты, обсчитывать каждый такой фрагмент по отдельности и затем суммировать полученные количества битов.

• Если количества цифр или дополнительных знаков указаны **точные**, то остальная часть пароля считается **только как буквы**. Если же для цифр и/или дополнительных знаков указано **«хотя бы»** (либо **«как минимум»**), то для оставшейся части пароля вычисляется **общий объём алфавита** по всем возможным символам.

Задача 7. Каждому школьнику выдаётся электронный пропуск, содержащий индивидуальный логин этого школьника, код, обозначающий класс и параллель, а также некоторая дополнительная информация. Логин состоит из 13 символов, каждый из которых может быть одной из 26 заглавных латинских букв. Для записи логина отведено минимально возможное целое число бай-

тов, при этом используют посимвольное кодирование, все символы кодируют одинаковым минимально возможным количеством бит. Данный код — это целое число от 1 до 180, записанное как двоичное число и занимающее минимально возможное целое число байтов. Всего в пропуске хранится 32 байта данных. Сколько байтов выделено для хранения дополнительных сведений?

Решение

1. Рассматриваем отдельно логин.

Алфавит состоит из 26 латинских заглавных букв, следовательно, потребуется 5 бит на один символ.

Всего в логине 13 символов — значит, под них нужно отвести $5 \times 13 = 65$ битов, или 9 байтов (8 байтов — недостаточно).

2. Код класса и параллели — это число от 1 до 180. Для хранения таких чисел в двоичном формате достаточно 1 байта.

3. На всю указанную выше информацию требуется: $9 + 1 = 10$ байтов.

4. Если на всю запись о школьнике отводится 32 байта, то дополнительные сведения занимают: $32 - 10 = 22$ байта.

Ответ: 22.

Равномерные и неравномерные двоичные коды

Конспект

Равномерные и неравномерные двоичные коды. 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 отведённых 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 всё 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 (поочерёдного, 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 учёта 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.

Дерево Фано

Дерево Фано — это удобный и наглядный способ решения задач, связанных с подбором неравномерных двоичных кодов.

Основные принципы построения дерева Фано

1. Учёт частоты встречаемости символов в тексте — чем чаще встречается какой-либо символ, тем короче для него должен быть код и тем раньше этот символ надо поместить в дерево.

2. Каждый узел дерева Фано порождает ровно две ветви (т.е. дерево Фано является *бинарным*), при этом одной ветви (например, левой) сопоставляется бит 0, а другой ветви — бит 1.

3. На каждом новом этапе ветвления, кроме самого последнего, одна ветвь может быть завершена каким-то символом из тех, для которых генерируются коды, но вторая ветвь обязательно должна служить продолжением дерева (иначе его не удастся построить для осталь-

ных символов). Только два последних символа рассматриваемого алфавита можно поместить на концах двух последних ветвей, тем самым «закрыв» и дерево, и генерацию кодов.

4. Для каждого символа код Фано получается последовательной записью всех нулей и единиц по кратчайшему пути от вершины дерева к соответствующему символу.

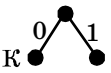
Разбор типовых задач

Задача 1. Для кодирования последовательности символов, состоящей из букв К, И, Н, О, используется неравномерный двоичный код, удовлетворяющий условию Фано. При этом для буквы К использован код 0, а для буквы И — код 11. Требуется определить наименьшую возможную суммарную длину всех кодовых слов указанных букв.

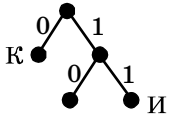
Решение

Решение задач, в которых требуется поиск кратчайших кодов, удовлетворяющих условию Фано, удобно и наглядно выполнять при помощи *дерева Фано*.

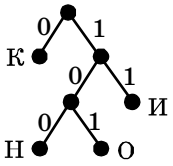
1. Начинаем построение дерева Фано. При этом по условию, код 0 уже занят для буквы К. Поэтому помещаем этот символ на конце соответствующей ветви. Вторая ветвь послужит для продолжения построения дерева.



2. Продолжая построение правой ветви дерева, получаем две ветви — 0 и 1. Но двум ветвям 1 и 1 соответствует код 11, которому, по условию, соответствует буква И. Помещаем её на конце соответствующей ветви.

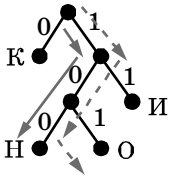


3. Два оставшихся символа мы можем разместить на двух последних ветвях дерева, построенных слева. При этом мы «закрываем» дерево, и больше никакие символы добавить в данный алфавит уже будет нельзя.



Это один из возможных вариантов построения дерева Фано: правильным решением было бы и, наоборот, поместить букву О слева, а букву Н — справа. Но в данной задаче требуется определить суммарную длину всех кодовых слов, поэтому порядок размещения самих добавляемых символов (их конкретные коды) не важен.

4. Пользуясь построенным деревом, записываем коды Фано для добавленных букв: Н — код 100 (сплошные стрелки на рисунке), О — код 101 (пунктирные стрелки).



5. Вычисляем суммарную длину кодов, помня, что код буквы К — 0, а буквы И — 11:

$1 + 2 + 3 + 3 = 9$.

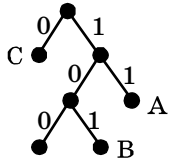
Ответ: 9.

Задача 2. По каналу связи передаются сообщения, содержащие шесть букв: A, B, C, D, E, F. Для передачи используется неравномерный двоичный код, удовлетворяющий условию Фано. Для букв A, B, C используются кодовые слова 11, 101 и 0 соответственно.

Укажите кодовое слово наименьшей возможной длины, которое можно использовать для буквы F. Если таких слов несколько, то укажите то из них, которое соответствует наименьшему возможному двоичному числу.

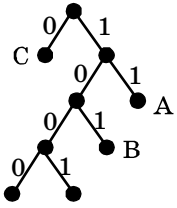
Решение

Начинаем с построения дерева Фано. При этом сразу размещаем на его ветвях заданные символы по их известным кодам.



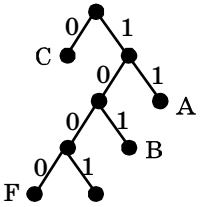
(На первом шаге мы поместили в ветвь 0 символ C, так как для него задан код Фано, равный 0. На втором шаге, поскольку код Фано для символа A равен 11, мы этот символ поместили на ветви 1, чтобы путь от вершины к A давал нам две единицы. На третьем шаге определяется место для символа B: его код 101, значит, после ветвления по ветви 1, а потом по ветви 0 нужно B поместить на конце ветви 1.)

Теперь строим новое ветвление от левой (оставшейся свободной) ветви:



Поскольку код для буквы F должен быть кратчайшим, именно эту букву нужно закрепить на одной из двух только что построенных ветвей. Но на какой — на 0 или на 1?

Здесь становится важным требование: код буквы F должен быть наименьшим двоичным числом. Поэтому нужно использовать ветвь 0, а не ветвь 1:



Остальные коды символов нас в условии задания не интересуют, хотя можно достроить дерево, включив в него эти символы (D и E).

Записываем по построенному дереву искомый код Фано для символа F: это код 1000.

Ответ: 1000.

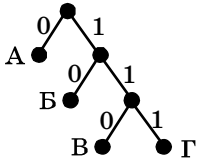
Задача 3. В сообщении встречается 50 букв А, 30 букв Б, 20 букв В и 5 букв Г. При его передаче использован неравномерный двоичный префиксный код, который позволил получить минимальную длину закодированного сообщения. Какова она в битах?

В этой задаче вообще нет изначально заданных кодов, но зато учитывается частота встречаемости букв. Напомним, что термин «префиксный» означает, что речь идёт о прямом условии Фано. Спрашивается же общая длина в битах всего получаемого сообщения, закодированного с использованием полученных кодов Фано.

Решение

Вначале нам не важны конкретные количества букв — важна только общая статистика, чтобы выстроить буквы по порядку уменьшения частоты их встречаемости в тексте. В нашем случае этот порядок уже задан: А, Б, В, Г, — но могут встретиться и задания, где правильный порядок букв придётся выстроить самому.

Теперь «выращиваем» дерево Фано, начиная с самых частых букв к менее частым:



По этому дереву получаем коды букв: А — 0, Б — 10, В — 110, Г — 111. (Коды могут быть и другими — важно, что для более частых букв коды должны быть короче.)

Теперь определяем общую длину сообщения в битах:

50 букв А с кодом из 1 бита + 30 букв Б с кодом из 2 бит + 20 букв В с кодом из 3 бит + 5 букв Г с кодом также из 3 бит = $50 \times 1 + 30 \times 2 + 20 \times 3 + 5 \times 3 = 50 + 60 + 60 + 15 = 185$ бит.

Ответ: 185.

Задача 4. При передаче информации используется равномерный двоичный код. Передаче подлежат сообщения, которые могут состоять только из четырёх букв: «И», «Н», «Ф», «О», при этом каждой букве соответствует отдельное кодовое слово.

Для используемого набора кодовых слов выполняется обязательное правило: любые два кодовых слова из этого набора должны различаться как минимум в трёх разрядах.

Для кодирования букв «И», «Н», «О» используются следующие 5-битовые кодовые слова:

«И»: 11110, «Н»: 10000, «О»: 01001. Про 5-битовый код для буквы «Ф» известно, что оно начинается с 0 и заканчивается 1.

Укажите кодовое слово для буквы «Ф».

Решение

В задачах этого типа речь идёт об обычном двоичном коде, длина которого одинакова для всех кодируемых

символов. Условия Фано в этом случае не используются, а для решения задачи достаточно обычных рассуждений.

1. По условию, *любые два кодовых слова из используемого набора различаются как минимум в трёх разрядах*. Запишем предполагаемый код буквы «Ф», заменяя неизвестные пока биты звёздочками: «Ф»: **0***1**.

2. Сравним этот код с известными кодами других букв:

«И»: **11110**
«Н»: **10000**
«О»: **01001**
«Ф»: **0***1**

Видим, что начальный и конечный биты кода буквы «Ф» не совпадают с соответствующими битами кодов букв «И» и «Н», но совпадают (причём оба!) с начальным и конечным битами для буквы «О».

3. По условию нужно, чтобы коды «О» и «Ф» различались как минимум в трёх позициях. А такое различие нам могут дать только три внутренних бита.

Тогда для «Ф» вместо звёздочек надо выбрать *биты, противоположные соответствующим битам* для «О»:

«О»: **01001**
«Ф»: **00111**

3. Для кодов «Ф» и «О» требуемое условие соблюдено. Проверим его соблюдение для кодов остальных букв:

«И»:	**11110**	«Н»:	**10000**
«Ф»:	**00111**	«Ф»:	**00111**
3 различия		4 различия	

Таким образом, условие соблюдено и для этих кодов.

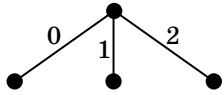
Ответ: 10110.

Задача 5. Для кодирования букв А, Б, В, Г, Д, Е, Ж, З, И, использован неравномерный троичный код, удовлетворяющий условию Фано. Для буквы А используется кодовое слово 0; для буквы Б используется кодовое слово 10; для буквы В используется кодовое слово 11; для буквы Г используется кодовое слово 21; для буквы Д используется кодовое слово 22. Какова минимальная общая длина кодовых слов для букв Е, Ж, З, И?

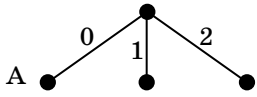
Решение

Задача, как и предыдущие, решается построением дерева Фано. Но если ранее использовался двоичный код и дерево, соответственно, было бинарным, то теперь требуется построение «троичного» дерева, в котором каждый узел порождает три возможные ветви.

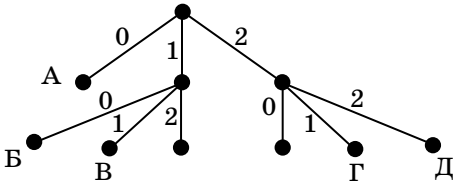
1. Начинаем строить дерево от некоторой начальной точки:



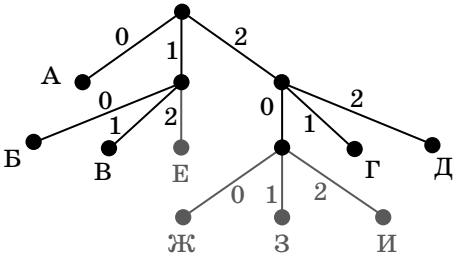
2. Поскольку для А использован код 0, сразу помещаем её на левую ветвь, тем самым «закрывая» её:



3. Продолжаем построение двух оставшихся ветвей, сразу размещая на них буквы Б, В, Г и Д, для которых заданы соответствующие коды:



4. Остаётся разместить в дереве четыре оставшиеся буквы Е, Ж, З, И. Одну из них можно поместить на одной из свободных ветвей («закрыв» её), а остальные три — на продолжении второй свободной ветви:



5. Подсчитываем суммарную длину кодов для букв Е, Ж, З, И: Е — 12, Ж — 200, З — 201, И — 202.

Суммарная длина кодов = 2 + 3 + 3 + 3 = 11.

Ответ: 11.

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 (*приёмнику информации*). При этом информация передаётся 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 та-

ких 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]) 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 неё 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

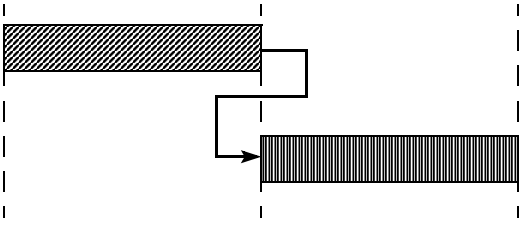
---

[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.

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 МБ по высокоскоростному каналу и ретранслирует их Сергею по низкоскоростному каналу.

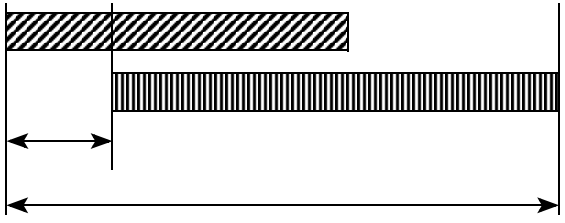
Компьютер Кати может начать ретрансляцию данных не раньше, чем им будут получены первые 1024 КБ этих данных. Каков минимально возможный промежуток времени (в секундах) с момента начала скачивания Катей данных до полного их получения Сергеем?

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.

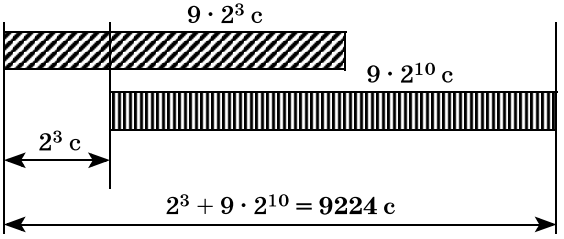
Длительность процесса 1: $9 \cdot 2^{23} / 2^{20} = 9 \cdot 2^3$ с.

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.

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. Данные объёмом 80 Мбайт передаются из пункта А в пункт Б по каналу связи, обеспечивающему скорость передачи данных 2^{23} бит в секунду, а затем из пункта Б в пункт В по каналу связи, обеспечивающему скорость передачи данных 2^{20} бит в секунду. От начала передачи данных из пункта А до их полного получения в пункте В прошло 13 минут.

Через какое время в секундах началась передача данных в пункте Б, т.е. каково время между началом передачи данных из пункта А и началом передачи данных в пункт В? В ответе укажите только число, слово «секунд» или букву «с» добавлять не нужно.

Решение

Если в предыдущих задачах нужно было искать общее время передачи данных из А в В, то теперь требуется составлять уравнение с одним неизвестным.

1) Построим диаграмму Ганта для этой задачи:

При этом следует учитывать, что процесс передачи данных из Б в В по условию задачи начинается уже после того, как завершится передача данных из А в Б.

2) Составляем по этой схеме уравнение (t — время задержки между окончанием приёма данных из А и началом их передачи в В). Для этого вычисляем время передачи данных из А в Б и из Б в В:

- из А в Б: 80 (Мбайт) / 2^{23} (бит в секунду) = $= 5 \cdot 2^{27}$ (бит) / 2^{23} (бит в секунду) $= 5 \cdot 2^4 = 80$ (с);
- из Б в В: 80 (Мбайт) / 2^{20} (бит в секунду) = $= 5 \cdot 2^{27}$ (бит) / 2^{20} (бит в секунду) $= 5 \cdot 2^7 = 640$ (с).

Записываем уравнение:

$80 + t + 640 = 13$ (минут) $= 13 \cdot 60$ (с), т.е. $720 + t = 780$, откуда $t = 60$.

3) Чтобы найти время от начала передачи данных из А в Б до начала их передачи из Б в В, нужно к най-

денному значению t прибавить время передачи данных из А в Б: 80 + 60 = 140 (с).

Ответ: 140.

Задача 3. 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:

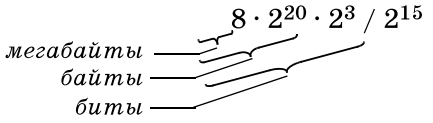
$$6 + (0{,}4 \cdot 8 \cdot 2^{20} \cdot 2^3 / 2^{15}) + 3$$

мегабайты — 8; *байты* — $2^{20} \cdot 2^3$; *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% ис-

ходного 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 4. Документ объёмом 50 Мб можно передать с одного компьютера на другой двумя способами:

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

Б) сжать *суперархиватором*, передать *суперархив* по каналу связи, распаковать его.

При этом:

- скорость передачи данных по каналу связи составляет 2^{22} бит/с;
- объём архива составляет 75% от исходного файла;
- объём суперархива составляет 50% от исходного файла;
- сжатие документа архиватором занимает 15 с, а распаковка — 10 с;
- сжатие документа суперархиватором занимает 30 с, а распаковка — 15 с.

Какой способ быстрее и насколько?

В ответе надо записать сначала букву, обозначающую соответствующий способ, а затем сразу после неё записать количество секунд, на сколько этот способ быстрее другого.

Например, если способ А быстрее способа Б на 30 секунд, то надо записать ответ в виде А30.

Решение

Задача решается аналогично предыдущей, только здесь лучше честно вычислить время, затрачиваемое на сжатие, передачу и распаковку информации в каждом из двух способов, а потом сравнить результаты.

А) Тратим 25 секунд на сжатие/распаковку, а затем передаём 75% от 50 МБ со скоростью 2^{22} бит/с:

$$25\ (\text{с}) + 0{,}75 \cdot 50\ (\text{МБ}) \,/\, 2^{22}\ (\text{бит/с}) =$$

$$= 25\ (\text{с}) + \frac{3}{4} \cdot 50 \cdot 2^{23}\ (\text{бит}) \,/\, 2^{22}\ (\text{бит/с}) =$$

$$= 25 + \frac{3}{2^2} \cdot 25 \cdot 22^{23}/2^{22} = 25 + \frac{3 \cdot 25 \cdot 2^{24}}{2^{24}} =$$

$$= 25 + 3 \cdot 25 = 25 + 75 = 100\ (\text{с}).$$

Б) Тратим 45 секунд на сжатие/распаковку, а затем передаём 50% от 50 МБ со скоростью 2^{22} бит/с:

$$45\ (\text{с}) + 0{,}5 \cdot 50\ (\text{МБ}) \,/\, 2^{22}\ (\text{бит/с}) =$$

$$= 45\ (\text{с}) + \frac{1}{2} \cdot 50 \cdot 2^{23}\ (\text{бит}) \,/\, 2^{22}\ (\text{бит/с}) =$$

$$= 45 + \frac{1}{2} \cdot 25 \cdot 2 \cdot 2^{23}/2^{22} = 45 + \frac{25 \cdot 2^{24}}{2^{23}} =$$

$$= 45 + 2 \cdot 25 = 45 + 50 = 95\ (\text{с}).$$

Сравнивая способы А и Б, получаем, что быстрее способ Б на 5 секунд.

Ответ: Б5.

Раздел 2. **Моделирование и компьютерный эксперимент**

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.

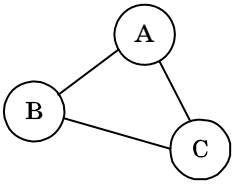
Связный граф — граф, в котором для любых двух вершин существует связывающий их путь.

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.

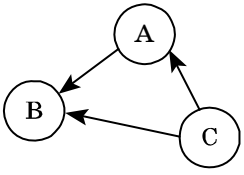
Пример неориентированного графа:



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.**

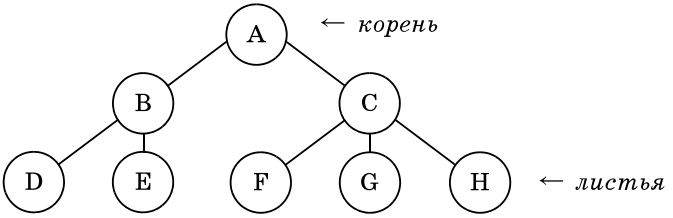
Пример ориентированного графа (является направленным, турниром):



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:

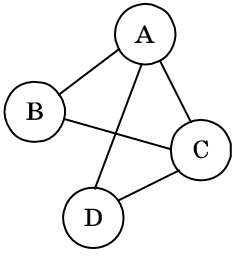


Двоичное дерево — ориентированное дерево, в котором для каждой вершины количество исходящих из неё дуг не превосходит двух.

Способы представления графов

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

Пример:



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

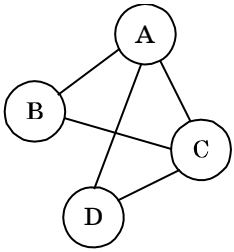
Пример:

{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.

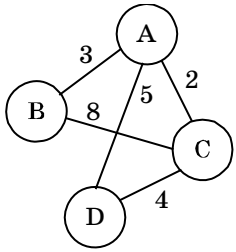
Пример:



	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:



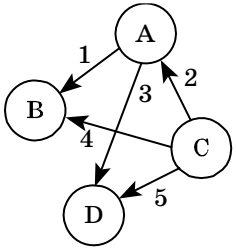
	A	B	C	D
A		3	2	5
B	3		8	
C	2	8		4
D	5		4	

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.

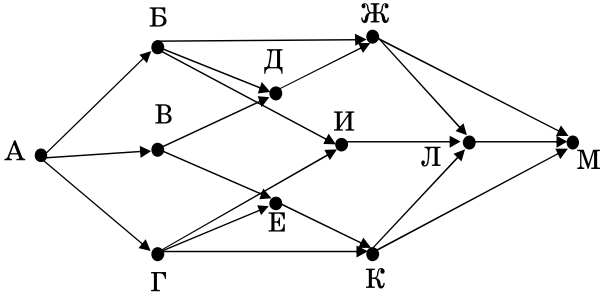
Пример:



	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. Дана схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К, Л, М. По каждой дороге можно двигаться только в направлении, указанном стрелкой. Сколько возможно различных путей из города А в город М?

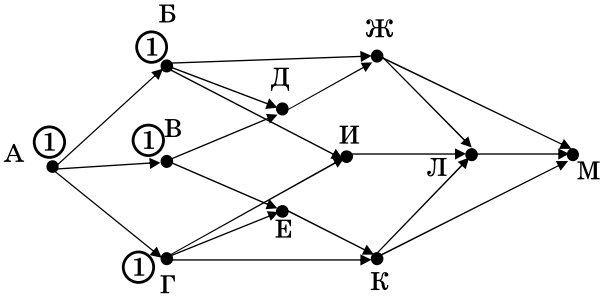


Решение

1) Возле города А записываем единицу. Это — некое «стартовое» значение, поскольку уж один-то путь из А в М есть всегда.

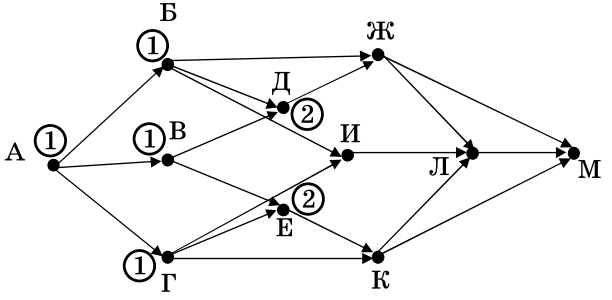
2) Смотрим город Б. В этот узел графа входит только одна стрелка, которая идёт от узла А со значением 1. Поэтому возле узла Б тоже записываем единицу.

3) То же с городами (узлами) В и Г — в них по единственной входящей стрелке «переносится» всё та же единица из узла А.



4) Смотрим город Д. В него входит две стрелки. Одна идёт из узла Б и «несёт с собой» оттуда единицу. Вторая же аналогичным способом переносит единицу по стрелке из узла В. Итого в узле Д в сумме получается значение 2 (1+1).

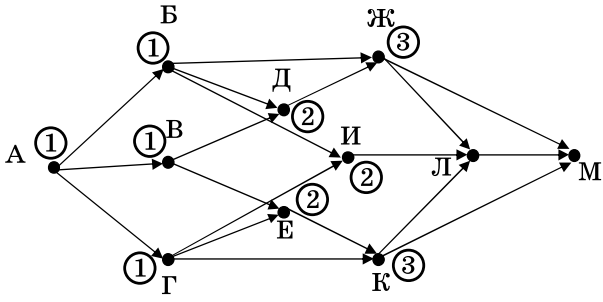
5) То же самое получается и для узла Е, куда по соответствующим двум стрелкам «приходят» единицы из узлов В и Г.



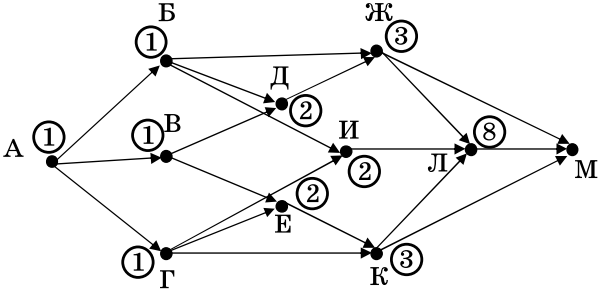
6) В узел Ж тоже входят две стрелки. Одна (из узла Б) «приносит» туда единицу. А вторая (из узла Д) «приносит» уже двойку. Итого в сумме получается 3 (1+2).

7) Для узла К история та же — рядом с ним тоже записываем 3 (1 по стрелке из узла Г плюс 2 по стрелке из узла Е).

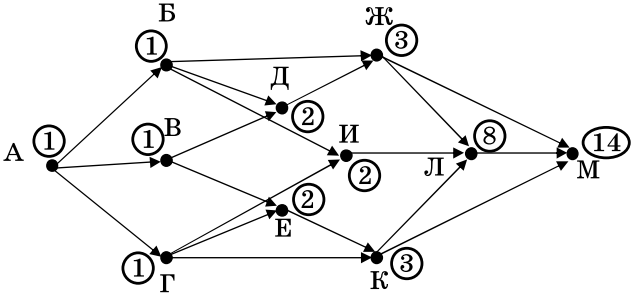
8) Для узла же И две стрелки «принесут» с собой из узлов Б и Г по единице каждая, итого в сумме получаем 2.



9) А теперь переходим к самому сложному узлу — Л. В него входит три стрелки. Первая, из узла Ж, «переносит» в Л тройку. Вторая, из узла К, «переносит» тоже тройку. И, наконец, третья, из узла И, «переносит» в Л двойку. В сумме же для Л получается значение 8 (3+2+3).

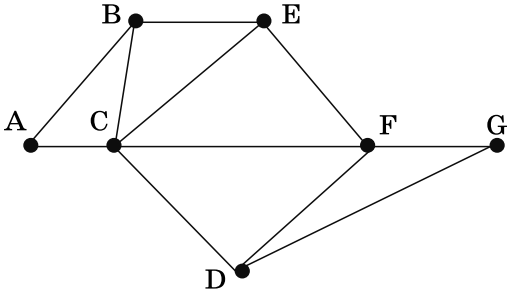


10) Остаётся узел М. В него приходит тоже три стрелки. Стрелка из узла Ж «приносит» в М тройку. Стрелка из узла К «приносит» в М тоже тройку. А стрелка из узла Л приносит в М восьмёрку. В сумме для М получается 14 (3+3+8).



Ответ: 14.

Задача 2. Схема дорог в некоторой области изображена в виде графа, а в таблице даны сведения о длинах этих дорог в километрах.



	X_1	X_2	X_3	X_4	X_5	X_6	X_7
X_1		20		17			
X_2	20			33		45	
X_3					8	70	5
X_4	17	33				11	22
X_5			8			25	
X_6		45	70	11	25		30
X_7			5	22		30	

Эту таблицу и схему создавали разные люди независимо друг от друга, и обозначения посёлков в таблице никак не связаны с их обозначениями на графе.

Требуется определить длину дороги из пункта C в пункт D. Ответом является целое число — длина этой дороги.

Решение

1. Заметим, что C — это единственный пункт с пятью рёбрами. Ищем в таблице такой столбец (или строку), в котором содержится 5 значений. Тогда C — это X_6.

2. Аналогично, F — это единственный пункт с четырьмя рёбрами. Тогда очевидно, что F — это X_4.

3. С учётом этого получаем таблицу в следующем виде (ребро CF отмечаем закраской):

	X_1	X_2	X_3	**F**	X_5	**C**	X_7
X_1		20		17			
X_2	20			33		45	
X_3					8	70	5
F	17	33				11	22
X_5			8			25	
C		45	70	11	25		30
X_7			5	22		30	

4. Пункт F, кроме пункта C, также связан с пунктами D, E и G. При этом пункт G — единственный среди них, имеющий только 2 ребра. Просматриваем в таблице строку, соответствующую пункту F:

F	17	33				11	22

и ищем в столбцах, в которых в данной строке есть числовые значения (т.е. в графе есть рёбра между пунктом F и соответствующими другими пунктами), в каком столбце есть только два числа. Это — первый столбец. Значит, G — это X_1:

	G	X_2	X_3	**F**	X_5	**C**	X_7
G		20		17			
X_2	20			33		45	
X_3					8	70	5
F	17	33				11	22
X_5			8			25	
C		45	70	11	25		30
X_7			5	22		30	

5. Пункт G связан с D и с F. Но столбец, соответствующий пункту F, мы уже определили. Тогда пункту D соответствует обозначение X_2 (также отмечаем в таблице все уже найденные рёбра):

	G	**D**	X_3	**F**	X_5	**C**	X_7
G		20		17			
D	20			33		45	
X_3					8	70	5
F	17	33				11	22
X_5			8			25	
C		45	70	11	25		30
X_7			5	22		30	

6. Тогда методом исключения получаем, что пункту E соответствует обозначение X_7:

	G	**D**	X_3	**F**	X_5	**C**	**E**
G		20		17			
D	20			33		45	
X_3					8	70	5
F	17	33				11	22
X_5			8			25	
C		45	70	11	25		30
E			5	22		30	

7. С пунктом C из числа ещё не рассмотренных связаны два пункта — B и A, причём пункт A — единственный с двумя рёбрами. Рассуждая аналогично предыдущему (см. п. 4), получаем, что A — это X_5:

	G	D	X_3	F	A	C	E
G		20		17			
D	20			33		45	
X_3					8	70	5
F	17	33				11	22
A			8			25	
C		45	70	11	25		30
E			5	22		30	

8. Очевидно, что оставшееся обозначение X_3 соответствует пункту B:

	G	D	B	F	A	C	E
G		20		17			
D	20			33		45	
B					8	70	5
F	17	33				11	22
A			8			25	
C		45	70	11	25		30
E			5	22		30	

9. Итак, таблица полностью заполнена в соответствии с обозначениями пунктов на графе. Теперь по ней легко определить, что искомая длина пути (ребра) CD равна 45.

Ответ: 45.

Возможные модификации этой задачи:

- после определения соответствия букв городов на графе переменным в таблице определить длину кратчайшего пути из одного указанного пункта в другой;

- после определения соответствия букв городов на графе переменным в таблице найти и записать в качестве ответа букву города, в который ведёт самый короткий путь из указанного в условии пункта.

При этом для определения кратчайшего пути необходимо:

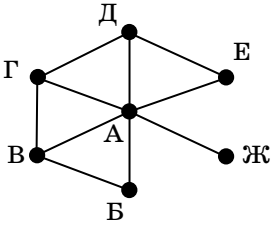
1) из таблицы определить длину каждого пути между пунктами и записать на графе соответствующее значение рядом с каждым ребром;

2) выписать **все** возможные пути между указанными в условии задачи вершинами графа и вычислить соответствующие им длины как суммы длин путей между промежуточными вершинами;

3) записать в качестве ответа наименьшее из полученных значений длины пути.

Задача 3. На рисунке схема дорог изображена в виде графа, а в таблице звёздочками отмечено наличие этих дорог (звёздочка означает, что между указанными двумя населёнными пунктами имеется дорога).

	П1	П2	П3	П4	П5	П6	П7
П1					*		*
П2			*		*		
П3		*			*	*	
П4					*		
П5	*	*	*	*		*	*
П6			*		*		*
П7	*				*	*	



Нумерация пунктов в таблице не связана с буквенными обозначениями этих пунктов на графе. Требуется определить, под какими номерами в таблице обозначены пункты В и Д. В качестве ответа записывается без пробелов и разделителей пара целых чисел — номеров

указанных пунктов по возрастанию этих номеров. Например, если искомые пункты имели бы номера 5 и 2, то в качестве ответа следовало бы записать число 25.

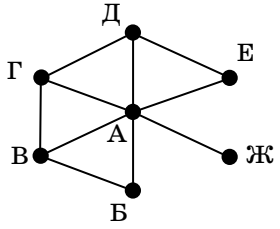
Решение

Основное отличие от задач на графы, предлагавшихся ранее, — в том, что в таблице не даны значения длины дорог, а отмечен только факт их наличия. Но ведь и раньше мы выполняли сопоставление графа и таблицы для определения, каким пунктам в таблице соответствуют какие вершины графа, исключительно по факту наличия в таблице непустых ячеек. А значения длин нужны были только для определения кратчайшего пути. Сейчас же речь идёт только о первой части решения — сопоставить номера пунктов буквенным обозначениям на графе, так что значения длин дорог нам и не нужны.

Решение же задачи — аналогично предыдущим: нужно выявлять на графе наиболее «характерные» пункты (по числу связей) и искать их в таблице.

1) На графе пункт А — единственный, который связан со **всеми** другими пунктами, а пункт Ж — единственный, который связан только с одним пунктом А. Поэтому найти в таблице их очень легко:

	П1	П2	П3	П4 Ж	П5 А	П6	П7
П1					*		*
П2			*		*		
П3		*			*	*	
П4 Ж					*		
П5 А	*	*	*	*		*	*
П6			*		*		*
П7	*				*	*	



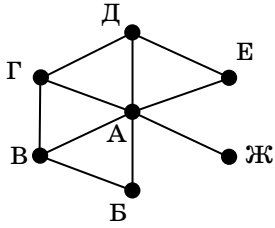
2) Далее нас поджидает одна хитрость.

На графе есть два пункта, которые связаны с другими только двумя путями — Б и Е. Можно сделать вывод, что задача имеет неоднозначное решение: мы не можем точно сказать, какой номер в таблице будет соответствовать пункту Б, а какой пункту Е.

Но если посмотреть на граф в целом и на то, что нас спрашивают в задаче, то можно заметить: граф симметричен (относительно «оси» Г–Ж), а в качестве ответа нужны номера двух симметрично расположенных пунктов В и Д. Поэтому нам безразлично какой номер в таблице мы выберем для пункта Б, а какой для Е: в итоге для искомых пунктов В и Д всё равно получится одна и та же пара номеров, которые надо будет записать по возрастанию.

По таблице мы видим, что с двумя «соседями» связаны пункты П1 и П2. Пусть у нас, скажем, П1 соответствует Б, а П2 — Е:

	П1 Б	П2 Е	П3	П4 Ж	П5 А	П6	П7
П1 Б					*		*
П2 Е			*		*		
П3		*			*	*	
П4 Ж					*		
П5 А	*	*	*	*		*	*
П6			*		*		*
П7	*				*	*	



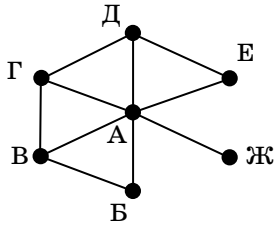
3) Пункты В, Г и Д — это оставшиеся, имеющие трёх «соседей» каждый. При этом пункт В связан с А, Б и Г, а пункт Д — с А, Е и Г.

Вполне очевидно, что пункт П3 в таблице, имеющий связь с пунктом Е (П2), — это Д, а пункт П7, имеющий

связь с пунктом П1 (Б) — это В. Тогда оставшийся пункт П6 — это, по методу исключения, пункт Г.

Окончательно заполняем нашу таблицу:

	П1 Б	П2 Е	П3 Д	П4 Ж	П5 А	П6 Г	П7 В
П1 Б					*		*
П2 Е			*		*		
П3 Д		*			*	*	
П4 Ж					*		
П5 А	*	*	*	*		*	*
П6 Г			*		*		*
П7 В	*				*	*	



По этой таблице определяем номера пунктов В и Д — это номера 7 и 3. Записав их так, как требуется в условии задачи, получаем число 37. Это и есть ответ.

(Если бы мы вначале выбрали в качестве пункта Б пункт П2, а в качестве Е — пункт П1, то получили бы для В номер 3, а для Д — номер 7, и ответ, очевидно, был бы тот же самый — 37.)

Ответ: 37.

Раздел 3. **Системы счисления**

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, бóльшим 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 удаётся 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 развёрнутой записи исходного числа.

Примеры:

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

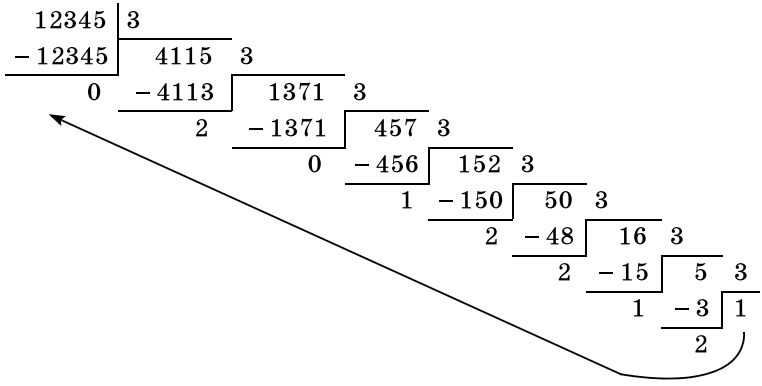
$101110{,}101_2 = 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 1\cdot32 + 0\cdot16 + 1\cdot8 + 1\cdot4 + 1\cdot2 + 0\cdot1 + 1\cdot0{,}5 + 0\cdot0{,}25 + 1\cdot0{,}125 = 32 + 8 + 4 + 2 + 0{,}5 + 0{,}125 = 46{,}625_{10}$;

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

$\mathrm{F4D9{,}7}_{16} = \mathrm{F}\cdot16^3 + 4\cdot16^2 + \mathrm{D}\cdot16^1 + 9\cdot16^0 + 7\cdot16^{-1} = (15)\cdot16^3 + 4\cdot16^2 + (13)\cdot16^1 + 9\cdot16^0 + 7\cdot16^{-1} = 15\cdot4096 + 4\cdot256 + 13\cdot16 + 9\cdot1 + 7\cdot0{,}0625 = 61440 + 1024 + 208 + 9 + 0{,}4375 = 62681{,}4375.$

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 приближённым (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:

$$1011010$$
$$\Downarrow$$
$$001\ 011\ 010$$
$$\Downarrow$$
$$1\ 3\ 2$$

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 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

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
--------
 101110
```

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       ← для 2-го разряда (нулевого) копия
+ 110101           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 $1001011_2 / 101_2$:

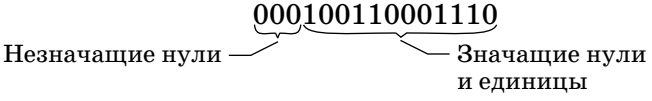
$$\begin{array}{r|l} 1001011 & 101 \\ -\,1001011 & 1111 \\ \hline 0 & \end{array}$$

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 и сколько значащих нулей содержится в значении выражения:

$2^{12} + 2^{10} - 2^8 - 2^6 + 6$?

Решение (*способ 1*)

Для удобства объяснений обозначим все слагаемые (и вычитаемые) буквами по порядку слева направо:

$$\begin{array}{ccccccccc} A & & B & & C & & D & & E \\ 2^{12} & + & 2^{10} & - & 2^8 & - & 2^6 & + & 6 \end{array}$$

1. Длина всего получаемого двоичного числа определяется степенью наибольшего по величине слагаемого A. Поскольку эта степень равна 12, получаем двоич-

ное число из 13 разрядов, причём самый старший из них равен 1. Изобразим это число, пронумеровав его разряды.

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0

 Первое слагаемое всегда (независимо от значения показателя степени) даёт одну двоичную единицу (одну старшую цифру в выбранной системе счисления). Само значение этого показателя степени важно только для определения общей длины получаемого числа, если требуется найти количество нулей.

2. Рассмотрим пару слагаемых B и C. Первое из них, равное $+2^{10}$, добавляет одну единицу в разряде, номер которого равен показателю степени (10):

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	0	0	0	0	0

Второе слагаемое — отрицательное. Оно означает вычитание двоичной единицы из разряда, номер которого равен значению показателя степени (8), причём в этом разряде находится нуль.

Вспомним, как выполняется вычитание в двоичной системе счисления единицы из нуля: в данном разряде остаётся единица, но выполняется заём единицы из более старшего разряда. Если и там записан нуль, то ситуация повторяется — в этом разряде остаётся единица и выполняется заём единицы из ещё более старшего разряда. И таким способом единицы распространяются по числу справа налево, пока не встретится разряд с единицей: она будет взята в качестве заёма, этот очередной разряд обнулится, а операция вычитания будет завершена.

В нашем случае вычитание 2^8 означает, что в 8-м разряде появится единица, затем единица появится и в разряде 9, а поскольку в разряде 10 уже имеется единица, этот 10-й разряд будет обнулён:

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	0	0	0	0

Таким образом, пара слагаемых $2^{10}–2^8$ в итоге дала нам **две** единицы. Можно сделать вывод: часть выражения, соответствующая **разности** двух степеней двойки **даёт количество единиц, равное разности значений этих степеней.**

3. Рассмотрим слагаемое (вернее, вычитаемое) *D*. Такое вычитание производится аналогично, при этом формируется цепочка единиц, идущая от разряда, номер которого равен показателю степени нашего вычитаемого, до предыдущей добавленной единицы. А где она добавлена? Смотрим на предыдущее слагаемое/вычитаемое *C*: его значение показателя степени равно 8, значит, интересующая нас единица — в 8-м разряде. Сколько единиц добавилось? Как и раньше, их количество равно разности значений показателей степеней: 8 − 6 = 2. Но при этом одна единица, бывшая в разряде 8, обнулилась! То есть в итоге добавилась всего одна единица.

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	0	0	0	0	0	0

Вывод: если в выражении встречается **второе вычитание подряд**, то оно **добавляет количество единиц, равное разности значений показателей предыдущей и данной степеней двойки, уменьшенного на 1.**

4. Оставшееся значение *E*, равное 6, уже нетрудно перевести в двоичную систему счисления ($6 = 110_2$) и прибавить к числу, переписав его в младшие разряды.

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	0	0	0	1	1	0

Это слагаемое добавило нам две единицы.

Итого в числе получилось 6 единиц.

5. Количество значащих нулей уже нетрудно подсчитать, вычтя найденное количество единиц из количества разрядов всего числа: 13 – 6 = 7.

Итого в числе имеется 7 нулей.

Ответ: 6 единиц, 7 нулей.

Решение (*способ 2*)

Упростим решение, чтобы не выделять в исходном выражении какие-либо пары слагаемых, а оперировать только со значениями их показателей степеней и знаками сложения/вычитания.

1. Для степеней двойки операция сложения **всегда** добавляет 1 единицу. Будем также считать, что операция вычитания **всегда** добавляет количество единиц, равное разности значений показателей степени данного вычитаемого и предыдущего слагаемого/вычитаемого, уменьшенной на 1 (см. первый способ решения, п. 3). Исключение составляет последнее слагаемое, в котором количество единиц может быть просто подсчитано.

2. Составим таблицу, в которой будем согласно вышеуказанным правилам подсчитывать, сколько единиц добавляет каждое слагаемое или вычитаемое:

Слагаемые	**Степени и знаки**	**Сколько добавляется единиц**
2^{12}	Степень = 12, знак «+»	1
$+ 2^{10}$	Степень = 10, знак «+»	1
$- 2^{8}$	Степень = 8, знак «–»	(10 – 8) – 1 = 1
$- 2^{6}$	Степень = 6, знак «–»	(8 – 6) – 1 = 1
+ 6	Знак «+»	2

3. Суммируем количества добавляемых единиц, указанные в последнем столбце таблицы:

1 + 1 + 1 + 1 + 2 = 6.

4. Количество значащих нулей подсчитывается точно так же, как в первом способе, зная, что всё число содержит 13 разрядов: $13 - 6 = 7$.

Ответ: 6 единиц, 7 нулей.

Задача 2. Сколько единиц и сколько значащих нулей содержится в двоичной записи числа $8^{1341} - 4^{1342} + 2^{1343} - 1344$?

Решение

1. Преобразуем все «степенные» слагаемые в степени двойки:

$8^{1341} - 4^{1342} + 2^{1343} - 1344 = (2^3)^{1341} - (2^2)^{1342} + 2^{1343} - 1344 = 2^{4023} - 2^{2684} + 2^{1343} - 1344$.

Последнее слагаемое переведём в двоичную систему счисления: $1344 = 10101000000_2$ и представим его как сумму степеней двоек: $1344 = 2^{10} + 2^8 + 2^6$. Перепишем исходное выражение в виде:

$2^{4023} - 2^{2684} + 2^{1343} - 2^{10} - 2^8 - 2^6$.

2. Составляем таблицу:

Слагаемые	Степени и знаки	Сколько добавляется единиц	Сколько единиц в числе на данный момент
2^{4023}	Степень = 4023, знак «+»	1	1
-2^{2684}	Степень = 2684, знак «–»	(4023 – 2684) – 1 = 1338	1338
$+2^{1343}$	Степень = 1343, знак «+»	1	1339
-2^{10}	Степень = 10, знак «–»	(1343 – 10) – 1 = 1332	2671
-2^8	Степень = 8, знак «–»	(10 – 8) – 1 = 1	2672
-2^6	Степень = 6, знак «–»	(8 – 6) – 1 = 1	2673

Получаем, что в заданном числе 2673 единицы.

3. Вычисляем количество нулей.

Количество разрядов в числе равно значению наибольшего показателя степени (4023), увеличенному на 1 (нулевой разряд). Следовательно, число имеет 4024 разряда.

Тогда если в нём 2673 единицы, то количество нулей равно $4024 - 2673 = 1351$.

Ответ: 2673 единицы, 1351 нуля.

Задача 3. Сколько единиц в двоичной записи числа: $8^{1000} + 4^{500} - 2^{250} + 3$?

Решение

1. Все числа, кроме последнего, представим как степени двоек (учитывая, что $8 = 2^3$, а $4 = 2^2$). Последнее число 3 оставляем без изменения — оно имеет знак «плюс» и, как легко определить, содержит две единицы.

Получаем запись: $2^{3000} + 2^{1000} - 2^{250} + 3$.

2. Составляем таблицу подсчёта количеств добавляемых единиц:

Слагаемые	Степени и знаки	Сколько добавляется единиц
2^{3000}	Степень = 3000, знак «+»	1
$+ 2^{1000}$	Степень = 1000, знак «+»	1
$- 2^{250}$	Степень = 250, знак «−»	$(1000 - 250) - 1 = 749$
$+ 3$	Знак «+»	2

3. Суммируем найденные количества единиц:
$1 + 1 + 749 + 2 = 753$.

Ответ: 753.

Задача 4. Сколько значащих нулей в двоичной записи числа $8^{256} + 16^{256} - 2^{128} - 245$?

Решение

В двоичной системе количество значащих нулей легко найти как разность общего количества разрядов в числе и количества единиц в нём, поэтому данная задача сводится к подсчёту количества единиц в числе.

1. Аналогично предыдущей задаче, сначала представим все числа как степени двоек.

В данном случае число 245 имеет знак «минус», и напрямую подсчитывать в нём количество единиц, чтобы добавить к ранее полученному числу, неудобно. Поэтому представим число 245 тоже как набор степеней двоек, например: $245 = 256 - 8 - 2 - 1$. Тогда получаем следующую запись:

$8^{256} + 16^{256} - 2^{128} - 245 = 2^{768} + 2^{1024} - 2^{128} - (2^8 - 2^3 - 2^1 - 2^0) = 2^{768} + 2^{1024} - 2^{128} - 2^8 + 2^3 + 2^1 + 2^0$.

2. Перепишем это выражение **по убыванию значений степеней двоек:**

$2^{768} + 2^{1024} - 2^{128} - 2^8 + 2^3 + 2^1 + 2^0 = 2^{1024} + 2^{768} - 2^{128} - 2^8 + 2^3 + 2^1 + 2^0$.

3. Составляем таблицу подсчёта количеств единиц для каждого слагаемого (вычитаемого):

2^{1024}	Степень = 1024, знак «+»	1
$+ 2^{768}$	Степень = 768, знак «+»	1
$- 2^{128}$	Степень = 128, знак «−»	(768 − 128) − 1 = 639
$- 2^8$	Степень = 8, знак «−»	(128 − 8) − 1 = 119
$+ 2^3$	Степень = 3, знак «+»	1
$+ 2^1$	Степень = 1, знак «+»	1
$+ 2^0$	Степень = 0, знак «+»	1

4. Суммируем вычисленные в таблице количества единиц: $1 + 1 + 639 + 119 + 1 + 1 + 1 = 763$.

5. Общее количество значащих разрядов определяется самой старшей степенью двойки, в нашем случае — числом 2^{1024}. Такое число, как мы знаем, содержит единицу в старшем разряде и 1024 нуля после неё, т.е. общая длина числа равна 1025 знаков. Тогда количество значащих нулей в результирующем двоичном числе равно $1025 - 763 = 262$.

Ответ: 262.

Решение подобных задач для других систем счисления в целом аналогично. При этом обычно в условии фигурируют цифры, получаемые вычитанием «из нуля», так что рассмотренные выше принципы решения вполне применимы и в этих случаях.

Задача 5. Значение арифметического выражения $9^{18} + 3^{54} - 9$ записано в системе счисления с основанием 3. Сколько цифр 2 содержится в этой записи?

Решение

1. Представим все числа как степени троек (т.е. как степени заданного основания системы счисления):

$9^{18} + 3^{54} - 9 = 3^{36} + 3^{54} - 3^2$.

2. Перепишем полученные значения по убыванию степеней: $3^{54} + 3^{36} - 3^2$.

3. Составляем таблицу подсчёта, аналогичную ранее рассмотренным при решении задач на двоичную систему, но теперь учитываем отдельно возникающие в троичной системе единицы (при знаке «плюс») и двойки (при знаке «минус»; в троичной системе цифра 2 является аналогом «девятки» в десятичной). При этом обратим внимание: во время заёмов из более старших разрядов двойки распространяются слева направо до тех пор, пока в очередном более старшем разряде не будет встречена единица; именно эта единица будет потрачена, а количество двоек будет равно просто разности степеней предыдущего и данного слагаемых.

3^{54}	Степень = 54, знак «+»	1 единица
$+ 3^{36}$	Степень = 36, знак «+»	**1 единица, которая уходит на заём**
$- 3^{2}$	Степень = 2, знак «−»	36 − 2 = 34 двойки

В нашей задаче нас интересуют только двойки, поэтому количества единиц мы не учитываем, а количество двоек равно 34.

Ответ: 34.

Задача 6. Значение арифметического выражения $343^{2017} + 49^{500} - 7^{777} + 3$ записано в системе счисления с основанием 7. Сколько шестёрок и сколько значащих нулей содержится в этой записи?

Решение

1. Представим все числа как степени семёрок:

$343^{2017} + 49^{500} - 7^{777} + 3 = 7^{6051} + 7^{1000} - 7^{777} + 3.$

Проверим, что степени записаны в порядке убывания.

2. Составляем таблицу подсчёта, помня, что знак «плюс» добавляет единицы, а знак «минус» добавляет шестёрки («старшие» цифры семеричной системы, аналогичные «девяткам» в десятичной).

7^{6051}	Степень = 6051, знак «+»	1 единица
$+ 7^{1000}$	Степень = 1000, знак «+»	**1 единица, которая уходит на заём**
$- 7^{777}$	Степень = 777, знак «−»	1000 − 777 = 223 шестёрки
+ 3	Знак «+»	1 тройка

3. Подсчитываем количество шестёрок: их 223.

4. Подчитываем количество значащих нулей.

Общее число разрядов числа определяется наибольшим значением показателя степени (на 1 больше этого

значения), тогда в числе содержится $6051 + 1 = 6052$ разряда.

Подсчитываем количество **всех ненулевых цифр**, определяемое как сумма всех цифр, добавляемых каждым слагаемым или вычитаемым, рассмотренным в нашей таблице (помня, что одна из добавившихся было единиц была потрачена на заём и, следовательно, превращена в нуль): $1 + 223 + 1 = 225$.

Тогда количество нулей в числе вычисляется как разность: $6052 - 225 = 5827$.

Ответ: 223 шестёрки и 5827 нулей.

Задача 7. Значение арифметического выражения $64^{100} + 4^{200} - 16$ записано в системе счисления с основанием 8. Сколько в полученной записи будет цифр «7»?

Решение

1. Запишем все слагаемые как степени **двойки** по убыванию значений показателя степени:

$64^{100} + 4^{200} - 16 = 2^{600} + 2^{400} - 2^4$.

2. Составляем таблицу для подсчёта количества **двоичных единиц**:

2^{600}	Степень = 600, знак «+»	1 единица
$+ 2^{400}$	Степень = 400, знак «+»	1 единица
$- 2^4$	Степень = 4, знак «−»	(400 − 4) − 1 = 395 единиц

3. Нас спрашивают, сколько значение этого выражения будет содержать **семёрок**, если его записать в **восьмеричной** системе счисления.

Вспомним правило преобразования двоичного числа в восьмеричное «по триадам»: двоичное число справа налево разбивается на тройки («триады») двоичных цифр, а затем каждая такая триада заменяется на её восьмеричное значение (как трёхзначное двоичное чис-

ло). Тогда можно определить, сколько в восьмеричной записи семёрок, подсчитав в ней количество триад единиц («111»).

В нашем случае отдельная первая единица гарантированно не даёт такую триаду, так что её мы исключаем из рассмотрения. А вот в полученной вычитанием длинной цепочке из 395 единиц (для слагаемых $2^{400}-2^4$) можно выделить такие триады (нужные нам «семёрки»). Их количество будет равно результату целочисленного деления количества единиц в цепочке на 3 (очевидно, что одна или две единицы, соответствующие остатку от такого деления, нужной триады не дают и потому отбрасываются).

Получаем[1]: 395 div 3 = 131.

Ответ: 131.

Задача 8. Значение арифметического выражения: $9^8 + 3^{25} - 14$ записали в системе счисления с основанием 3. Найдите сумму цифр в этой записи. Ответ запишите в десятичной системе.

1. $9^8 + 3^{25} - 14 = 3^{16} + 3^{25} - (3^2 + 3^1 + 2) =$
$= 3^{25} + 3^{16} - 3^2 - 3^1 - 2.$

2. Составляем таблицу:

3^{25}	Степень = 25, знак «+»	1 единица
3^{16}	Степень = 16, знак «+»	**1 единица — истрачена на заём** — получен 0
-3^2	Степень = 2, знак «−»	**(16 − 2) = 14 двоек, одна истрачена на заём** — получено 13 двоек и 1 единица
-3^1	Степень = 1, знак «−»	**1 двойка, истрачена на заём,** — получена 1 единица
-2		1 единица

3. Сумма цифр записи: $1 + 0 + 13 \cdot 2 + 1 + 1 + 1 = 30$.

Ответ: 30.

[1] div — операция целочисленного деления.

Задачи на кодирование, решаемые с применением недесятичных систем счисления

Конспект

Система счисления — знаковая система, позволяющая по определённым 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 латинских 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.

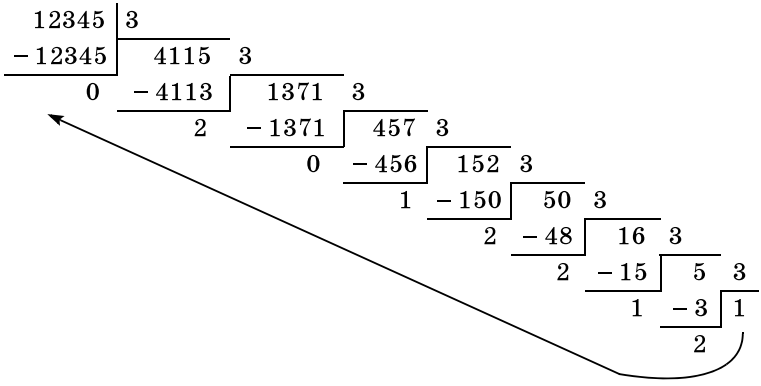
Перевод числа из троичной системы счисления в десятичную осуществляется путём выполнения вычислений по развёрнутой записи исходного числа.

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, 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: $12345_{10} = 121221020_3$.

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:

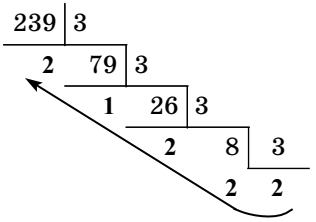
- последовательность чисел начинается с значения $00000_3 = 0_{10}$;
- нумерация чисел в списке начинается с единицы.

Тогда для определения числа, стоящего на 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. Все трёхбуквенные слова, составленные из букв К, О, Д, Е, Р, записаны в алфавитном порядке и пронумерованы, начиная с 1. Начало списка выглядит так:

1. ООО
2. ООК
3. ООД
4. ООР
5. ООЕ
6. ОКО

...

Под каким номером в списке идёт первое слово, которое начинается с буквы Д?

Решение

1. Очевидно, что приведённый в условии список слов идентичен трёхзначным числам в пятеричной си-

стеме, где буквы соответствуют цифрам: О — 0, К — 1, Д — 2, Р — 3, Е — 4. Тогда нужно искать первое трёхзначное число, начинающееся с цифры 2, т.е. число 200_5 или, в десятичном представлении, 50.

2. Заметим, что в списке число 000 идёт под номером 1, — значит, номер каждого числа на 1 больше самого этого числа. Тогда искомое число (а значит, и слово) в списке стоит под номером 51.

Ответ: 51.

Задача 4. Сколько существует различных символьных последовательностей длины 5 в трёхбуквенном алфавите {К, О, Т}, которые содержат ровно две буквы О?

Решение (*способ 1*)

Задача подобна предыдущим, и принцип её решения будет во многом аналогичным — путём «перевода» букв в цифры соответствующей системы счисления и работы с полученными числами.

1) Поскольку никаких слов изначально не задано, мы можем сами «назначить» цифры, соответствующие каждой букве. А поскольку букв в алфавите три, мы имеем дело с троичной системой счисления и выберем следующие соответствия букв и цифр:

О — 0, К — 1, Т — 2.

(Мы выбрали нуль именно для буквы О, так как нам понадобится искать слова, содержащие ровно две буквы О, а с нулями это окажется проще.)

2) Итак, речь идёт об определении количества пятиразрядных троичных чисел (причём незначащие нули слева важны!), содержащих ровно два нуля.

3) Первый нуль может быть в одной из 5 позиций — получаем 5 вариантов.

В каждом из этих вариантов второй нуль может располагаться так:

- когда первый нуль находится в позиции «1», второй нуль может располагаться в одной из четырёх оставшихся позиций — «2», «3», «4», «5»;

- когда первый нуль находится в позиции «2», второй нуль может располагаться в одной из 3 оставшихся позиций правее — «3», «4» и «5» (ведь ситуацию, когда нули располагаются в позициях «1» и «2», мы уже рассмотрели перед этим);
- когда первый нуль находится в позиции «3», второй нуль может располагаться в одной из 2 оставшихся позиций правее — «4» и «5» (почему — рассмотрено выше);
- когда первый нуль находится в позиции «4», второй нуль может располагаться только в 1 позиции правее — «5».

Итого получаем 4 + 3 + 2 + 1 = 10 вариантов размещения в пятиразрядном числе двух нулей.

 Возможны и другие рассуждения, приводящие к тому же результату.

Условно обозначим наши нули разными цветами: например, первый — синим, а второй — красным. Тогда синий нуль можно разместить в пяти разрядах — получаем 5 вариантов. И в каждом из этих пяти вариантов красный нуль можно разместить в любой из оставшихся четырёх позиций — т.е. по четырём вариантам. Значит, всего получаем $5 \cdot 4 = 20$ вариантов размещения наших разноцветных нулей. Но теперь вспомним, что на самом деле оба нуля (синий и красный) — совершенно равноправны. А значит, пары «синий—красный» и «красный—синий» (если читать число слева направо) — это одни и те же пары. Следовательно, каждая такая пара посчитана дважды, и всего вариантов размещения в пятиразрядном числе двух одинаковых нулей будет 10.

4) Итак, существует 10 вариантов размещения в числе двух нулей. В каждом из этих 10 вариантов остаётся три цифры, которые могут быть равны или 1, или 2.

Сколько может быть таких неповторяющихся комбинаций? Очевидно, столько, сколько может быть различных трёхразрядных чисел в системе счисления, состоящей из двух цифр (т.е. двоичной). Значит, для

каждого из 10 ранее найденных вариантов получается $2^3 = 8$ «подвариантов».

 Количество различных n-разрядных чисел в системе счисления с основанием m определяется по формуле: m^n (основание системы счисления, возведённое в степень, равную числу разрядов).

Тогда общее число троичных чисел, в которых из пяти цифр ровно две — нулевые, будет равно $8 \cdot 10 = 80$.

Решение (*способ 2*)

Другой возможный способ решения — использование формул комбинаторики.

1) В последовательности из 5 букв должно быть две буквы О и соответственно три другие буквы. Сначала ищем количество **перестановок с повторениями** из двух букв О и трёх произвольных букв (обозначим их символом «#»).

 Формула для вычисления количества перестановок с повторениями:

$$P = \frac{n_\Sigma!}{n_1! n_2!},$$

где n_S — общее количество букв в слове, n_1 — количество обязательных букв (в данном случае — букв О), n_2 — количество прочих букв (т.е. знаков #). Знак «!» обозначает вычисление факториала (произведение всех натуральных чисел от 1 до n: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$.

В данной задаче количество перестановок с повторениями будет равно:

$$\mathrm{P} = \frac{5!}{2!3!} = \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{1 \cdot 2 \times 1 \cdot 2 \cdot 3} = \frac{4 \cdot 5}{2} = 10.$$

2) В остальных трёх позициях слова (кроме тех, что заняты двумя буквами О) может стоять любая из двух оставшихся букв — К или Т. Следовательно, нужно

вычислить количество всех возможных трёхбуквенных слов, состоящих из двух возможных букв.

Такое трёхбуквенное слово можно рассматривать как трёхразрядное число, которое представлено в системе счисления с основанием 2 (так как в алфавите этой «системы счисления» всего две «цифры» — К и Т). Тогда количество таких слов совпадает с количеством возможных различных трёхразрядных двоичных чисел. А оно равно $2^3 = 8$.

 Количество различных n-разрядных чисел в системе счисления с основанием m определяется по формуле: m^n (основание системы счисления, возведённое в степень, равную числу разрядов).

Ответ: 80.

Задача 5. Составляется таблица кодовых слов для передачи сообщений, где каждому сообщению должно быть сопоставлено отдельное кодовое слово из 4 букв. В этих кодовых словах можно использовать только пять букв: «Л», «О», «Г», «И», «К», при этом буква «Л» может быть использована ровно один раз, а все остальные буквы могут встречаться в кодовом слове сколько угодно раз (или отсутствовать).

Сколько различных кодовых слов можно получить по этой таблице?

Решение

1. Выпишем все возможные варианты четырёхбуквенных кодовых слов с ровно одной буквой «Л»: Л***, *Л**, **Л* или ***Л (где * обозначает любые другие буквы). Получили 4 возможных варианта.

2. Вариантов распределения четырёх оставшихся в нашем алфавите букв в трёх позициях кодовых слов столько же, сколько возможно трёхзначных чисел в системе счисления с основанием 4, т.е. $4^3 = 64$.

3. Тогда всего возможных кодовых слов: $4 \times 64 = 256$.

Это решение можно записать более наглядно в виде схемы:

Варианты		Кол-во вариантов в позициях		Всего
Л***	$\Rightarrow$	$1 \cdot 4 \cdot 4 \cdot 4$	=	64
*Л**	$\Rightarrow$	$4 \cdot 1 \cdot 4 \cdot 4$	=	64
**Л*	$\Rightarrow$	$4 \cdot 4 \cdot 1 \cdot 4$	=	64
***Л	$\Rightarrow$	$4 \cdot 4 \cdot 4 \cdot 1$	=	64
		ИТОГО:		$\mathbf{4 \times 64 = 256}$

Ответ: 256.

Задача 6. Составляется таблица кодовых слов из 5 букв, в которых можно использовать только буквы: «К», «А», «Т», «Е», «Р». При этом буква «Р» должна быть использована хотя бы 2 раза, а остальные буквы могут встречаться сколько угодно раз или отсутствовать. Сколько можно получить различных кодовых слов?

Решение

В отличие от предыдущей задачи, буква «Р» должна встречаться не *ровно* какое-то количество раз, а ***хотя бы*** два раза. Это означает, что в каждом получаемом слове мы должны обеспечить две буквы «Р» обязательные, а в остальных позициях слов могут быть любые буквы, в том числе и та же буква «Р».

По этой причине вычислять количество вариантов в каждой позиции, обозначенной звёздочкой, в данном случае чуть сложнее, чем в предыдущей задаче. Если, например, мы рассматриваем два варианта — «РР*» и «*РР», то нетрудно увидеть: дополнительную букву «Р» вместо звёздочки можно использовать только один раз, иначе мы дважды посчитаем одно и то же слово. Чтобы избежать такого дублирования, вместо звёздочек, расположенных справа от обозначенных букв, мы будем подставлять *все* возможные буквы из числа заданных, а вместо звёздочек, расположенных слева, — на одну букву меньше.

В данной задаче, например, в варианте «*Р*Р*» звёздочка справа (после обеих букв «Р») будет заменяться на 5 разных букв, а звёздочки левее любой из букв «Р» — только на 4 разные буквы (буква «Р» тут исключается).

Запишем решение в виде схемы. При этом варианты в первой колонке таблицы формируются так: первая буква «Р» сначала стоит в первой позиции, а вторая буква «Р» поочередно «пробегает» по всем позициям правее, затем первая «Р» ставится во вторую позицию, а вторая «Р» опять «пробегает» все оставшиеся позиции вправо от первой, и т.д.

Варианты		**Кол-во вариантов в позициях**		**Всего**
РР***	$\Rightarrow$	$1 \cdot 1 \cdot 5 \cdot 5 \cdot 5$	$=$	$5^3 = 125$
Р*Р**	$\Rightarrow$	$1 \cdot 4 \cdot 1 \cdot 5 \cdot 5$	$=$	$4 \cdot 5^2 = 100$
Р**Р*	$\Rightarrow$	$1 \cdot 4 \cdot 4 \cdot 1 \cdot 5$	$=$	$4^2 \cdot 5 = 80$
Р***Р	$\Rightarrow$	$1 \cdot 4 \cdot 4 \cdot 4 \cdot 1$	$=$	$4^3 = 64$
*РР**	$\Rightarrow$	$4 \cdot 1 \cdot 1 \cdot 5 \cdot 5$	$=$	$4 \cdot 5^2 = 100$
*Р*Р*	$\Rightarrow$	$4 \cdot 1 \cdot 4 \cdot 1 \cdot 5$	$=$	$4^2 \cdot 5 = 80$
*Р**Р	$\Rightarrow$	$4 \cdot 1 \cdot 4 \cdot 4 \cdot 1$	$=$	$4^3 = 64$
**РР*	$\Rightarrow$	$4 \cdot 4 \cdot 1 \cdot 1 \cdot 5$	$=$	$4^2 \cdot 5 = 80$
**Р*Р	$\Rightarrow$	$4 \cdot 4 \cdot 1 \cdot 4 \cdot 1$	$=$	$4^3 = 64$
***РР	$\Rightarrow$	$4 \cdot 4 \cdot 4 \cdot 1 \cdot 1$	$=$	$4^3 = 64$
		ИТОГО:		**$125 + 2\times100 + 3\times80 + 4\times64 = 821$**

Ответ: 821.

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»:

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

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 таблицы

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 Си) 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:* $\neg((X > 2) \rightarrow \rightarrow (X > 3))$» 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. Логическая функция F задана выражением $(a \vee b) \to (b \equiv c)$. Имеется частично заполненный фрагмент таблицы истинности функции F, содержащий **неповторяющиеся** строки.

Требуется определить соответствие каждой из переменных (a, b, c) столбцам таблицы истинности. В ответе нужно без пробелов и разделителей записать буквы a, b и c в том порядке, в каком в таблице следуют соответствующие им столбцы.

Переменная 1	Переменная 2	Переменная 3	Функция
???	???	???	F
0		0	0
		0	0

Подобные задачи уже встречались на ЕГЭ ранее. Усложнение здесь — в том, что в ячейках таблицы имеется слишком мало исходных данных, и это существенно усложняет рассуждения.

Решение 1 (метод рассуждений)

Рассмотрим заданную функцию: $(a \vee b) \to (b \equiv c)$.

1. Согласно таблице, в обоих случаях значение F равно 0. А операция следования $(\to)$ даёт нуль только в одном единственном случае: $1 \to 0$. Следовательно, для обеих строк таблицы должно обязательно соблюдаться условие: $(a \vee b) = 1$ И $(b \equiv c) = 0$.

2. Начнём анализ таблицы со столбца, в котором обе ячейки заполнены нулями («*Переменная 3*»).

Предположим, что переменная b, которая в обоих случаях равна нулю, — это «*Переменная 3*». Тогда из условия $(b \equiv c) = 0$ получаем, что переменная b может быть только ненулевой (нам нужно, чтобы тождество было ложным, а значит, переменные b и c должны быть

разными). Но если $b = 0$, то из условия $(a \vee b) = 1$ переменная a должна быть равна только единице. Делаем вывод: если в какой-то строке таблицы $b = 0$, то в этой строке **не должно быть** больше никаких других нулей! А первая строка таблицы этому противоречит. Следовательно, b не может быть переменной «*Переменная 3*».

Предположив, что b — это «*Переменная 1*», и рассуждая аналогично, получаем опять-таки противоречие в первой строке таблицы: для нулевого значения b в той же строке ни в какой ячейке нулей не должно быть. Значит, b не может быть и переменной «*Переменная 1*».

Остаётся только одна возможность: b — это «*Переменная 2*»:

Переменная 1	**Переменная 2**	**Переменная 3**	**Функция**
???	*b*	???	*F*
0		0	0
		0	0

3. Остаётся решить, где находится переменная a, а где c.

Опять-таки начинаем анализ с полностью заполненного нулями столбца.

Предположим, что «*Переменная 3*» — это a. Тогда:

- из условия $(a \vee b) = 1$ при **обоих** нулевых значениях a получаем, что $b = 1$;
- тогда из условия $(b \equiv c) = 0$ для $b = 1$ в **обеих** строках таблицы c должна быть равна 0.

Но тогда таблица примет вид:

Переменная 1	**Переменная 2**	**Переменная 3**	**Функция**
c	*b*	*a*	*F*
0	1	0	0
0	1	0	0

В этом случае у нас получаются две одинаковые строки, тогда как по условию эти строки обязаны быть неповторяющимися. Вывод: наше предположение неверно, тогда остаётся только один-единственный вариант: «*Переменная 3*» — это c, тогда «*Переменная 1*» — это a. Таблица в этом случае имеет вид:

Переменная 1	**Переменная 2**	**Переменная 3**	**Функция**
a	b	c	F
0		0	0
		0	0

Ответ: abc.

Примечание. Если бы мы начали с предположения, что «*Переменная 3*» — это c, то получили бы, что из условия $(b \equiv c) = 0$ для обоих нулевых значений c обязательно $b = 1$, и соответственно, из условия $(a \vee b) = 1$ нам допустимо любое значение a, т.е. сделанное предположение вполне может быть правильным. Но тогда обязательно надо было бы проверить и вторую возможность — предположение, что «*Переменная 3*» — это a, поскольку иначе анализ был бы неполным. И мы бы в этом случае получили, что данное предположение неверно, а следовательно, правильно наше первое предположение, что «*Переменная 3*» — это c.

Решение 2

(*построение полной таблицы истинности*)

Если приведённые выше логические рассуждения покажутся слишком сложными, то можно «пойти в лобовую атаку» и построить полную таблицу истинности, благо переменных здесь всего три, а не пять или шесть. (Напомним, что все возможные сочетания значений переменных a, b, c в левой части таблицы — это последовательные двоичные числа от 000 до 111.)

a	b	c	$(a \vee b)$	$(b \equiv c)$	$F = (a \vee b) \to (b \equiv c)$
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	1

Выделим в этой таблице строки, в которых значение функции F нулевое, и посмотрим на соответствующие значения исходных переменных:

a	b	c
0	1	0
1	0	1
1	1	0

Видим, что у нас в этой таблице есть единственная строчка с двумя нулями (первая) — и только один столбец, где есть два нуля (столбец переменной c). Сравнивая эти строку и столбец с заданной в условии «частичной» таблицей, нетрудно заметить, что «*Переменной 3*» может быть только c (поскольку только для c мы получили в соответствующем столбце два нуля). А по сравнению первой строки нашей таблицы со строкой с двумя нулями в исходной таблице получаем, что этот второй нуль для переменной a может быть только «*Переменной 1*». Соответственно то, что «*Переменная 2*» — это b, определяется методом исключения.

 В построенной полной таблице истинности может быть и большее число строк с подходящим значением логического выражения, чем указано в задаче: ведь в условии сказано, что приводится **фрагмент** полной таблицы!

Ответ: abc.

Задача 2. На числовой прямой даны два отрезка: P = [2,35] и Q = [12,54].

Укажите наибольшую возможную длину отрезка A, для которого формула

$$((x \in P) \rightarrow ((x \in Q)) \wedge (x \in P))) \rightarrow \neg(x \in A)$$

тождественно истинна, т.е. принимает значение 1 при любом значении переменной x.

Решение

1. Для упрощения записи заменим высказывания логическими переменными: $a = (x \in A)$, $p = (x \in P)$, $q = (x \in Q)$. Тогда исходное логическое выражение принимает вид:

$(p \rightarrow (q \wedge p)) \rightarrow \bar{a}$.

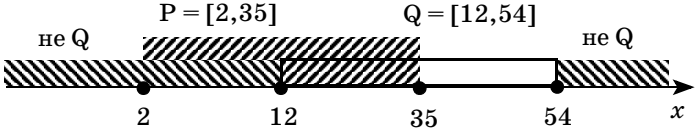
2. Избавимся от операции следования, использовав соотношение $x \rightarrow y = \neg x \vee y$:

$$(p \rightarrow (q \wedge p)) \rightarrow \bar{a} = \overline{(\bar{p} \vee (q \wedge p))} \vee \bar{a} =$$

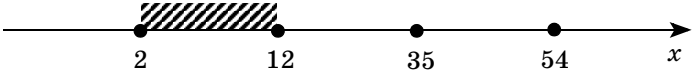
$$= (\bar{p} \wedge \overline{(q \wedge p)}) \vee \bar{a} = (p \wedge (\bar{q} \vee \bar{p})) \vee \bar{a}.$$

3. Выражение $p \wedge (\bar{q} \vee \bar{p})$ можно упростить: $p \wedge (\bar{q} \vee \vee \bar{p}) = (p \wedge \bar{q}) \vee (p \wedge \bar{p})$. Но $(p \wedge \bar{p})$ тождественно равно логическому нулю. Тогда $(p \wedge \bar{q}) \vee (p \wedge \bar{p}) = (p \wedge \bar{q}) \vee \vee 0 = (p \wedge \bar{q})$. В результате получаем выражение $(p \wedge \bar{q}) \vee \vee \bar{a}$, которое должно быть тождественно истинно.

4. Рисуем числовую прямую и изображаем на ней отрезки P, Q и «не Q»:

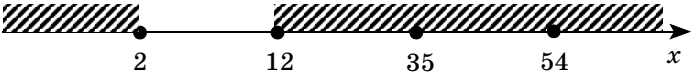


5. Легко найти, какой отрезок соответствует выражению $(p \wedge \bar{q})$:

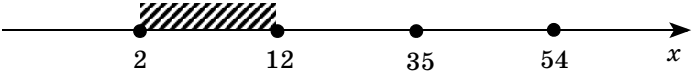


6. Нам нужно, чтобы $(p \wedge \bar{q}) \vee \bar{a}$ было истинно, причём в нём использована логическая операция ИЛИ, а длина отрезка А должна быть максимально возможной. Если $(p \wedge \bar{q})$ уже истинно, то значение $\bar{a}$ нам безразлично. Когда же $(p \wedge \bar{q})$ ложно, истинность всего выражения может обеспечить только истинность $\bar{a}$.

Ложности $(p \wedge \bar{q})$, очевидно, соответствует вся числовая прямая, кроме обозначенного на предыдущем чертеже отрезка [2,12]. Тогда переменной $\bar{a}$ тоже соответствует вся числовая прямая, кроме отрезка [2,12].



Но $\bar{a}$ означает, что $x \notin$ А. Значит, искомый отрезок А — это всё, что *не покрывается* отрезком для $\bar{a}$. Тогда искомый отрезок А будет таким:



7. Если искомый отрезок А соответствует интервалу [2,12], то его длина равна разности $12 - 2 = 10$.

Ответ: 10.

Рассуждения на шаге 6 могут быть и другими.

6. Нам нужно, чтобы $(p \wedge \bar{q}) \vee \bar{a}$ было истинно. Но переменная a здесь стоит под отрицанием. Чтобы от него избавиться, проинвертируем всё выражение:

$$(p \wedge \bar{q}) \vee \bar{a} = 1 \Rightarrow \overline{(p \wedge \bar{q}) \vee \bar{a}} = 0 \Rightarrow \overline{(p \wedge \bar{q})} \wedge a = 0.$$

Когда $\overline{(p \wedge \bar{q})} = 0$, значение выражения уже будет гарантированно ложным и значение a нас не интересует. Если же $\overline{(p \wedge \bar{q})} = 1$, то ложность выражения обеспечивается только ложностью a.

$\overline{(p \wedge \bar{q})} = 1$, когда $(p \wedge \bar{q}) = 0$, т.е. на всей числовой прямой, кроме отрезка [2,12]. Значит, на всем интервале $(-\infty,2)$ и $(12,\infty)$ значение a должно быть ложно. Следовательно, искомого отрезка А на интервале $(-\infty,2)$ и $(12,\infty)$ **быть не**

должно. Значит, отрезок A соответствует все тому же интервалу [2,12].

Далее длина найденного отрезка вычисляется аналогично уже рассмотренному шагу 7.

Задача 3. На числовой прямой даны три отрезка: P = [10,30], Q = [15,30] и R = [20,35].

Укажите такой отрезок A, для которого выражения

$$(x \notin \mathrm{A}) \to (x \notin \mathrm{P})\ ) \text{ и } (x \in \mathrm{Q}) \to (x \notin \mathrm{R})$$

тождественно равны, т.е. оба истинны или оба ложны при любом значении переменной x.

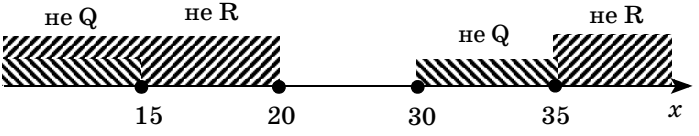
Решение

1. Аналогично предыдущей задаче, вводим переменные $a = (x \in \mathrm{A})$, $p = (x \in \mathrm{P})$, $q = (x \in \mathrm{Q})$, $r = (x \in \mathrm{R})$, переписываем с использованием этих переменных оба выражения и избавляемся в них от операции следования:

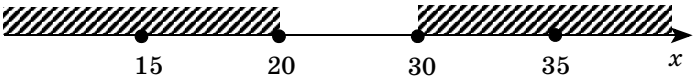
— первое выражение: $\bar{a} \to \bar{p} = \bar{\bar{a}} \vee \bar{p} = a \vee \bar{p}$;

— второе выражение: $q \to \bar{r} = \bar{q} \vee \bar{r}$.

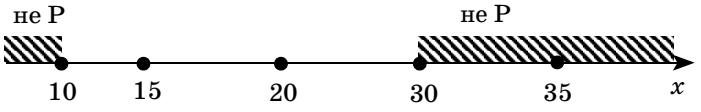
2. Определяем интервалы истинности и ложности второго выражения, для которого оба отрезка заданы в условии:



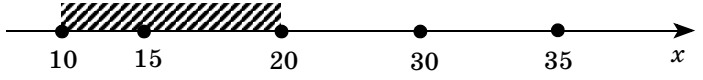
3. Тогда для операции $\bar{q} \vee \bar{r}$ получаем интервал:



4. Изображаем отрезок для значения $\bar{p}$ из первого выражения:



5. Каким должен быть отрезок A, чтобы результат $a \vee \bar{p}$ операции был таким же, как на рисунке для шага 3? Очевидно, нам нужно «закрасить» отрезком A участок [10,20].



Ответ: [10,20].

Задача 4. Элементами множества X являются натуральные числа. Известно, что выражение

$(n \in \{2,3,6,8,10\}) \to (((n \in \{1,3,6,7,9\}) \wedge \neg(n \in X)) \to$
$\to \neg(n \in \{2,3,6,8,10\}))$

истинно (принимает значение 1) при любом значении n. Требуется определить наименьшее возможное значение произведения элементов множества X.

Решение

Начало решения такое же, как в предыдущей задаче: введём для используемых в выражении множеств обозначения в виде латинских букв R и Q, а затем заменим записи условий принадлежности переменной n соответствующему множеству одноимёнными логическими переменными:

- множества: $Q = \{2,3,6,8,10\}$, $R = \{1,3,6,7,9\}$;
- логические переменные: $Q = n \in \{2,3,6,8,10\}$, $R = n \in \{1,3,6,7,9\}$, $X = (n \in X)$.

Затем переписываем заданное логическое выражение с учётом этих замен и выполняем его упрощение, избавляясь от операции следования и (по возможности) от скобок:

$(n \in \{2,3,6,8,10\}) \to (((n \in \{1,3,6,7,9\}) \wedge \neg(n \in X)) \to$
$\to \neg(n \in \{2,3,6,8,10\})) \Rightarrow Q \to ((R \wedge \neg X) \to \neg Q)$.

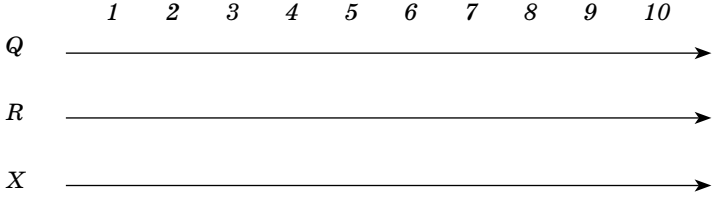
Удобнее для обозначения операции отрицания использовать надчёркивание переменных, поэтому перепишем выражение в виде $Q \to ((R \wedge \overline{X}) \to \overline{Q})$.

Упрощаем это выражение, пользуясь правилом замены операции следования на операцию ИЛИ ($A \to B = \overline{A} \vee B$):

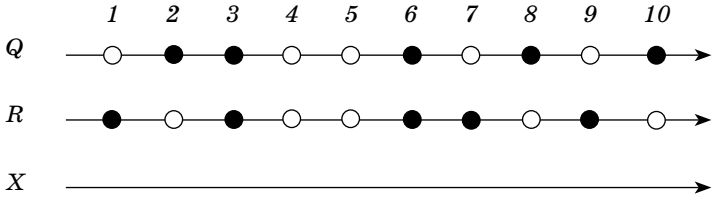
$$Q \to ((R \wedge \overline{X}) \to \neg\overline{Q}) = \overline{Q} \vee (\overline{(R \wedge \overline{X})} \vee \overline{Q}) =$$
$$= \overline{Q} \vee (\overline{R} \vee X \vee \overline{Q}) = \overline{Q} \vee \overline{R} \vee X \vee \overline{Q}) =$$
$$= \overline{Q} \vee \overline{R} \vee X.$$

А теперь перейдём к графической части решения.

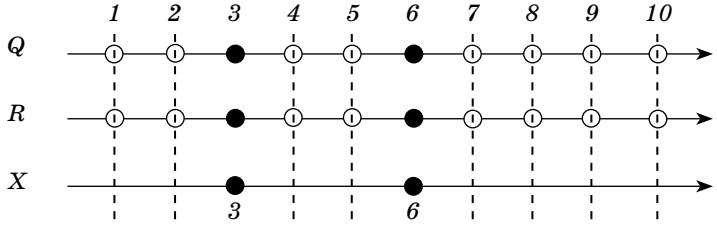
1) Рисуем числовые прямые, соответствующие множествам Q, R и X (лучше всего делать это на тетрадном листке в клеточку). Размечаем на них позиции, соответствующие всем возможным числам, используемым в множествах (в данном случае — от 1 до 10, так как наименьшее число, встречающееся в заданных множествах, равно 1, а наибольшее равно 10).



2) На числовой прямой, соответствующей множеству Q, размечаем точки по следующему правилу: для числа, которое *есть* в этом множестве, точка должна быть *чёрной* (закрашенной), а для числа, которого *нет* в этом множестве, точка должна быть белой (в виде пустого кружочка). Затем то же самое делаем и для числовой прямой, соответствующей множеству R.



3) Смотрим на получившееся у нас логическое выражение: $\overline{Q} \vee \overline{R} \vee X$. В нём заданные нам (в виде условий принадлежности чисел соответствующим множествам) значения $\overline{Q}$ и $\overline{R}$ связаны друг с другом и с переменной X логической операцией ИЛИ. Для гарантированной истинности этого выражения достаточно, чтобы *хотя бы одно* из значений — $\overline{Q}$ или $\overline{R}$ — было истинно. В нашем графическом методе действует правило: *для переменной без надчёркивания* (т.е. без отрицания) «истинными» являются *чёрные* точки, а *для переменной с надчёркиванием* (т.е. под отрицанием) «истинными» являются *белые* точки. У нас переменные Q и R записаны с надчёркиванием. Поэтому на нашем рисунке мы вычёркиваем вертикальными пунктирными линиями все такие числовые позиции, где на осях Q и R есть *хотя бы одна белая точка*. А в позициях, которые на числовой прямой множества X остались не зачёркнутыми, проставляем «чёрные» точки.



Задача практически решена: найденные значения — числа 3 и 6 — это и есть минимально необходимые элементы множества X, обеспечивающие истинность заданного выражения на всей числовой прямой для натуральных чисел.

Завершить же вычисления — подсчитать количество элементов множества X, их сумму или произведение — уже совсем просто. В нашей задаче требуется произведение — оно равно $3 \cdot 6 = 18$.

Ответ: 18.

Задача 5. P — множество всех 8-битовых цепочек, начинающихся с 101. Q — множество всех 8-битовых цепочек, в которых на предпоследнем месте стоит 1. A — некоторое множество произвольных 8-битовых цепочек. Сколько элементов содержит минимальное множество A, при котором для любой 8-битовой цепочки x истинно выражение:

$$\neg(x \in A) \to ((x \in Q) \to (x \in P)).$$

Решение

1. Как и в других задачах с логическими выражениями, сначала вводим логические переменные a, q и p и избавляемся от операции следования:

$$\bar{a} \to (q \to p) = \bar{a} \to (\bar{q} \vee p) = a \vee \bar{q} \vee p.$$

2. Множество P можно представить в виде {101*****}, где звёздочки обозначают знаковые позиции, в которых может быть любое значение бита — как 0, так и 1. Сколько возможно вариантов таких цепочек? Очевидно, столько же, сколько существует различных двоичных пятизначных (по количеству звёздочек) чисел: 2^5 вариантов (32 штуки).

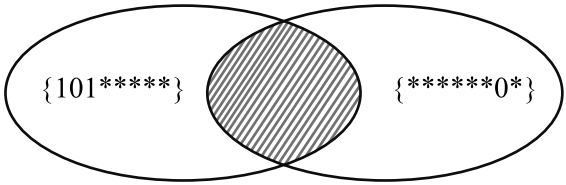
Аналогично, множество Q можно представить в виде {******1*}. Таких цепочек возможно 2^7 вариантов (128 штук).

«Обратное» множество, соответствующее $\bar{q}$, — это множество, которое получается вычитанием только что найденного множества Q = {******1*} из множества всех возможных 8-битовых цепочек.

Последних — столько, сколько возможно 8-битовых двоичных чисел, т.е. $2^8 = 256$ штук.

Тогда множество «не Q» будет содержать 256 – 128 = = 128 различных 8-битовых цепочек и иметь вид {******0*} (жёстко заданные в исходном множестве Q биты инвертируются, а звёздочки остаются такими же звёздочками).

3. Какое множество и какого размера соответствует выражению $\bar{q} \vee p$? Изобразим эту операцию ИЛИ в виде диаграммы Эйлера–Венна:



Если вычислять общее количество цепочек, соответствующих выражению $\bar{q} \vee p$, суммированием количеств цепочек для p и $\bar{q}$, то некоторые цепочки (соответствующие заштрихованной общей области на диаграмме) окажутся посчитаны дважды. Для избежания этого нужно из суммы вычесть количество цепочек, соответствующих заштрихованной области, т.е. выражению $\bar{q} \wedge p$. А ему соответствуют цепочки, в которых жёстко зафиксированы все биты, которые заданы и в первом, и во втором множестве, т.е. цепочки вида {101***0*}. Таких цепочек (по количеству звёздочек) существует $2^4 = 16$ вариантов.

Тогда выражению $\bar{q} \vee p$ соответствует количество цепочек, равное $2^7 + 2^5 - 2^4 = 128 + 32 - 16 = 144$ вариантам.

4. Итак, 144 возможные цепочки соответствуют истинности выражения $\bar{q} \vee p$. Но эти цепочки как раз нас и не интересуют — для них истинность исходного логического выражения уже обеспечена. Нам же важны цепочки, при которых выражение $\bar{q} \vee p$ ложно и истинность исходного выражения обеспечивается как раз истинностью переменной a. Очевидно, что это — все остальные возможные цепочки.

Как уже было подсчитано, всего 8-битовых цепочек возможно $2^8 = 256$ штук. И если из этого количества отбрасываются 144 цепочки, то в множестве A остаётся $256 - 144 = 112$.

Ответ: 112.

Задача 6. Обозначим через & поразрядную конъюнкцию неотрицательных целых чисел. Для какого наименьшего неотрицательного целого числа P формула

$$[A\&P \neq 0] \to ([A\&12 = 0] \to [A\&68 \neq 0]) = 1$$

тождественно истинна (т.е. принимает значение 1 при любом неотрицательном целом значении переменной A)?

Решение

1. Преобразуем операцию следования по правилу: $X \to Y = \overline{X} \vee Y$.

$$[A\&P = 0] \vee ([A\&\mathbf{12}] \neq 0] \vee [A\&\mathbf{68} \neq 0]) = 1.$$

2. Раскладываем оба имеющихся числа на степени двойки:

Число десятичное	12	68
Число двоичное	1100_2	1000100_2
Степени двойки	$8 + 4$	$64 + 4$

3. Полученные степени двойки записываем как множества:

Число	12	68
Множество	{8, 4}	{64, 4}

4. Если для числа стоит **неравенство нулю**, то записываем знак **«принадлежит» (∈)**.

Если для числа стоит **равенство нулю**, то записываем знак **«не принадлежит» (∈)**.

Запись	$[A\&\mathbf{12}] \neq 0$	$[A\&\mathbf{68} \neq 0]$
Множество	$A \in \{8, 4\}$	$A \in \{64, 4\}$

5. Записываем оба полученных условия через операцию ИЛИ: $A \in \{8, 4\}$ ИЛИ $A \in \{64, 4\}$.

6. Смотрим на выражение с P ($[A\&P \neq 0]$).

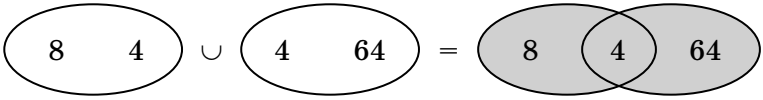
Если при P стоит **неравенство нулю**, то полученную запись **инвертируем**.

Если при P стоит **равенство нулю**, то полученную запись **оставляем как есть:**

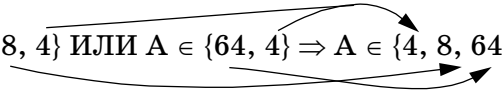
$$[A\&P = 0] \Rightarrow A \in \{8, 4\} \text{ ИЛИ } A \in \{64, 4\}.$$

7. Задача сводится к операции с множествами. Поскольку в обоих случаях A **принадлежит** соответствующим множествам, логическая операция **ИЛИ** соответствует **объединению множеств.**

При объединении двух множеств получаемое множество содержит все элементы объединяемых множеств, с исключением их повторов:



Следовательно, нужно выбрать **все элементы из обоих множеств по одному разу.**



8. Складываем все полученные в итоговом множестве числа — степени двойки:

P = 4 + 8 + 64 = **76.**

Ответ: 76.

Задача 7. Обозначим через & поразрядную конъюнкцию неотрицательных целых чисел. Для какого наименьшего неотрицательного целого числа *P* формула

$$[A\&61 \neq 0] \rightarrow ([A\&49 = 0] \rightarrow [A\&P \neq 0]) = 1$$

тождественно истинна (т.е. принимает значение 1 при любом неотрицательном целом значении переменной A)?

Решение

1. Преобразуем операцию следования по правилу: $X \rightarrow Y = \overline{X} \vee Y$.

$$[A\&\mathbf{61} = 0] \vee ([A\&\mathbf{49}] \neq 0] \vee [A\&P \neq 0]) = 1.$$

2. Раскладываем оба имеющихся числа на степени двойки:

Число десятичное	61	49
Число двоичное	111101_2	110001_2
Степени двойки	$32 + 16 + 8 + 4 + 1$	$32 + 16 + 1$

3. Полученные степени двойки записываем как множества:

Число	61	49
Множество	{1, 4, 8, 16, 32}	{1, 16, 32}

4. Если для числа стоит **неравенство нулю**, то записываем знак **«принадлежит» (∈)**.

Если для числа стоит **равенство нулю**, то записываем знак **«не принадлежит» (∉)**.

Запись	[A&**61** = 0]	[A&**49** ≠ 0]
Множество	A ∉ {1, 4, 8, 16, 32}	A ∈ {1, 16, 32}

5. Записываем оба полученных условия через ИЛИ:

$$A \notin \{1, 4, 8, 16, 32\} \text{ ИЛИ } A \in \{1, 16, 32\}.$$

6. Смотрим на выражение с P.

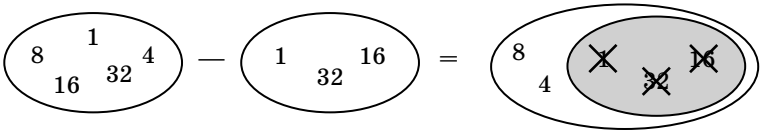
Если при P стоит **неравенство нулю**, то полученную запись **инвертируем**.

Если при P стоит **равенство нулю**, то полученную запись **оставляем как есть**:

$$[A\&P \neq 0] \Rightarrow \overline{A \notin \{1, 4, 8, 16, 32\} \text{ ИЛИ } A \in \{1, 16, 32\}} \Rightarrow$$
$$\Rightarrow A \in \{1, 4, 8, 16, 32\} \text{ И } A \notin \{1, 16, 32\}.$$

7. Задача сводится к операции с множествами. Здесь A **принадлежит первому множеству**, но **не принадлежит второму множеству**. Поэтому логическая операция **И** соответствует **разности множеств**.

При вычислении разности двух множеств получаемое множество содержит такие элементы первого множеств, которых нет во втором:



Следовательно, нужно выбрать **только те числа, которые в этих множествах не повторяются:**

$A \in \{\cancel{1}, 4, 8, \cancel{16}, \cancel{32}\}$ И $A \notin \{\cancel{1}, \cancel{16}, \cancel{32}\} \rightarrow A \in \{4, 8\}$.

8. Складываем все полученные в итоговом множестве числа — степени двойки:

$P = 4 + 8 = \mathbf{12}$.

Ответ: 12.

Задача 8. Обозначим через & поразрядную конъюнкцию неотрицательных целых чисел. Для какого наименьшего неотрицательного целого числа P формула $[A\&120 \neq 0] \rightarrow ([A\&56 \neq 0] \rightarrow [A\&P \neq 0])$ тождественно истинна, т.е. принимает значение 1 при любом неотрицательном целом значении переменной A?

Решение

1. Преобразуем операцию следования по правилу: $X \rightarrow Y = \overline{X} \vee Y$.

$(A\&120 \neq 0) \rightarrow ((A\&56 \neq 0) \rightarrow (A\&P \neq 0)) =$
$= \neg(A\&120 \neq 0) \vee (\neg(A\&56 \neq 0) \vee (A\&P \neq 0)) =$
$= (A\&120 = 0) \vee (A\&56 = 0) \vee (A\&P \neq 0)$.

2. Раскладываем оба имеющихся числа на степени двойки:

Число десятичное	120	56
Число двоичное	1111000_2	111000_2
Степени двойки	$64 + 32 + 16 + 8$	$32 + 16 + 8$

3. Полученные степени двойки записываем как множества:

Число	120	56
Множество	{64, 32, 16, 8}	{32, 16, 8}

4. Если для числа стоит **неравенство нулю**, то записываем знак **«принадлежит»** ($\in$). Если для числа стоит **равенство нулю**, то записываем знак **«не принадлежит»** ($\notin$).

Запись	**(A&120 = 0)**	**(A&56=0)**
Множество	$A \notin \{64, 32, 16, 8\}$	$A \notin \{32, 16, 8\}$

5. Записываем оба полученных условия через операцию ИЛИ:

$$A \notin \{64, 32, 16, 8\} \text{ ИЛИ } A \notin \{32, 16, 8\}.$$

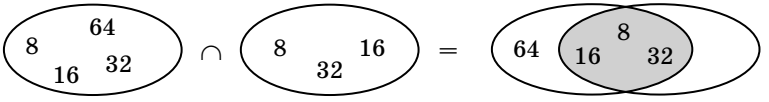
6. Смотрим **на** выражение с P (A&P ≠ 0).

Если при P стоит **неравенство нулю**, то полученную запись **инвертируем**. Если при P стоит **равенство нулю**, то полученную запись **оставляем как есть**:

$$(A\&P \neq 0) \quad \Rightarrow \quad A \in \{64, 32, 16, 8\} \text{ И } A \in \{32, 16, 8\}.$$

7. Задача сводится к операции с множествами. Здесь A **принадлежит обоим множествам**. Поэтому логическая операция **И** соответствует **пересечению множеств**.

При пересечении множеств получаемое множество содержит только элементы, которые имеются в обоих этих множествах:



Следовательно, нужно выбрать из обоих множеств **все числа, которые есть в обоих множествах**:

$$A \in \{32, 16, 8\}.$$

8. Складываем все полученные в итоговом множестве числа — степени двойки:

$P = 32 + 16 = 8 = \underline{\mathbf{56}}$.

Ответ: 56.

Задача 9. Сколько существует целых значений числа A, при которых формула

$$((x < 6) \to (x^2 < A)) \wedge ((y^2 \leqslant A) \to (y \leqslant 6))$$

тождественно истинна при любых целых неотрицательных x и y?

Решение (1-й способ)

1. В заданном выражении самой последней выполняется логическая операция И. Поэтому для истинности всего выражения необходимо, чтобы обе соединяемые ею части выражения были истинными:

$$\begin{cases}((x < 6) \to (x^2 < A)) = 1;\\ ((y^2 \leqslant A) \to (y \leqslant 6)) = 1.\end{cases}$$

При этом каждая из этих частей зависит только от одной переменной, так что их можно рассматривать по отдельности.

2. Рассматриваем первое логическое выражение: $((x < 6) \to (x^2 < A))$. Заменяем в нём операцию следования по правилу: $a \to b = \neg a \vee b$.

$$(x < 6) \to (x^2 < A) = \neg(x < 6) \vee (x^2 < A) = \\ = (x \geqslant 6) \vee (x^2 < A).$$

3. Начинаем рассмотрение с условия $(x \geqslant 6)$.

Всегда берём в качестве «опорного» условие без параметра А и смотрим, каким для его истинности/ложности должно тогда быть условие с А.

Если условие $(x \geqslant 6)$ истинно, то всё вышеприведённое выражение тоже истинно (операция ИЛИ). Поэтому такой случай мы не рассматриваем.

Если условие $(x \geqslant 6)$ ложно, то для истинности всего выражения условие $(x^2 < A)$ обязательно должно быть истинным.

4. Когда условие $(x \geqslant 6)$ ложно, будет истинным противоположное условие $(x < 6)$. Так как неравенство строгое, берём в качестве x ближайшее подходящее значение: $x = 5$.

Тогда, подставив найденное значение $x = 5$ во второе условие, получаем:

$(x^2 < A) = (5^2 < A) = (25 < A)$ или $A > 25$.

Это — первое граничное условие на значения A, которое, очевидно, определяет **минимальное** возможное значение этого параметра.

5. Рассматриваем второе логическое выражение: $((y^2 \leqslant A) \to (y \leqslant 6))$, тоже заменяя операцию следования на операцию ИЛИ: $(y^2 \leqslant A) \to (y \leqslant 6) = (y^2 > A) \vee (y \leqslant 6)$.

6. Начинаем рассмотрение с условия $(y \leqslant 6)$, которое не содержит параметра A.

Если условие $(y \leqslant 6)$ истинно, то всё вышеприведённое выражение тоже истинно (операция ИЛИ). Поэтому такой случай мы не рассматриваем.

Если условие $(y \leqslant 6)$ ложно, то для истинности всего выражения условие $(y^2 > A)$ обязательно должно быть истинным.

7. Когда условие $(y \leqslant 6)$ ложно, будет истинным противоположное условие $(y > 6)$. Так как неравенство строгое, берём в качестве y ближайшее подходящее значение: $y = 7$.

Тогда, подставив найденное значение $y = 7$ во второе условие, получаем:
$(y^2 > A) = (7^2 > A) = (49 > A)$ или $A < 49$, что определяет **максимальное** возможное значение A.

8. Итак, для A мы получили интервал возможных значений $(25, 49)$, в котором обе границы в интервал **не входят** (соответствующие знаки неравенств — строгие). Остаётся определить количество целых чисел, входящих в этот интервал: $49 - 25 - 1 = 23$ (единица дополнительно вычитается, так как в интервал не входят **обе** границы).

Ответ: 23.

Решение (2-й способ)

1. Аналогично предыдущему способу решения, преобразуем исходное выражение в систему из двух логических выражений, которые оба должны быть истинными:

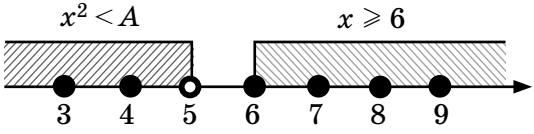
$$\begin{cases} (x < 6) \to (x^2 < A) = 1; \\ (y^2 \leqslant A) \to (y \leqslant 6) = 1. \end{cases}$$

2. Рассмотрим первое из них, заменяя в нём операцию следования по правилу: $a \to b = \neg a \vee b$.

$(x < 6) \to (x^2 < A) = \neg(x < 6) \vee (x^2 < A) = (x \geqslant 6) \vee$
$\vee (x^2 < A)$.

3. Изобразим оба соответствующих интервала на координатной прямой, не забывая при этом, что речь идёт

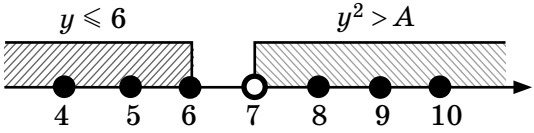
о **дискретном наборе целых чисел.** Сначала изображаем интервал, соответствующий условию $x \geqslant 6$ (значение 6 входит в интервал). Теперь нам нужно условием $x^2 < A$ покрыть оставшуюся часть координатной прямой — от значения 5 и меньше. Получаем, что $5^2 < A$, откуда $A > 25$.



4. Аналогичным образом преобразуем второе уравнение:

$(y^2 \leqslant A) \to (y \leqslant 6) = \neg(y^2 \leqslant A) \vee (y \leqslant 6) = (y^2 > A) \vee$
$\vee (y \leqslant 6)$.

5. Изобразим эти два интервала на такой же «дискретной» координатной прямой. Сначала изображаем интервал для условия $y \leqslant 6$, а затем покрываем вторым условием оставшуюся часть чисел от 7 и больше. Получаем, что $7^2 > A$, откуда $A < 49$.



6. Таким образом, для A мы получили интервал возможных значений (25, 49), в котором обе границы в интервал не входят (соответствующие знаки неравенств — строгие). Определим количество целых чисел, входящих в этот интервал: $49 - 25 - 1 = 23$.

Ответ: 23.

Задача 10. Сколько существует целых значений числа A, при которых формула

$$((x < A) \to (x^2 < 100)) \wedge ((y^2 \leqslant 64) \to (y \leqslant A))$$

тождественно истинна при любых целых неотрицательных x и y?

Решение

1. Две операции следования соединены операцией И, тогда должны быть истинными оба эти следования:

$$\begin{cases}(x < A) \to (x^2 < 100) = 1;\\ (y^2 \leqslant 64) \to (y \leqslant A) = 1.\end{cases}$$

Избавившись от операции следования, получаем систему уравнений:

$$\begin{cases}(x \geqslant A) \vee (x^2 < 100) = 1;\\ (y^2 > 64) \vee (y \leqslant A) = 1.\end{cases}$$

2. Рассмотрим первое уравнение $(x \geqslant A) \vee (x^2 < 100) = 1$. Рассмотрение начинаем с условия $(x^2 < 100)$, которое не содержит параметра.

Если условие $(x^2 < 100)$ истинно, то всё вышеприведённое выражение тоже истинно (операция ИЛИ). Поэтому такой случай мы не рассматриваем.

Если условие $(x^2 < 100)$ ложно, то для истинности всего выражения условие $(x \geqslant A)$ обязательно должно быть истинным.

3. Когда условие $(x^2 < 100)$ ложно, будет истинным противоположное условие: $(x^2 \geqslant 100)$. Тогда $x = 10$ (неравенство нестрогое).

Подставив найденное значение $x = 10$ во второе условие, получаем:
$(x \geqslant A) = (10 \geqslant A)$ или $A \leqslant 10$. Это **верхняя** граница допустимого интервала параметра.

4. Рассмотрим второе уравнение $(y^2 > 64) \vee (y \leqslant A) = 1$. Рассмотрение начинаем с условия $(y^2 > 64)$, которое не содержит параметра.

Если условие $(y^2 > 64)$ истинно, то всё вышеприведённое выражение тоже истинно (операция ИЛИ). Поэтому такой случай мы не рассматриваем.

Если условие $(y^2 > 64)$ ложно, то для истинности всего выражения условие $(y \leqslant A)$ обязательно должно быть истинным.

5. Когда условие $(y^2 > 64)$ ложно, будет истинным противоположное условие: $(y^2 \leqslant 64)$. Тогда $y = 8$ (неравенство нестрогое).

Подставив найденное значение $y = 8$ во второе условие, получаем: $(y \leqslant A) = (8 \leqslant A)$ или $A \geqslant 8$.

Это **нижняя** граница допустимого интервала параметра.

6. Получили интервал значений [8, 10], где обе границы **входят в интервал** (соответствующие знаки неравенств — нестрогие). Поэтому количество подходящих целых значений A вычисляется как $10 - 8 + 1 = 3$.

Ответ: 3.

Возможны варианты подобных задач, в которых две части выражения соединены не операцией И, а операцией ИЛИ, но требуется найти значения параметра, обеспечивающие тождественную ложность всего выражения. В этом случае принципы решения остаются теми же самыми, но:

- либо в самом начале утверждаем, что если заданное выражение тождественно ложно, то отрицание всего этого выражения — тождественно истинно; в этом случае соединяющая части выражения операция ИЛИ преобразуется в операцию И, соединяющую отрицания этих частей выражения, а сама задача сводится к уже рассмотренным вариантам;
- либо рассматриваем исходное выражение без изменений, рассуждая, что всё выражение может быть тождественно ложным только в том случае, если тождественно ложны обе его части, соединённые операцией ИЛИ, — после чего опять-таки разделяем выражение на две отдельно рассматриваемые его части и выполняем решение по аналогии с рассмотренными выше.

Задача 11. На числовой прямой задан отрезок A. Известно, что формула

$$((x \in A) \to (x^2 \leqslant 100)) \wedge ((x^2 \leqslant 64) \to (x \in A))$$

тождественно истинна при любом вещественном x. Какую наименьшую длину может иметь отрезок A?

Решение

1. Как и в предыдущих задачах, сначала разделяем исходное выражение на две независимые части, соединённые операцией И.

$((x \in A) \to (x^2 \leqslant 100)) \wedge ((x^2 \leqslant 64) \to (x \in A)) = 1 \quad \Rightarrow$

$$\begin{cases} (x \in A) \to (x^2 \leqslant 100) = 1; \\ (x^2 \leqslant 64) \to (x \in A) = 1. \end{cases}$$

2. Рассматриваем первое выражение и избавляемся от операции следования:

$(x \in A) \to (x^2 \leqslant 100) = \neg(x \in A) \vee (x^2 \leqslant 100) =$

$= (x \notin A) \vee (x^2 \leqslant 100).$

Если $(x^2 \leqslant 100)$ — истинно, то выражение истинно при любом A, а нам нужна минимальная длина отрезка A. Поэтому рассматриваем случай, когда $(x^2 \leqslant 100)$ ложно, т.е. $(x^2 > 100)$.

Тогда $x \in (-\infty, -10) \cup (10, \infty)$.

В этом случае нужно, чтобы $(x \notin A)$ было истинно, т.е. отрезок A должен покрывать ту часть координатной прямой, которая не закрыта отрезком x. Следовательно, $A \in [-10, 10]$.

3. Рассматриваем второе выражение и избавляемся от операции следования:

$(x^2 \leqslant 64) \to (x \in A) = \neg(x^2 \leqslant 64) \vee (x \in A) =$

$= (x^2 > 64) \vee (x \in A).$

Если $(x^2 > 64)$ — истинно, то выражение истинно при любом A, а нам нужна минимальная длина отрезка A. Поэтому рассматриваем случай, когда $(x^2 > 64)$ ложно, т.е. $(x^2 \leqslant 64)$.

Тогда $x \in [-8, 8]$.

В этом случае нужно, чтобы $(x \in A)$ было истинно, т.е. отрезок A должен совпадать с x. Следовательно, $A \in [-8, 8]$.

4. В первом случае $A \in [-10, 10]$, во втором — $A \in [-8, 8]$. Берём из них наименьший.

Проверка.

В первом случае требуется $(x \notin A) \vee (x^2 \leqslant 100) = 1$. Если $(x^2 \leqslant 100) = 1$, то выражение истинно на всем интервале $[-10, 10]$, так что если брать $A \in [-8, 8]$, то на отрезках $[-10, -8)$ и $(8,10]$ получаем, что и $(x \notin A) = 1$, и $(x^2 \leqslant 100) = 1$, что нас устраивает.

Во втором случае требуется $(x^2 > 64) \vee (x \in A) = 1$. Если брать $A \in [-10, 10]$, то на отрезках $[-10, -8)$ и $(8,10]$ получаем, что и $(x^2 > 64) = 1$, и $(x \in A) = 1$, что тоже нас устраивает.

5. Вычисляем длину отрезка $[-8, 8]$: $8 - (-8) = 8 + 8 = 16$. (Длина отрезка всегда вычисляется как разность координат его концов.)

Ответ: 16.

Задача 12. Каково наименьшее значение числа A, для которого формула

$$(n > 10) \vee (m \geqslant 20) \vee (2n+m < A)$$

тождественно истинна при любых целых неотрицательных значениях n и m?

Решение

Раньше заданное выражение не содержало «смешанного» условия, в котором имелись бы обе переменные (в данном случае — m и n), поэтому можно было разделить его на две независимые части и решать каждую отдельно как одно из уравнений системы. Как быть теперь?

А теперь мы решаем только одно уравнение, а не два, и метод решения аналогичен предыдущим задачам!

1) В заданном выражении выделяем две части: одну — в которой имеется параметр A, а другую — без этого параметра:

$$\underbrace{(n > 10) \vee (m \geqslant 20)}_{\text{без } A} \vee \underbrace{(2n+m < A)}_{\text{с } A}$$

2) Всё выражение должно быть истинным для всех m и n, а обе выделенные части соединены операцией ИЛИ. Поэтому рассуждаем следующим образом:

- Если условие $(n > 10) \vee (m \geqslant 20)$ истинно, то всё выражение будет истинным независимо от второго условия и от значения параметра A. Поэтому данный случай можно не рассматривать.

• Если условие $(n > 10) \vee (m \geqslant 20)$ ложно, то выражение будет истинным только за счёт истинности второго условия. Именно эту ситуацию нам и нужно рассмотреть.

3) Если условие $(n > 10) \vee (m \geqslant 20)$ ложно, то истинным является противоположное:

$\neg(n > 10) \vee (m \geqslant 20) = \neg(n > 10) \wedge \neg(m \geqslant 20) =$
$= (n \leqslant 10) \wedge (m < 20)$.

Именно эта ситуация для нас является «критичной». А граничными значениями для неё являются значения $n = 10$ (так как соответствующее неравенство — нестрогое) и $m = 19$ (так как соответствующее неравенство строгое, а число 19 — это первое, которое подходит согласно условию).

4) Подставив найденные значения m и n во вторую часть выражения, ищем значение A:

$(2n+m < A) \Rightarrow (2\times10 + 19 < A) \Rightarrow (39 < A) \Rightarrow (A > 39)$.

Из этого условия определяется и искомое наименьшее значение A. Поскольку неравенство строгое, наименьшее значение A равно 40.

Ответ: 40.

Решение систем логических уравнений

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: $\vee$ или +.

Таблица истинности:

A	B	A $\vee$ 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: $\equiv$ или $\leftrightarrow$.

Таблица истинности:

A	B	A $\leftrightarrow$ 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»:

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

Поразрядные (побитовые) логические операции

Кроме обычных логических операций, применимых по отношению к логическим переменным, возможны поразрядные (побитовые) логические операции, выполняемые для пар «одноимённых» (соответствующих одним и тем же разрядам) битов двух целых чисел. При этом двоичное значение 1 рассматривается как «истина», а значение 0 — как «ложь». Результатом выполнения поразрядной логической операции является целое число.

- *Поразрядная конъюнкция*

Для каждой пары битов выполняется логическая операция «И».

Пример:

$$\begin{array}{r} 15 \\ \& \underline{28} \\ ? \end{array} \quad \to \quad \begin{array}{r} 01111 \\ \& \underline{11100} \\ 01100 \end{array} \quad \to \quad \begin{array}{r} 15 \\ \& \underline{28} \\ 12 \end{array}$$

- *Поразрядная дизъюнкция*

Для каждой пары битов выполняется логическая операция «ИЛИ».

Пример:

$$\begin{array}{r} \vee 14 \\ 28 \\ \hline ? \end{array} \quad \rightarrow \quad \begin{array}{r} \vee 01110 \\ 11100 \\ \hline 11100 \end{array} \quad \rightarrow \quad \begin{array}{r} \vee 14 \\ 28 \\ \hline 30 \end{array}$$

Основные законы алгебры логики

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

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

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

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

Разбор типовых задач

Задача 1. 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:

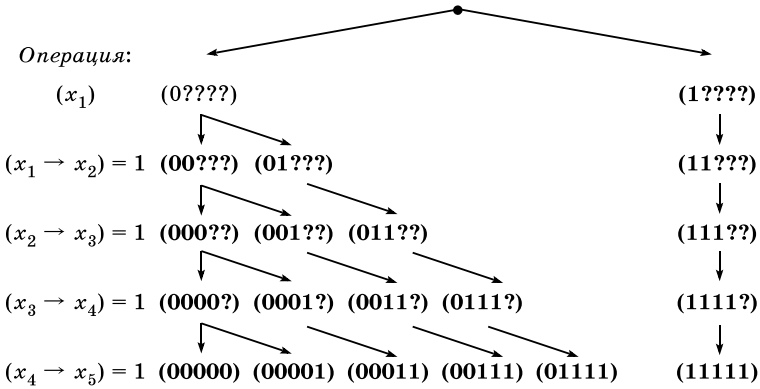
$(x_1 \to x_2) \wedge (x_2 \to x_3) \wedge (x_3 \to x_4) \wedge (x_4 \to x_5) = 1.$

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: $y_5 \to x_5 = 1$.

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 $y_5 \to x_5 = 1$ 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 подчёркиванием.

$(y_1y_2y_3y_4y_5)$	$(x_1x_2x_3x_4x_5)$						***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 2. 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

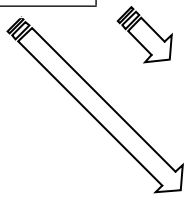


Таблица 1.2

x_7	x_8	x_9	x_{10}
0	1 (¬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)

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

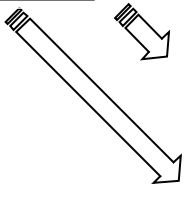


Таблица 2.2

x_5	x_6	x_7	x_8
0	1 (¬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)

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 ($\neg$0)	0	1 ($\neg$0)	3 ❶
		0	0 ($\neg$1)	3 ❷
		1	1 ($\neg$0)	3 ❸
0	0 ($\neg$1)	0	1 ($\neg$0)	3 ❶
		0	0 ($\neg$1)	3 ❷
		1	1 ($\neg$0)	3 ❸
1	1 ($\neg$0)	0	1 ($\neg$0)	3 ❶
		0	0 ($\neg$1)	3 ❷
		1	1 ($\neg$0)	3 ❸

Таблица 1.2

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

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 ($\neg$ 0)	0	1 ($\neg$ 0)	9 (3+3+3) ❶
		0	0 ($\neg$ 1)	9 (3+3+3) ❷
		1	1 ($\neg$ 0)	9 (3+3+3) ❸
0	0 ($\neg$ 1)	0	1 ($\neg$ 0)	9 (3+3+3) ❶
		0	0 ($\neg$ 1)	9 (3+3+3) ❷
		1	1 ($\neg$ 0)	9 (3+3+3) ❸
1	1 ($\neg$ 0)	0	1 ($\neg$ 0)	9 (3+3+3) ❶
		0	0 ($\neg$ 1)	9 (3+3+3) ❷
		1	1 ($\neg$ 0)	9 (3+3+3) ❸

Таблица 2.2

x_5	x_6	x_7	x_8	Кол-во вариантов
0 ❶	1 ($\neg$ 0)	0	1 ($\neg$ 0)	3
		0	0 ($\neg$ 1)	3
		1	1 ($\neg$ 0)	3
0 ❷	0 ($\neg$ 1)	0	1 ($\neg$ 0)	3
		0	0 ($\neg$ 1)	3
		1	1 ($\neg$ 0)	3
1 ❸	1 ($\neg$ 0)	0	1 ($\neg$ 0)	3
		0	0 ($\neg$ 1)	3
		1	1 ($\neg$ 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** $(\neg 0)$	**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** $(\neg 0)$

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 учётом x_5, x_6, x_7, x_8, x_9, x_{10}
0	1 $(\neg 0)$	0	1 $(\neg 0)$	27 (9+9+9) ❶
		0	0 $(\neg 1)$	27 (9+9+9) ❷
		1	1 $(\neg 0)$	27 (9+9+9) ❸
0	0 $(\neg 1)$	0	1 $(\neg 0)$	27 (9+9+9) ❶
		0	0 $(\neg 1)$	27 (9+9+9) ❷
		1	1 $(\neg 0)$	27 (9+9+9) ❸
1	1 $(\neg 0)$	0	1 $(\neg 0)$	27 (9+9+9) ❶
		0	0 $(\neg 1)$	27 (9+9+9) ❷
		1	1 $(\neg 0)$	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 $(\neg 0)$	0	1 $(\neg 0)$	9
		0	0 $(\neg 1)$	9
		1	1 $(\neg 0)$	9
0	0 $(\neg 1)$	0	1 $(\neg 0)$	9
		0	0 $(\neg 1)$	9
		1	1 $(\neg 0)$	9
1	1 $(\neg 0)$	0	1 $(\neg 0)$	9
		0	0 $(\neg 1)$	9
		1	1 $(\neg 0)$	9

Ответ: 243 варианта.

Задача 3. Сколько существует различных наборов значений логических переменных a_1, a_2, …, a_5, b_1, b_2, …, b_5, c_1, c_2, …, c_5, которые удовлетворяют перечисленным уравнениям?

$(a_1 \to a_2) \wedge (a_2 \to a_3) \wedge (a_3 \to a_4) \wedge (a_4 \to a_5) = 1$
$(b_1 \to b_2) \wedge (b_2 \to b_3) \wedge (b_3 \to b_4) \wedge (b_4 \to b_5) = 1$
$(c_1 \to c_2) \wedge (c_2 \to c_3) \wedge (c_3 \to c_4) \wedge (c_4 \to c_5) = 1$
$a_1 \wedge b_2 \wedge c_3 = 0.$

В качестве ответа нужно указать количество таких наборов переменных.

Решение

1) Анализируем первое уравнение:
$(a_1 \to a_2) \wedge (a_2 \to a_3) \wedge (a_3 \to a_4) \wedge (a_4 \to a_5) = 1.$

Начинаем составлять таблицу возможных значений переменных a_1, a_2, …, a_5. При этом учитываем, что при использовании логической операции «И» результат 1 обеспечивается только единичными значениями всех её аргументов, а также помним, что операция следования истинна в трёх случаях: $1 \to 1$, $0 \to 1$ и $0 \to 0$. Поэтому если в каждом следовании первая переменная равна единице, то из неё может следовать только единица, а из нуля может следовать как единица, так и нуль. В результате получаем следующую таблицу значений переменных:

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

2) Второе уравнение полностью аналогично первому, только имена переменных другие. Поэтому таблица значений переменных будет такая же. И то же самое можно сказать про третье уравнение. Получаем:

b_1	b_2	b_3	b_4	b_5
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	0

c_1	c_2	c_3	c_4	c_5
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	0

3) Анализируем четвёртое уравнение. Оно является *«ключевым»*:

$$a_1 \wedge b_2 \wedge c_3 = 0.$$

Строим для него таблицу истинности:

a_1	b_2	c_3	**Результат**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Из неё нам требуются все строки, кроме последней.

4) Подсчитаем теперь количества значений переменных, соответствующие таблице истинности четвёртого

уравнения. Для этого в таблицах значений переменных первых трёх уравнений выделяем столбцы, соответствующие переменным a_1, b_2 и c_3:

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

b_1	b_2	b_3	b_4	b_5
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	0

c_1	c_2	c_3	c_4	c_5
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	0

И теперь для каждой строки таблицы значений четвёртого уравнения перемножаем друг на друга количества нулей и единиц соответственно, содержащихся в выделенном столбце для каждой из переменных:

a_1	b_2	c_3	**Кол-во наборов переменных**
0	0	0	$5 \cdot 4 \cdot 3 = 60$
0	0	1	$5 \cdot 4 \cdot 3 = 60$
0	1	0	$5 \cdot 2 \cdot 3 = 30$
0	1	1	$5 \cdot 2 \cdot 3 = 30$
1	0	0	$1 \cdot 4 \cdot 3 = 12$
1	0	1	$1 \cdot 4 \cdot 3 = 12$
1	1	0	$1 \cdot 2 \cdot 3 = 6$
Итого:			$\mathbf{60 + 60 + 30 + 30 + 12 + 12 + 6 = 210}$

Ответ: 210.

Задача 4. Сколько существует различных наборов значений логических переменных x_1, x_2, ..., x_8, y_1, y_2, ..., y_8, которые удовлетворяют всем перечисленным условиям?

$(x_1 \to x_2) \wedge (x_1 \to y_1) = 1$
$(x_2 \to x_3) \wedge (x_2 \to y_2) = 1$
…
$(x_7 \to x_8) \wedge (x_7 \to y_7) = 1$
$(x_8 \to y_8) = 1.$

В ответе **не нужно** перечислять все различные наборы значений переменных x_1, x_2, ..., x_8, y_1, y_2, ..., y_8, при которых выполнена данная система равенств. В качестве ответа нужно указать только количество таких наборов.

Решение

1. Рассмотрим первое уравнение: $(x_1 \to x_2) \wedge (x_1 \to y_1) = 1$. Из него следует, что обе операции следования должны быть истинными (равны 1):

$$\begin{cases}(x_1 \to x_2) = 1;\\ (x_1 \to y_1) = 1.\end{cases}$$

2. Операция следования истинна в трёх случаях: 0 0, 0 1, 1 1. То есть из единицы всегда должна следовать единица, а из нуля может следовать что угодно (варианты «размножаются»). Тогда, учитывая, что все «иксы» следуют друг из друга, можно по всем имеющимся уравнениям сразу построить таблицу возможных значений для всех операций следования «с иксами» в виде:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Полезно запомнить, как выглядит эта матрица значений переменных для цепочки следований, — это ещё не раз вам пригодится!

Согласно этой таблице, для истинности всех операций следования «с иксами» существует ровно 9 возможных наборов значений переменных x (столько, сколько строк в полученной таблице).

3. Кроме уже рассмотренных операций «с иксами», у нас в систему уравнений входят операции следования вида $(x_i \to y_i)$ для всех $i = 1 \ldots 8$, которые должны быть тоже истинными. Их истинность зависит от найденных значений переменных x. При этом каждое y_i зависит от значения «одноимённого» ему x_i, что отслеживается сразу по всем соответствующим условиям из всех уравнений.

Допишем это в составленную нами таблицу. При этом учитываем следующие соображения:

- если некоторый x равен 1, то соответствующий ему y однозначно равен 1;
- если некоторый x равен 0, то соответствующий ему y может быть равен как 0, так и 1;
- по каждой **строке** суммарное количество вариантов («Всего») вычисляется как **произведение** количеств вариантов для каждого y_i, а **общее количество** вариантов («Итого») — как **сумма** полученных количеств вариантов в графе «Всего».

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Кол-во возможных вариантов								Всего
								y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	*2*	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2
0	*0*	**1**	**1**	**1**	**1**	**1**	**1**	*2*	*2*	**1**	**1**	**1**	**1**	**1**	**1**	4
0	*0*	*0*	**1**	**1**	**1**	**1**	**1**	*2*	*2*	*2*	**1**	**1**	**1**	**1**	**1**	8
0	*0*	*0*	*0*	**1**	**1**	**1**	**1**	*2*	*2*	*2*	*2*	**1**	**1**	**1**	**1**	16
0	*0*	*0*	*0*	*0*	**1**	**1**	**1**	*2*	*2*	*2*	*2*	*2*	**1**	**1**	**1**	32
0	*0*	*0*	*0*	*0*	*0*	**1**	**1**	*2*	*2*	*2*	*2*	*2*	*2*	**1**	**1**	64
0	*0*	*0*	*0*	*0*	*0*	*0*	**1**	*2*	*2*	*2*	*2*	*2*	*2*	*2*	**1**	128
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	256
														Итого:		**511**

Ответ: 511.

Задача 5. Сколько существует различных наборов значений логических переменных x_1, x_2, ..., x_7, x_8, y_1, y_2, ..., y_7, y_8, которые удовлетворяют всем перечисленным ниже условиям?

$(x_1 \equiv x_2) \equiv (y_1 \equiv y_2) = 1$

$(x_2 \equiv x_3) \equiv (y_2 \equiv y_3) = 1$

...

$(x_7 \equiv x_8) \equiv (y_7 \equiv y_8) = 1$.

В ответе не нужно перечислять все различные наборы значений переменных x_1, x_2, ..., x_7, x_8, y_1, y_2, ..., y_7, y_8, при которых выполнена данная система равенств. В качестве ответа нужно указать количество таких наборов.

Решение

1. Построим таблицу допустимых значений для первого уравнения $(x_1 \equiv x_2) \equiv (y_1 \equiv y_2) = 1$.

При этом учитываем, что операция тождества истинна, если оба значения одинаковы (1 и 1 либо 0 и 0) и ложна, если значения различны (1 и 0 либо 0 и 1).

Обе части уравнения	x_1	x_2	y_1	y_2
1	1	1	1	1
			0	0
	0	0	1	1
			0	0
0	1	0	1	0
			0	1
	0	1	1	0
			0	1

2. Для всех уравнений таблицы допустимых значений однотипны. Повторяем для второго уравнения $(x_2 \equiv x_3) \equiv (y_2 \equiv y_3) = 1$ ту же таблицу, добавив справа столбец количества наборов переменных с учётом предыдущей таблицы.

Чтобы определить эти количества наборов переменных, в каждой строке новой таблицы нужно посмотреть значения x_2 и y_2 (которыми вторая таблица «связана»

с первой) и записать, сколько в предыдущей таблице имеется строк с такими значениями x_2 и y_2.

Обе части уравнения	x_1	x_2	y_1	y_2
1	1	1	1	1
			0	0
	0	0	1	1
			0	0
0	1	0	1	0
			0	1
	0	1	1	0
			0	1

Обе части уравнения	x_2	x_3	y_2	y_3	Кол-во наборов
1	1	1	1	1	2
			0	0	2
	0	0	1	1	2
			0	0	2
0	1	0	1	0	2
			0	1	2
	0	1	1	0	2
			0	1	2

Например:

- для случая $x_2 = 1$ и $y_2 = 1$ в первой таблице есть две строки (самая первая и самая последняя), — значит, во второй таблице во всех строках, где $x_2 = 1$ и $y_2 = 1$, запишем количество вариантов, равное 2;
- для случая $x_2 = 1$ и $y_2 = 0$ в первой таблице есть тоже две строки (вторая и предпоследняя), — значит, во второй таблице во всех строках, где $x_2 = 1$ и $y_2 = 0$, тоже пишем количество 2;

- аналогично поступаем и для остальных сочетаний x_2 и y_2 (количество вариантов получается везде по два).

Делаем вывод: второе уравнение **удваивает** число вариантов, имевшихся в первом.

3. Аналогично можно составить таблицу для третьего уравнения $(x_3 \equiv x_4) \equiv (y_3 \equiv y_4) = 1$.

Чтобы определить количества наборов переменных, в каждой строке этой таблицы нужно посмотреть значения x_3 и y_3, которыми третья таблица «связана» со второй, и записать, сколько в предыдущей таблице имеется вариантов с такими значениями x_3 и y_3, помня, что во второй таблице каждая строка уже содержит некоторое количество вариантов (в нашем случае — по 2).

Обе части уравнения	x_3	x_4	y_3	y_4	Кол-во наборов
1	1	1	1	1	4
			0	0	4
	0	0	1	1	4
			0	0	4
0	1	0	1	0	4
			0	1	4
	0	1	1	0	4
			0	1	4

Делаем вывод: третье уравнение тоже удваивает число вариантов, имевшихся во втором.

4. Обобщив этот вывод, мы можем предположить, что каждое последующее уравнение удваивает число вариантов предыдущего по каждой строке таблицы, и сразу записать таблицы для оставшихся уравнений.

Обе части уравнения	x_4	x_5	y_4	y_5	Кол-во наборов
1	1	1	1	1	8
			0	0	8
	0	0	1	1	8
			0	0	8
0	1	0	1	0	8
			0	1	8
	0	1	1	0	8
			0	1	8

Обе части уравнения	x_5	x_6	y_5	y_6	Кол-во наборов
1	1	1	1	1	16
			0	0	16
	0	0	1	1	16
			0	0	16
0	1	0	1	0	16
			0	1	16
	0	1	1	0	16
			0	1	16

Обе части уравнения	x_6	x_7	y_6	y_7	Кол-во наборов
1	1	1	1	1	32
			0	0	32
	0	0	1	1	32
			0	0	32
0	1	0	1	0	32
			0	1	32
	0	1	1	0	32
			0	1	32

Обе части уравнения	x_7	x_8	y_7	y_8	Кол-во наборов
1	1	1	1	1	64
			0	0	64
	0	0	1	1	64
			0	0	64
0	1	0	1	0	64
			0	1	64
	0	1	1	0	64
			0	1	64

5. В последней полученной таблице остаётся только просуммировать количества наборов по всем строкам, тогда общее количество наборов переменных равно $64 \times 8 = 512$.

Ответ: 512.

Задача 6. Сколько существует различных наборов значений логических переменных x_1, x_2, …, x_5, y_1, y_2, …, y_5, которые удовлетворяют всем перечисленным ниже условиям?

$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_4) \wedge (x_4 \vee \neg x_5) = 1$;
$(\neg y_1 \vee y_2) \wedge (\neg y_2 \vee y_3) \wedge (\neg y_3 \vee y_4) \wedge (\neg y_4 \vee y_5) = 1$;
$x_1 \vee y_1 = 1$.

В ответе не нужно перечислять все различные наборы значений переменных, при которых выполнена данная система равенств. В качестве ответа нужно указать количество таких наборов.

Решение

1. Начинаем рассмотрение с первого уравнения. Таблицу заполняем слева направо, помня, что каждое из выражений с ИЛИ должно быть истинным, так как они связаны операцией И.

При этом в выражении типа $x_1 \vee \neg x_2$ для $x_1 = 0$ значение $\neg x_2$ может быть только ненулевым, т.е. x_2 может быть только нулём, а при $x_1 = 1$ возможны любые значения x_2. То есть в таблице нуль сохраняется, а по 1 идёт «размножение» строк:

x_1	x_2	x_3	x_4	x_5
0	0	0	0	0
1	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1

2. Для второго уравнения — аналогично, но в выражении типа $\neg y_1 \vee y_2$, наоборот, при $y_1 = 1$ значение y_2 должно быть только 1, а при $y_1 = 0$ значение y_2 может быть любое, т.е. в таблице единицы сохраняются, а нули «размножаются».

y_1	y_2	y_3	y_4	y_5
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	0

3. Третье уравнение — «ключевое», оно показывает, что нам подходят все полученные выше наборы, в которых x_1 и y_1 одновременно не будут нулевыми. Причём, так как оба первых уравнения независимы, надо комбинировать возможные варианты.

Проще вычислить так. Общее количество вариантов равно 6×6 (для каждого набора первой таблицы возможно 6 наборов из второй), т.е. 36 вариантов. Но из них «запрещённых» (когда $x_1 = 0$ и $y_1 = 0$ одновременно) — 5 вариантов (для одного такого варианта в первой таблице — 5 вариантов во второй). Значит, допустимых — $36 - 5 = 31$ вариант.

Ответ: 31.

Раздел 5. **Элементы теории алгоритмов**

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 трёхзначное число. По этому числу формируется новое число по следующим правилам.

1. Складываются первая и вторая, а затем — вторая и третья цифры исходного числа.

2. Полученные два числа записываются друг за другом подряд в порядке возрастания.

Пример. Исходное число: 176. Полученные числа: 1 + 7 = 8, 7 + 6 = 13. Результат: 813.

Найдите наибольшее исходное число, для которого автомат выдаст результат 815.

Решение

Сначала определим, как можно «разрезать» результирующее число на два, вычисленных автоматом.

Вспомним, что сумма двух десятичных цифр (которые могут быть равны от 0 до 9) может равняться от 0 (0+0) до 18 (9+9).

Если пытаться разделить число 815 как 81 и 5, то первое из этих чисел не соответствует этому возможному диапазону значений сумм цифр (да к тому же тогда числа записаны не по возрастанию).

Поэтому остаётся только один вариант: число 815 как 8 и 15.

Теперь посмотрим, какие цифры могут давать такие суммы:

- 8 = 0 + 8, 1 + 7, 2 + 6, 3 + 5, 4 + 4;
- 15 = 9 + 6, 8 + 7.

А теперь — самое главное. По условию, одна сумма — это сумма первой и второй цифры исходного числа, а вторая — это сумма второй и третьей цифры. Вторая цифра, таким образом, должна повторяться в обеих суммах.

Ищем такие две суммы в обеих «подборках» (для числа 8 и для числа 15).

Это пары: 0 + 8 и 8 + 7; 1 + 7 и 8 + 7; 9 + 6 и 2 + 6.

Им соответствуют числа (учитывая, что нуль не может быть записан самым первым — тогда он был бы незначащим, а число — двузначным): 780, 178, 871, 962, 269 (везде повторяющаяся цифра стоит в середине).

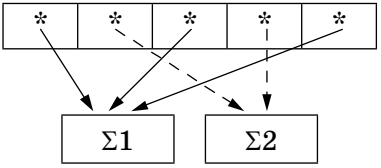
Остаётся найти среди них наибольшее. Очевидно, это число 962.

Ответ: 962.

Задача 2. Процессор получает пятизначное число и по нему вычисляет новое, используя следующие правила:

1) отдельно суммируются первая, третья и пятая цифры, а также вторая и четвёртая цифры;

2) два полученных числа записываются подряд по неубыванию.



где $\Sigma 1 \leqslant \Sigma 2$

Требуется определить наименьшее возможное число, при обработке которого будет получен результат 621.

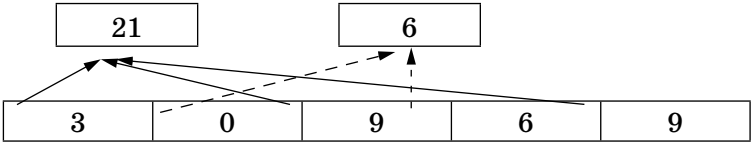
Решение

1. Сумма трёх десятичных цифр может иметь значение от 1 («1+0+0», причём первая цифра числа — всегда ненулевая!) до 27 («9+9+9»). Сумма двух десятичных цифр может иметь значение от 0 («0+0») до 18 («9+9»).

2. Анализируем число 621. Это может быть «6 и 21» либо «62 и 1». Но условию записи чисел в порядке неубывания соответствует только пара 6 и 21.

3. Так как сумма двух цифр не может быть равна 21, можно сделать вывод: сумма 1-й, 3-й и 5-й цифр исходного числа равна 21, а сумма 2-й и 4-й цифр равна 6.

4. Чтобы исходное число было наименьшим из возможных, в нём цифры должны быть записаны слева направо по возрастанию. Таким условиям соответствует число 30969:



Рассуждения при «конструировании» этого числа следующие:

- для 2-й и 4-й цифр вторую цифру (более левую) выбираем минимально возможной — равной 0, тогда 4-я цифра равна 6;
- для 1-й, 3-й и 5-й цифр тоже стараемся взять первую цифру минимально возможной: это 3, так как тогда на сумму 3-й и 5-й цифр остаётся 18 — максимально возможное значение суммы 2 цифр; тогда остаётся взять 3-ю и 5-ю цифры равными 9.
  Ответ: 30969

Задача 3. Исполнитель преобразует поданное на вход натуральное число N в новое число M по следующим правилам:

1) записывается двоичное представление числа N;

2) к полученному двоичному числу справа приписывается ещё один разряд, равный 0, если в исходном числе количество двоичных единиц чётно, либо равный 1, если в исходном числе количество двоичных единиц нечётно (примеры: 10111 → 101110, 10011 → 100111);

3) над полученным числом (включая и только что дописанный справа дополнительный разряд) ещё раз выполняется та же процедура — дописывание справа ещё одного разряда, равного 0 при чётном количестве единиц или 1 при нечётном их количестве.

Полученная таким образом запись (в которой на два разряда больше, чем в двоичной записи числа N) — это двоичная запись результирующего числа M.

Определите минимальное число M, превышающее 63, которое может являться результатом работы данного исполнителя. (Ответ записывается как десятичное значение.)

 Условие, которое определяет, дописывается ли к числу справа разряд, равный 0 или 1, может быть сформулировано и иначе: «все цифры исходного числа складываются, и справа дописывается остаток от деления суммы этих цифр на 2». Нетрудно понять, что это условие полностью аналогично определению чётного или нечётного количества в числе двоичных единиц.

Решение

Нам нужно найти такое десятичное число, двоичная запись которого могла бы быть получена путём дописывания справа битов по указанным правилам.

1. Поскольку требуется найти минимальное значение, превышающее (т.е. строго большее) число 63, начинаем проверку с числа 64.

$64_{10} = 2^6 = 1000000_2$

Может ли крайний справа бит (0) быть дописанным к двоичному числу 100000? Нет, потому что в нём только одна единица (нечётное количество).

2. Проверяем следующее по порядку число $65_{10} = 1000001_2$.

Может ли крайний справа бит (1) быть дописанным к двоичному числу 100000? Да, может.

А может ли тогда перед этим быть дописан справа бит 0 к числу 10000? Нет, не может.

3. Проверяем следующее число $66_{10} = 1000010_2$.

Может ли крайний справа бит (0) быть дописанным к двоичному числу 100001? Да, может.

А может ли перед этим быть дописан справа бит 1 к числу 10000? Да, может.

Следовательно, число 66 удовлетворяет условию задачи.

Ответ: 66.

Задача 4. На вход алгоритма подаётся натуральное число N. По нему строится новое число R по следующим правилам.

1. Строится двоичная запись числа N.

2. Если полученное число — чётное, то справа к нему дописываются два нулевых бита. Если полученное число — нечётное, то справа дописываются два единичных бита.

Полученная таким способом запись —это двоичная запись числа R.

Определить **наименьшее** число R, большее 118, которое может быть получено в результате работы такого алгоритма. В ответе число R запишите в десятичной системе.

Решение

1) Берём за основу заданное число 118. Переводим его в двоичную систему счисления, получаем число 1110110.

2) Отсекаем два последних разряда, получаем число 11101. Оно нечётное (последняя цифра — 1), поэтому алгоритм допишет к нему две единицы. Получим: 1110111 (больше, чем для исходного числа). Переводим полученное число в десятичную систему счисления. Получаем число 119.

Ответ: 119.

Исполнители: Робот, Чертёжник, Редактор

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:

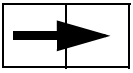
- перемещение: вверх, вниз, влево, вправо;

- проверка наличия препятствия (стенки): сверху свободно, снизу свободно, слева свободно, справа свободно — играют роль условия;
- команда ветвления:

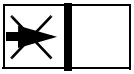
```
ЕСЛИ <условие>
          ТО команда1
          ИНАЧЕ команда2
КОНЕЦ ЕСЛИ
```

— 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`



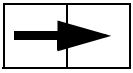
Стенки нет — Робот перемещается в соседнюю клетку



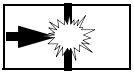
Стенка есть — Робот остаётся на месте

Проверка наличия препятствия выполняется ***до*** начала движения Робота.

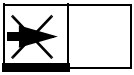
- При несовпадении *направления проверки наличия препятствия совпадает с направлением движения Робота*: `ЕСЛИ <снизу свободно> ТО вправо`



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.

Исполнитель Чертёжник — перемещается по координатной плоскости. Его основная команда — **сместиться на (*a*, *b*)**, где a, b — некоторые целые числа, она перемещает Чертёжника из исходной точки с координатами (x, y) в точку с координатами $(x + a, y + b)$.

Цикл

```
ПОВТОРИ число РАЗ
  последовательность команд
КОНЕЦ ПОВТОРИ
```

означает, что заданная последовательность команд выполняется указанное количество раз (значение должно быть натуральным).

Исполнитель Редактор — обрабатывает текстовые строки. Он получает на вход некоторую текстовую строку и преобразует её по заданным правилам.

Исполнитель Редактор может выполнять две команды (в них обозначения *v* и *w* обозначают некоторые последовательности — цепочки символов):

- **заменить** (*v*, *w*) — заменяет в строке первое встреченное при просмотре слева направо вхождение цепочки символов *v* на цепочку *w*; если же цепочка символов *v* в строке не найдена, выполнение данной команды не меняет строку;
- **нашлось** (*v*) — проверяет, встречается ли цепочка символов *v* в обрабатываемой текстовой строке, и возвращает логическое значение «истина» или «ложь»; сама текстовая строка при этом не изменяется.

Собственно алгоритм обработки текстовой строки реализуется при помощи конструкций цикла ПОКА и ветвления (ЕСЛИ ... ТО ... ИНАЧЕ).

Разбор типовых задач

Задача 1. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости, состоит из 8 команд. Четыре команды — это команды-приказы: **вверх**, **вниз**, **влево**, **вправо**. При выполнении любой из этих команд РОБОТ перемещается на одну клетку, соответственно, вверх, вниз, влево или вправо. Четыре другие 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
```

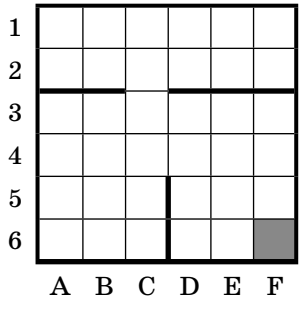
выполняется *команда1* (если условие истинно) или *команда2* (если условие ложно).

В конструкциях ПОКА и ЕСЛИ условие может содержать команды проверки, а также слова 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 этой клетке и выполнив предложенную программу, РОБОТ уцелеет и остановится в закрашенной клетке (клетка F6)?

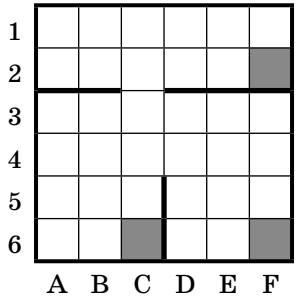
```
НАЧАЛО
  ПОКА снизу свободно ИЛИ справа свободно
    ЕСЛИ снизу свободно
      ТО вниз
      ИНАЧЕ вправо
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```



Решение

Анализируем предложенную программу РОБОТа.

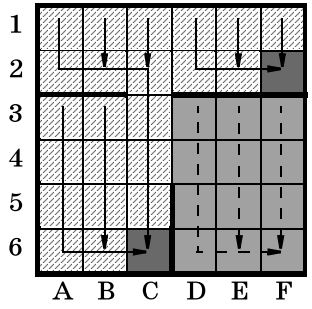
- В начале очередного прохода цикла проверяется: есть ли стенки снизу и справа. Если хотя бы одно из этих направлений свободно, то выполняется очередной проход цикла. Значит, точка остановки (клетка, в которой выполнение цикла корректно прекратится) имеет вид: ⌟. Причём «правильной» является только одна такая клетка, а остальные две (пометим их пунктиром) — это «ловушки» для РОБОТа.



- Когда начинается очередной проход цикла, проверяется наличие стенки снизу. Если её нет, то выполняется перемещение на 1 клетку вниз. Если же стенка снизу есть, то выполняется перемещение на 1 клетку вправо, причём *без проверки наличия стенки справа*. Следовательно, все клетки, в которых есть стенки снизу и есть стенки справа, можно было бы сразу пометить как непригодные (начав движение из них, РОБОТ разобьётся). Но это не требуется: ведь такое условие записано в цикле ПОКА, который контролирует каждый отдельный шаг РОБОТа. (Вот если бы в цикле ПОКА было записано два или больше таких оператора ЕСЛИ, то для второго и т.д. такого оператора пришлось бы это исключение клеток сделать.)

- После выполнения одного такого хода проход цикла завершается, и вновь выполняется проверка его условия. Нетрудно понять, что в этом случае движение РОБОТа будет следующим: «идти вниз, пока это возможно (до стенки снизу), а наткнувшись на стенку снизу — пытаться уходить вправо, отыскивая проход вниз в этой стенке».

Именно так мы и будем пытаться «трассировать» движение РОБОТа, «честно» выполняя предложенную программу. Увидев же, куда в итоге придёт РОБОТ, помечаем все пройденные клетки штриховкой (если попадаем в «ловушку») или закрашиваем (если доходим до клетки F6). Сама же клетка F6 нам тоже подходит: если РОБОТ изначально находится в ней, то он просто никуда не пойдёт и сразу же выполнит условие задачи.



Остаётся только подсчитать количество клеточек, закрашенных светло-серым: их 12.

Ответ: 12.

Задача 2. Исполнитель Чертёжник перемещается по координатной плоскости. Основная команда Чертёжника — **сместиться на (*a*, *b*)**, где a, b — целые числа. Она перемещает Чертёжника из точки с координатами (x, y) в точку с координатами $(x + a, y + b)$. Например, из точки с координатами (3, 5) команда **сместиться на (−2, 1)** переместит Чертёжника в точку (1, 6).

Цикл

```
ПОВТОРИ число РАЗ
  последовательность команд
КОНЕЦ ПОВТОРИ
```

означает, что заданная последовательность команд будет выполнена указанное количество раз (значение должно быть натуральным).

Чертёжник выполнял алгоритм, в котором количество повторений и оба смещения в первой из повторяемых команд неизвестны:

```
НАЧАЛО
  сместиться на (-2, 3)
ПОВТОРИ … РАЗ
  сместиться на (…, …)
  сместиться на (-2, -1)
КОНЕЦ ПОВТОРИ
  сместиться на (-19, -18)
КОНЕЦ
```

При этом, выполнив этот алгоритм, Чертёжник вернулся в исходную точку. Какое наибольшее количество повторений могло быть указано в конструкции «ПОВТОРИ … РАЗ»?

Решение (*способ 1*)

Помните, как на физике решали задачу о движении тела, брошенного под заданным углом к горизонту («задача о пушечном выстреле»)? Тогда мы составляли два отдельных уравнения движения по координате x и координате y и рассматривали их в виде системы из двух уравнений. В нынешней задаче нужно сделать то же самое, обозначив неизвестные нам значения переменными: k — количество повторений, x и y — неизвестные смещения в первой повторяемой команде. При этом каждое уравнение приравнивается нулю, так как Чертёжник после выполнения всех перемещений вернулся в исходную точку.

1) Перемещение Чертёжника по координате x:

$-2 + k \cdot (x - 2) - 19 = 0$.

2) Перемещение Чертёжника по координате y:

$3 + k \cdot (y - 1) - 18 = 0$.

3) Объединяем оба уравнения в систему:

$$\begin{cases} -2 + k \cdot (x - 2) - 19 = 0; \\ 3 + k \cdot (y - 1) - 18 = 0; \end{cases} \qquad \begin{cases} k \cdot (x - 2) = 21; \\ k \cdot (y - 1) = 15. \end{cases}$$

4) На первый взгляд данная система нерешаема: два уравнения и три неизвестных. Но обратим внимание — в обоих случаях слева от знака равенства стоят произведения, один из сомножителей в которых один и тот же (k).

Разложим числа справа от знаков равенства на простые сомножители (благо это можно сделать одним-единственным способом): $21 = 3 \cdot 7$; $15 = 3 \cdot 5$.

Заметим: в обоих получившихся произведениях тоже повторяется один и тот же сомножитель — 3. Значит, это и есть k.

Определив, что $k = 3$, можно при желании (или при необходимости, если это потребуют в условии задачи) вычислить и значения x и y. Нам же требуется только это значение k.

Решение (*способ 2*)

Можно упростить и ускорить решение, исключив составление уравнений. Для этого нужно, не обращая внимания на команды внутри цикла ПОВТОРИ, отдельно сложить только пары чисел — значений координат в командах перемещений до и после цикла:

```
сместиться на (-2, 3)
…
сместиться на (-19, -18)
```

Получаем:

- по X: $-2 + (-19) = -21$;
- по Y: $3 + (-18) = -15$.

Полученные значения сумм берутся по модулю — получаем числа 21 и 15.

Их можно разложить на простые множители так:

$21 = 3 \cdot 7$,

$15 = 3 \cdot 5$.

Повторяется в них число 3. Это и есть значение количества повторений цикла.

Ответ: 3.

Задача 3. Исполнитель Чертёжник выполнял алгоритм, в котором количество повторений и оба смещения в первой из повторяемых команд неизвестны:

```
НАЧАЛО
  сместиться на (24, -28)
ПОВТОРИ … РАЗ
  сместиться на (…, …)
  сместиться на (-22, 14)
КОНЕЦ ПОВТОРИ
  сместиться на (12, 20)
КОНЕЦ
```

Чертёжник начинал выполнение алгоритма из начальной точки с координатами (−10, 7), а завершил работу в точке с координатами (−37, 35). Какое наибольшее количество повторений могло быть указано в конструкции «ПОВТОРИ … РАЗ»?

Решение

Отличие от предыдущей задачи в том, что ранее Чертёжник начинал и заканчивал работу в одной и той же исходной точке, а теперь это разные точки. Но кто мешает нам изменить программу исполнителя так, чтобы он начинал работу в исходной точке с нулевыми координатами, затем перемещался в требуемую исходную точку, а после выполнения всех заданных перемещений дополнительно переходил снова в исходную точку с координатами (0, 0)?

Для этого добавляем в начало программы (сразу после слова НАЧАЛО) команду «сместиться на (−10, 7)», а в конце программы (перед словом КОНЕЦ) — команду «сместиться на (37, −35)» (здесь координаты конечной точки берутся с обратным знаком, так как мы возвращаемся из неё «в нули»).

Дальнейшее решение — точно такое же, как в предыдущей задаче, но помним, что команд смещения до и после цикла у нас уже по две. Можно полностью расписать все перемещения по осям x и y (способ 1) либо учесть в вычислениях только значения в командах смещения вне цикла (способ 2):

— по оси x: $-10 + 24 + 12 + 37 = 63$,

— по оси y: $7 - 28 + 20 - 35 = -36$.

Разложив взятые по модулю оба полученных числа на простые множители ($63 = 3 \cdot 3 \cdot 7$, $36 = 3 \cdot 3 \cdot 4$), ищем в них повторяющиеся сомножители. В данном случае это два числа 3 (или просто 9).

По условию, мы ищем наибольшее количество повторений — следовательно, повторений было $3 \cdot 3 = 9$.

Ответ: 9

Если бы в условии требовалось найти наименьшее количество повторений, то нужно было бы брать только одно, наименьшее из возможных, значение повторяющегося сомножителя. Тогда ответ в этой задаче был бы равен 3.

Задача 4. Исполнитель Редактор получает на вход строку цифр и преобразовывает её. Редактор может выполнять две команды: **заменить** (v, w) и **нашлось** (v).

Дана программа для исполнителя Редактор:

```
НАЧАЛО
ПОКА нашлось (111) ИЛИ нашлось (999)
  ЕСЛИ нашлось (111)
    ТО заменить (111, 9)
    ИНАЧЕ заменить (999, 1)
  КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
КОНЕЦ
```

Какая строка получится из строки, состоящей из 74 идущих подряд цифр «9»?

Решение

1. Начинаем анализировать алгоритм Редактора.

Вначале имеется строка из 74 цифр «9». Поэтому условие ПОКА истинно (три подряд девятки в этой строке есть). А вот условие ЕСЛИ — ложно (единиц в строке нет). Поэтому выполняется ветвь программы ИНАЧЕ и производится замена первых трёх девяток на одну-единственную единицу:

$$\underbrace{999999999\ldots9999}_{\text{74 цифры «9»}} \rightarrow \underbrace{1999999\ldots9999}_{\text{71 цифра «9»}}$$

2. На втором проходе цикла условие ПОКА всё ещё истинно (после полученной единицы имеется три девятки подряд). Условие ЕСЛИ по-прежнему ложно (трёх единиц подряд в строке пока нет). Поэтому вновь выполняется ветвь программы ИНАЧЕ и производится замена очередных трёх девяток на одну единицу:

$$\underbrace{1999999\ldots9999}_{\text{71 цифра «9»}} \rightarrow \underbrace{11999\ldots9999}_{\text{68 цифр «9»}}$$

3. На третьем проходе цикла условие ПОКА по-прежнему истинно (после двух единиц есть девятки подряд). Условие же ЕСЛИ всё ещё ложно (трёх единиц подряд в строке нет). Поэтому выполняется ветвь программы ИНАЧЕ и снова производится замена очередных трёх девяток на одну единицу:

$$\underbrace{1199999\ldots9999}_{\text{68 цифр «9»}} \rightarrow \underbrace{11199\ldots9999}_{\text{65 цифр «9»}}$$

4. На четвёртом проходе цикла условие ПОКА тоже истинно (хотя бы по первым трём единицам, образовавшимся в начале строки). А вот условие ЕСЛИ теперь истинно — у нас в начале строки имеется три подряд единицы. Поэтому теперь выполняется уже ветвь

программы **ТО** и производится замена этих трёх единиц снова на девятку — но только на одну:

$$\underbrace{11199\ldots9999}_{\text{65 цифр «9»}} \rightarrow \underbrace{999\ldots9999}_{\text{новая цифра «9» + 65 цифр «9»}}$$

5. Делаем следующий вывод: в работе программы для исполнителя Редактор можно выделить некий «суперцикл» — каждые четыре шага цикла ПОКА приводят к тому, что исходная строка из девяток просто уменьшает свою длину на 8 символов (было 74 девятки, а стало 66 девяток, включая только что добавленную вместо единиц). Поэтому дальнейшую работу программы можно подробно не рассматривать, а просто «отсекать» от строки по 8 девяток, пока не останется «обрубок» строки длиной 8 цифр или меньше. Поэтому начинаем вычитать из количества девяток число 8 до тех пор, пока их не останется восемь или меньше: 66 → 58 → → 50 → 42 → 34 → 26 → 18 → 10 → 2.

6. А теперь снова подробно анализируем, что будет происходить с этим фрагментом. В нашем случае всё достаточно просто: у нас осталась строчка из двух девяток («99»), для которой условие ПОКА ложно, и выполнение программы прекращается. А если получившаяся строка девяток имеет длину 3 цифры или больше, то потребуется подробно рассмотреть, какие преобразования происходят в этом фрагменте строки.

Вот, например, каким было бы окончание решения, если бы в строке осталось 5 девяток:

1) первые 3 девятки заменяются на одну единицу:

$$99999 \rightarrow 199$$

2) теперь в полученной строке больше нет ни трёх подряд единиц, ни трёх подряд девяток, следовательно, условие ПОКА становится ложным, и выполнение программы прекращается. А в результате остаётся строка «199».

Ответ: 99.

Задача 5. Исполнитель Редактор получает на вход строку цифр и преобразовывает её. Редактор может выполнять две команды: **заменить** (*v*, *w*) и **нашлось** (*v*).

Дана программа для исполнителя Редактор:

```
НАЧАЛО
ПОКА нашлось (111) ИЛИ нашлось (999)
  ЕСЛИ нашлось (999)
    ТО заменить (999, 1)
    ИНАЧЕ заменить (111, 9)
  КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
КОНЕЦ
```

Какая строка получится из строки, состоящей из 74 идущих подряд цифр «9»?

Единственное отличие этой задачи от предыдущей — другая запись условия ЕСЛИ: в предыдущей задаче в этом условии выполнялся поиск цифр, которых не было в исходной строке, а в данной задаче — поиск цифр, из которых и состоит исходная строка. Поэтому ход решения хотя и будет аналогичен в обеих задачах, но совершенно различен — как и получаемый ответ.

Решение

1. Начинаем анализировать алгоритм Редактора.

Вначале имеется строка из 74 цифр «9». Поэтому условие ПОКА истинно (три подряд девятки в этой строке есть). Условие ЕСЛИ тоже истинно, поэтому выполняется ветвь программы ТО и производится замена первых трёх девяток на одну-единственную единицу:

$$\underbrace{999999999\ldots9999}_{\text{74 цифры «9»}} \rightarrow \underbrace{1999999\ldots9999}_{\text{71 цифра «9»}}$$

2. На втором проходе цикла условие ПОКА истинно (после полученной единицы имеется три девятки подряд). Условие ЕСЛИ по-прежнему истинно. Поэтому вновь выполняется ветвь программы ТО и производится замена очередных трёх девяток на одну единицу:

$$\underbrace{1999999\ldots9999}_{\text{71 цифра «9»}} \rightarrow \underbrace{11999\ldots9999}_{\text{68 цифр «9»}}$$

3. На третьем проходе цикла условие ПОКА истинно (после двух единиц есть девятки подряд). Условие ЕСЛИ тоже всё ещё истинно. Поэтому выполняется ветвь программы ТО и снова производится замена очередных трёх девяток на одну единицу:

$$\underbrace{1199999\ldots9999}_{\text{68 цифр «9»}} \rightarrow \underbrace{11199\ldots9999}_{\text{65 цифр «9»}}$$

4. На четвёртом проходе цикла условие ПОКА тоже истинно. А что условие ЕСЛИ? А оно по-прежнему истинно — ведь мы ищем триады девяток, а не единицы! Поэтому по-прежнему выполняется ветвь программы ТО и производится замена очередных трёх девяток на единицу:

$$\underbrace{11199\ldots9999}_{\text{65 цифр «9»}} \rightarrow \underbrace{111199\ldots9999}_{\text{62 цифр «9»}}$$

5. Учитывая это, можно сделать вывод: при работе данной программы исполнитель Редактор просто заменяет все встреченные тройки девяток на единицы. Это будет происходить до тех пор, пока в строке ещё остаются триады девяток.

Какой станет в результате строка? Очевидно, в её начале будет столько единиц, сколько в исходной строке имелось целых триад, и это количество можно определить путём целочисленного деления: 74 DIV 3 = 24. А после них останутся «невостребованными» девятки, количество которых меньше трёх. Сколько их, можно вычислить как остаток от деления исходного количества девяток на три: 74 MOD 3 = 2. Следовательно, строка будет такой:

$$\underbrace{11111\ldots111199}_{\text{24 цифры «1»}}$$

6. Что будет дальше? Очевидно, условие ПОКА по-прежнему истинно: если нет трёх девяток подряд, то три единицы подряд в строке имеются. А вот условие ЕСЛИ по-прежнему ложно (девяток только две), поэтому на очередном проходе цикла будет выполнена ветвь ИНАЧЕ. В ней выполняется замена первых имеющихся трёх единиц на одну девятку:

$$\underbrace{1111111\ldots1199}_{\text{24 цифры «1»}} \rightarrow \underbrace{91111\ldots1199}_{\text{21 цифра «1»}}$$

7. Условие ПОКА по-прежнему истинно. Условие ЕСЛИ тоже по-прежнему ложно (трёх девяток подряд нет), поэтому опять будет выполнена ветвь ИНАЧЕ, в которой выполняется замена очередных имеющихся трёх единиц на одну девятку:

$$\underbrace{91111\ldots1199}_{\text{21 цифра «1»}} \rightarrow \underbrace{991\ldots1199}_{\text{18 цифр «1»}}$$

8. Условие ПОКА истинно. Условие ЕСЛИ ложно. Поэтому вновь будет выполнена ветвь ИНАЧЕ, в которой выполняется замена очередных трёх единиц на одну девятку:

$$\underbrace{991111\ldots1199}_{\text{18 цифр «1»}} \rightarrow \underbrace{9991\ldots1199}_{\text{15 цифр «1»}}$$

9. Условие ПОКА истинно. А вот условие ЕСЛИ теперь опять становится истинным — ведь в начале строки уже снова появились три девятки. Поэтому выполняется ветвь ТО и выполняется замена только что получившихся трёх девяток на одну единицу:

$$\underbrace{9991\ldots1199}_{\text{15 цифр «1»}} \rightarrow \underbrace{11\ldots1199}_{\text{новая цифра «1» + 15 цифр «1»}}$$

10. Как и в предыдущей задаче, мы видим, что через каждые четыре шага цикла ПОКА полученная «проме-

жуточная» строка из 24 единиц и оставшихся в её конце двух девяток уменьшает длину своей цепочки единиц на 8 символов (было 24 единички, а стало 16 единичек, включая только что добавленную вместо трёх девяток). Поэтому начинаем вычитать из количества единичек число 8 до тех пор, пока их не останется восемь или меньше: 16 → 8.

11. А теперь снова начинаем подробно смотреть, что происходит с нашей строкой, которая содержит 8 единичек и две девятки:

1)

1111111199 → **9**1111199

2)

91111199 → **99**1199

Обратим внимание: теперь в полученной строке больше нет ни трёх подряд единиц, ни трёх подряд девяток, следовательно, условие ПОКА становится ложным и выполнение программы прекращается. А в результате остаётся строка «991199».

На этом последнем этапе решения задачи нужно быть особенно внимательным — именно поэтому мы «вручную» вычитали восьмёрки до момента, когда единиц в строке осталось 8 или меньше, а не вычисляли остаток от деления. В зависимости от количества цифр в строке финал решения может быть различным:

- в начале строки может образоваться менее трёх девяток, а в середине — остаться менее трёх единиц (как это и произошло в нашем случае);
- преобразование единиц в девятки может оказаться полным — при этом девятки, получившиеся в начале строки, «соединятся» с девятками, которые всё время оставались в её конце, так что в строке снова окажется три подряд девятки и нужно будет продолжить выполнение программы до тех пор, пока условие ПОКА не станет ложным.

Ответ: 991199.

Ещё раз обратим внимание: алгоритм работы Редактора различен для двух возможных вариантов его программы — ищутся ли в условии ЕСЛИ те же цифры, что уже есть в исходной строке, или цифры, которые предполагается в ней заменять. И в зависимости от этого нужно выбирать один из двух соответствующих способов решения.

Задача 6. Исполнитель Редактор получает на вход строку цифр и преобразовывает её. Редактор может выполнять две команды: **заменить** (*v*, *w*) и **нашлось** (*v*).

Дана программа для исполнителя Редактор:

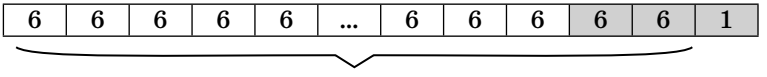
```
НАЧАЛО
ПОКА нашлось (155) ИЛИ нашлось (661)
  ЕСЛИ нашлось (155)
    ТО заменить (155, 61)
    ИНАЧЕ заменить (661, 15)
  КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
КОНЕЦ
```

На вход этой программе подаётся строка, состоящая из 100 цифр; последняя цифра в строке — цифра 1, а остальные цифры — шестёрки. Какая строка получится в результате применения программы к этой строке?

В отличие от предыдущих задач указанные в программе для исполнителя наборы символов теперь состоят не из одинаковых цифр.

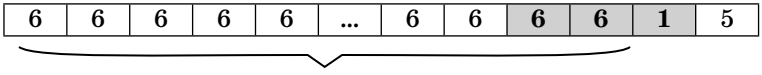
Решение

1) Исходная строка имеет вид:



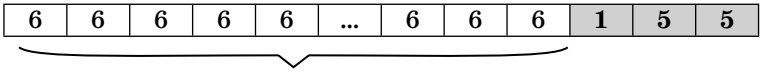
Очевидно, в ней имеется одна комбинация «661» (выделена серым), поэтому условие в операторе ПОКА истинно.

Однако условие в операторе ЕСЛИ ложно, поэтому выполняется ветвь ИНАЧЕ. В результате указанная комбинация будет заменена на «15», а строка примет вид:



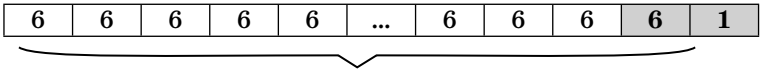
97 шестёрок

2) На втором проходе цикла (условие ПОКА по-прежнему истинно) изменение строки производится аналогично:



95 шестёрок

3) На третьем проходе цикла условие ЕСЛИ оказывается истинным — в конце строки появилась нужная комбинация «155». Поэтому выполняется ветвь ТО, а строка получает вид:

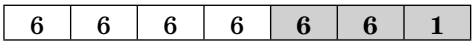


96 шестёрок

4) Итак, после трёх шагов цикла ПОКА строка снова стала аналогичной исходной (все шестёрки и последняя единица), но её длина сократилась на три шестёрки (99 – 96 = 3).

Делаем вывод: в ходе работы программы исполнителя строка и далее будет укорачиваться — каждые три прохода цикла на три шестёрки: 96 → 93 → 90 →...

Нетрудно догадаться, что количество шестёрок на каждом таком «промежуточном» этапе кратно 3. Проследим теперь, что происходит со строкой, когда её длина сократится до, например, семи знаков (шесть шестёрок плюс единица в конце):



5) Условие ПОКА истинно, условие ЕСЛИ ложно — выполняется ветвь ИНАЧЕ:

6	6	**6**	**6**	**1**	5

6) Условие ПОКА истинно, условие ЕСЛИ ложно — выполняется ветвь ИНАЧЕ:

6	6	**1**	**5**	**5**

7) Условие ПОКА истинно, условие ЕСЛИ истинно — выполняется ветвь ТО:

6	**6**	**6**	**1**

8) Условие ПОКА истинно, условие ЕСЛИ ложно — выполняется ветвь ИНАЧЕ:

6	1	5

Теперь условие ПОКА стало ложным, и выполнение программы прекращается. Оставшаяся строка и является ответом.

Ответ: 615.

В любом случае в подобной задаче мы начинаем анализировать изменения, происходящие в строке начиная с её исходного состояния, пока строка на каком-то шаге цикла не примет вид, аналогичный исходному (но став при этом короче на сколько-то цифр).

Далее мы «шагами» уменьшаем длину строки на вычисленное количество цифр, получив строку небольшой длины (но с некоторым «запасом», например, в только что рассмотренной задаче берём строку из 6 шестёрок и единицы, а не из 3 шестёрок и единицы).

И наконец завершаем решение, отрабатывая заданный алгоритм применительно к полученной короткой строке.

Задача 7. Исполнитель Редактор получает на вход строку цифр и преобразует её.

Редактор может выполнять две команды, в обеих командах v и w обозначают цепочки цифр.

А) **заменить (*v*, *w*)** — заменяет в строке первое слева вхождение цепочки *v* на цепочку *w*. Если в строке нет вхождений цепочки *v*, то выполнение команды **заменить (*v*, *w*)** не меняет эту строку.

Б) **нашлось (*v*)** — проверяет, встречается ли цепочка *v* в строке исполнителя Редактор. Если она встречается, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Сама строка исполнителя при этом не изменяется.

Исполнителю на вход подаётся строка, состоящая из семёрки и 90 записанных вслед за ней нулей.

Сколько нулей будет в строке, полученной в результате работы исполнителя? В качестве ответа нужно записать только число.

```
НАЧАЛО
  ПОКА нашлось (70) ИЛИ нашлось (7)
    ЕСЛИ нашлось (70) ТО
      заменить (70,0007)
    ИНАЧЕ
      ЕСЛИ нашлось (7) ТО
        заменить (7, 00)
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

Решение

Важное отличие от предыдущих задач с Редактором состоит в том, что при работе исполнителя длина строки не сокращается, а, наоборот, увеличивается (поскольку две цифры заменяются на четыре, а одна — на две). Поэтому решение будет немного иным, но сам его принцип остаётся прежним: начать выполнять алгоритм, а затем, проследив, что происходит со строкой, попытаться вывести общую закономерность.

Исходная строка:

$$7\underbrace{0000\ldots0000}_{\text{90 нулей}}$$

1) В строке есть пара цифр «70», следовательно, цикл ПОКА работает.

Первый же оператор ЕСЛИ срабатывает, так как комбинация «70» в строке имеется. Выполняется её замена на «0007». Получаем строку:

$$0007\underbrace{0000\ldots0000}_{\text{89 нулей}}$$

Ветвь ИНАЧЕ, соответственно, пропускается.

2) В строке по-прежнему есть пара цифр «70», поэтому цикл продолжает работу.

Теперь в строке заменяется очередная комбинация «70» на «0007»:

$$0000007\underbrace{0000\ldots0000}_{\text{88 нулей}}$$

Уже сейчас мы видим закономерность: количество нулей справа от семёрки каждый раз уменьшается на 1, но зато слева от семёрки каждый раз добавляется три нуля. То есть при каждом проходе цикла в строке один ноль справа заменяется на три нуля слева, и это будет происходить до тех пор, пока справа от семёрки есть хотя бы один ноль.

Очевидно, тем самым мы «меняем» каждый «правый» ноль на три «левых», а значит, по завершении этих действий количество нулей в строке будет утроено, а в самом конце строки (крайней справа) окажется цифра 7:

$$\underbrace{000\ldots0007}_{90 \times 3 = 270 \text{ нулей}}$$

3) Что происходит дальше?

В строке уже больше нет пары цифр «70», но есть отдельная семёрка, поэтому цикл ПОКА все ещё работает.

Первый оператор ЕСЛИ, очевидно, пропустит ветвь ТО и запустит на исполнение ветвь ИНАЧЕ (поскольку

комбинация «70» в строке отсутствует). Второй оператор ЕСЛИ, размещённый в ветви ИНАЧЕ первого, очевидно, запустит свою ветвь ТО и тем самым запустит на исполнение команду, которая ищет семёрку и заменяет её на пару нулей. В результате мы получим строку, которая состоит из 270 нулей в начале и ещё двух нулей в конце (вместо бывшей там семёрки). Всего — 272 нуля.

4) поскольку в полученной строке больше нет ни пары «70», ни даже отдельной семёрки, работа цикла ПОКА будет прекращена. А в полученной строке будет 272 нуля.

Ответ: 272.

Числовые 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:

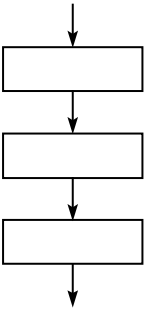
— **среда** — условия, в которых функционирует исполнитель (например, исполнитель Черепашка имеет определённую систему координат, исполнитель Робот перемещается по клетчатому полю и т.д.);

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

Алгоритм — запись в той или иной форме (словесной, графической, на языке программирования) последовательности команд для исполнителя. Команды алгоритма должны соответствовать системе команд исполнителя.

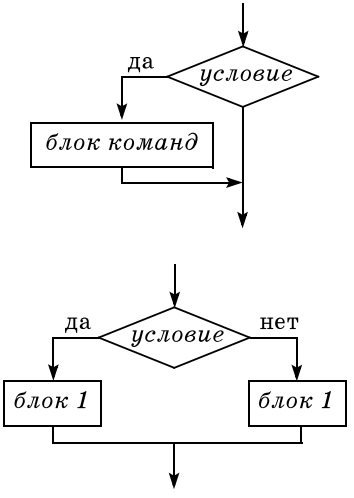
Алгоритмические конструкции

1. **Следование** — последовательное выполнение команд сверху вниз (линейный алгоритм).



алг программа	Название алгоритма
нач	Начало алгоритма
вещ SR **цел** A, B, C, N	Объявление переменных (**цел** — целых, **вещ** — вещественных)
A = 1	Расчётная часть (операции выполняются поочерёдно сверху вниз)
B = 2	
C = 3	
N = 3	
SR = (A + B + C)/N	
кон	Конец алгоритма

2. **Ветвление** — команды выполняются в зависимости от истинности некоторого условия либо возможно выполнение одного из двух или более наборов команд в зависимости от истинности условия.

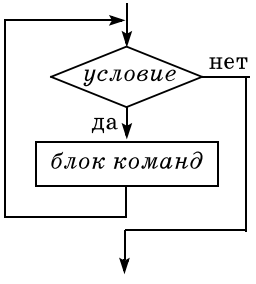


алг программа	Название алгоритма
нач	Начало алгоритма
цел A, B, C	Объявление переменных
A = 1	Линейная часть алгоритма
B = 2	
если A < B	Условие ветвления
то C = A	Команда (или команды), выполняемая, если условие истинно
иначе C = B	Команда (или команды), выполняемая, если условие ложно
кон	Конец алгоритма

3. Цикл — многократное повторение блока команд (количество повторений зависит от условия цикла либо задаётся явно).

Цикл с предусловием
(*цикл ПОКА*)

Блок команд выполняется, пока условие остаётся истинным (выход из цикла — по ложности условия).

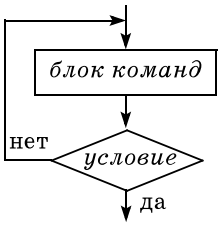


алг программа	Название алгоритма
нач	Начало алгоритма
цел A, B	Объявление переменных
A = 0 ввод(B)	Линейная часть алгоритма
нц пока B <> 0	Условие выполнения цикла
A = A + B ввод(B)	Команды, выполняемые, пока условие истинно
кц	Конец блока команд, составляющих цикл
вывод(A)	Команда, которая выполняется после завершения цикла (когда условие цикла станет ложным)
кон	Конец алгоритма

Условие проверяется *до* начала выполнения цикла, поэтому возможна ситуация, что тело цикла не будет выполнено ни разу.

Цикл с постусловием
(*цикл ДО*)

Блок команд выполняется до тех пор, пока условие не станет истинным (цикл выполняется, пока условие ложно).

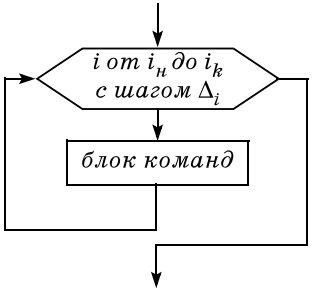


алг программа	Название алгоритма
нач	Начало алгоритма
цел A, B	Объявление переменных
A = 0	Линейная часть алгоритма
нц	Начало блока команд, составляющих цикл
ввод(B) A = A + B	Команды, выполняемые, пока условие цикла ложно
кц до B = 0	Конец блока команд, составляющих цикл
вывод(A)	Команда, которая выполняется после завершения цикла (когда условие цикла станет истинным)
кон	Конец алгоритма

Условие проверяется *после* выполнения цикла, поэтому тело цикла всегда выполняется хотя бы один раз.

Цикл с параметром
(*цикл со счётчиком, цикл ДЛЯ*)

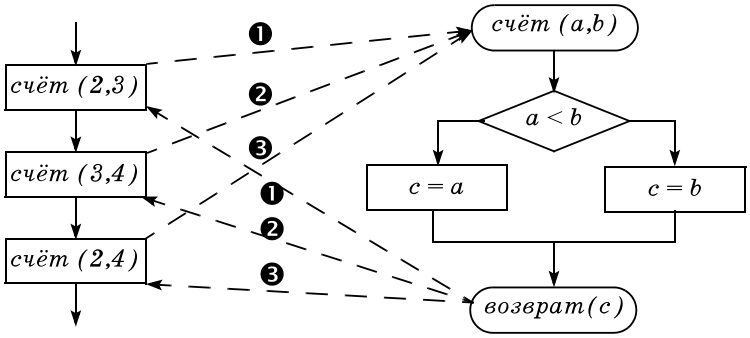
Блок команд выполняется количество раз, заданное начальным, конечным значениями переменной-счётчика и шагом её изменения.



алг программа	Название алгоритма
нач	Начало алгоритма
цел A, i	Объявление переменных
A = 0	Линейная часть алгоритма
нц для i от 1 до 10 шаг 2	Начало цикла, задание его параметров
A = A + I	Команды, выполняемые в цикле
кц	Конец цикла
вывод(A)	Команда, которая выполняется после завершения цикла
кон	Конец алгоритма

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 D_i 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.

4. **Подпрограмма** — поименованный блок команд выполняется после их вызова из любого места основной программы. При этом одна и та же подпрограмма может быть вызвана из основной программы любое количество раз и ей каждый раз могут передаваться свои значения переменных (*параметров*).



Разбор типовых задач

Задача 1. 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

Для решения можно построить полное дерево вариантов для получения числа 1 из числа 29 (не «обрубая» повторяющиеся ветви!) и подсчитать количество возможных 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 2. 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*}.

$R(39) = R(39/3) + R(39 - 6) = \sout{R(13)} + R(33) = 2$.

$R(45) = R(45/3) + R(45 - 6) = R(15) + R(39) = 1 + 2 = 3$.

$R(51) = R(51/3) + R(51 - 6) = \sout{R(17)} + R(45) = 3$.

$R(57) = R(57/3) + R(57 - 6) = \sout{R(19)} + R(51) = 3$.

$R(63) = R(63/3) + R(63 - 6) = R(21) + R(57) = 1 + 3 = 4$.

$R(69) = R(69/3) + R(69 - 6) = \sout{R(23)} + R(63) = 4$.

$R(75) = R(75/3) + R(75 - 6) = \sout{R(25)} + R(69) = 4$.

$R(81) = R(81/3) + R(81 - 6) = R(27) + R(75) = 2 + 4 = 6.$
$R(87) = R(87/3) + R(81 - 6) = \sout{R(29)} + R(81) = 6.$

Ответ: 6 программ.

Задача 3. 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»*}.

$R(4 + 2) = R(6) = R(\sqrt{6}) + R(6 - 2) = \sout{R(\sqrt{6})} + R(4) =$
$= R(4) = 1$ {*очередную строку записываем для значения n, увеличенного на 2; слагаемые, где квадратный корень извлекается не нацело, исключаются; для получаемого значения R(n) берётся ранее вычисленное значение; если значение R(n) не существует, то оно исключается*}.

$R(8) = R(\sqrt{8}) + R(8 - 2) = \sout{R(\sqrt{8})} + R(6) = R(6) = 1.$
$R(10) = R(\sqrt{10}) + R(10 - 2) = \sout{R(\sqrt{10})} + R(8) = R(8) = 1.$
$R(12) = R(\sqrt{12}) + R(12 - 2) = \sout{R(\sqrt{12})} + R(10) = R(10) = 1.$
$R(14) = R(\sqrt{14}) + R(14 - 2) = \sout{R(\sqrt{14})} + R(12) = R(12) = 1.$
$R(16) = R(\sqrt{16}) + R(16 - 2) = R(4) + R(14) = 1 + 1 = 2.$

$R(18) = R(\sqrt{18}) + R(18-2) = \cancel{R(\sqrt{18})} + R(16) = R(16) = 2.$
$R(20) = R(\sqrt{20}) + R(20-2) = \cancel{R(\sqrt{20})} + R(18) = R(18) = 2.$
$R(22) = R(\sqrt{22}) + R(22-2) = \cancel{R(\sqrt{22})} + R(20) = R(20) = 2.$
$R(24) = R(\sqrt{24}) + R(24-2) = \cancel{R(\sqrt{24})} + R(22) = R(22) = 2.$
$R(26) = R(\sqrt{26}) + R(26-2) = \cancel{R(\sqrt{26})} + R(24) = R(24) = 2.$
$R(28) = R(\sqrt{28}) + R(28-2) = \cancel{R(\sqrt{28})} + R(26) = R(26) = 2.$
$R(30) = R(\sqrt{30}) + R(30-2) = \cancel{R(\sqrt{30})} + R(28) = R(28) = 2.$
$R(32) = R(\sqrt{32}) + R(32-2) = \cancel{R(\sqrt{32})} + R(30) = R(30) = 2.$
$R(34) = R(\sqrt{34}) + R(34-2) = \cancel{R(\sqrt{34})} + R(32) = R(32) = 2.$
$R(36) = R(\sqrt{36}) + R(36-2) = R(6) + R(34) = 1 + 2 = 3.$
$R(38) = R(\sqrt{38}) + R(38-2) = \cancel{R(\sqrt{38})} + R(36) = R(36) = 3.$
$R(40) = R(\sqrt{40}) + R(40-2) = \cancel{R(\sqrt{40})} + R(38) = R(38) = 3.$
$R(42) = R(\sqrt{42}) + R(42-2) = \cancel{R(\sqrt{42})} + R(40) = R(40) = 3.$
$R(44) = R(\sqrt{44}) + R(44-2) = \cancel{R(\sqrt{44})} + R(42) = R(42) = 3.$
$R(46) = R(\sqrt{46}) + R(46-2) = \cancel{R(\sqrt{46})} + R(44) = R(44) = 3.$
$R(48) = R(\sqrt{48}) + R(48-2) = \cancel{R(\sqrt{48})} + R(46) = R(46) = 3.$
$R(50) = R(\sqrt{50}) + R(50-2) = \cancel{R(\sqrt{50})} + R(48) = R(48) = 3.$
$R(52) = R(\sqrt{52}) + R(52-2) = \cancel{R(\sqrt{52})} + R(50) = R(50) = 3.$
$R(54) = R(\sqrt{54}) + R(54-2) = \cancel{R(\sqrt{54})} + R(52) = R(52) = 3.$
$R(56) = R(\sqrt{56}) + R(56-2) = \cancel{R(\sqrt{56})} + R(54) = R(54) = 3.$
$R(58) = R(\sqrt{58}) + R(58-2) = \cancel{R(\sqrt{58})} + R(56) = R(56) = 3.$
$R(60) = R(\sqrt{60}) + R(60-2) = \cancel{R(\sqrt{60})} + R(58) = R(58) = 3.$
$R(62) = R(\sqrt{62}) + R(62-2) = \cancel{R(\sqrt{62})} + R(60) = R(60) = 3.$
$R(64) = R(\sqrt{64}) + R(64-2) = R(8) + R(62) = 1 + 3 = 4.$
$R(66) = R(\sqrt{66}) + R(66-2) = \cancel{R(\sqrt{66})} + R(64) = R(64) = 4.$

Ответ: 4 программы.

Задача 4. Исполнитель Числовик преобразует введённое число, используя две возможные команды:

1. Прибавить 1

2. Умножить на 2

(т.е. первая команда увеличивает число на 1, а вторая умножает его на 2).

Программа для исполнителя Числовик представляет собой последовательность номеров команд.

Траекторией вычислений называется последовательность результатов (как промежуточных, так и итогового) выполнения команд программы. Например, для программы 121 и исходного числа 7 траектория состоит из чисел 8, 16, 17.

Сколько существует программ, для которых при исходном числе 2 результатом является число 29 и при этом траектория вычислений содержит число 14 и не содержит числа 25?

Решение

Поиск количества возможных программ производится так же, как и в предыдущих задачах без указания «особых» чисел (обязательных и/или отсутствующих).

Основные идеи при решении таких задач.

— Если задано число, которое обязательно должно содержаться в траектории вычислений, то нужно дважды провести решение: сначала найти количество возможных программ от исходного числа до обязательного, затем отдельно найти количество возможных программ от обязательного числа до конечного, а два полученных значения перемножить.

— Если задано число, которого не должно быть в траектории вычислений, то при подсчётах исключаются все значения $R()$, соответствующие этому числу.

В данной задаче исходное число равно 2, конечное равно 29, обязательное число — 14, отсутствующее число — 25.

1. Как и ранее, обозначим $R(n)$ — количество программ, которые преобразуют число 1 в число n. Для анализа решения будем рассматривать обратный процесс — получение числа 29 из числа 2 при помощи обратных команд «вычесть 1» и «разделить на 2». Тогда $R(n) = R(n-1) + R(n/2)$ *{в общем случае очередное число может быть получено или вычитанием единицы, или делением на два}*. При этом важно понимать, что в

каждой такой строке слагаемые $R(n-1)$ и $R(n/2)$ означают две возможные «траектории», приводящие к данному числу n.

2. Сначала расписываем каждый из шагов последовательности действий для получения из исходного числа 2 обязательного числа 14.

$R(2) = 1$ *{в любом случае исходное число — «в единственном экземпляре»}.*

$R(3) = R(3-1) + R(3/2) = R(2) = 1$ *{значение R(3/2) в целых числах не существует}.*

$R(4) = R(4-1) + R(4/2) = R(3) + R(2) = 1 + 1 = 2.$

$R(5) = R(5-1) + R(5/2) = R(4) = 2.$

$R(6) = R(6-1) + R(6/2) = R(5) + R(3) = 2 + 1 = 3.$

$R(7) = R(7-1) + R(7/2) = R(6) = 3.$

$R(8) = R(8-1) + R(8/2) = R(7) + R(4) = 3 + 2 = 5.$

$R(9) = R(9-1) + R(9/2) = R(8) = 5.$

$R(10) = R(10-1) + R(10/2) = R(9) + R(5) = 5 + 2 = 7.$

$R(11) = R(11-1) + R(11/2) = R(10) = 7.$

$R(12) = R(12-1) + R(12/2) = R(11) + R(6) = 7 + 3 = 10.$

$R(13) = R(13-1) + R(13/2) = R(12) = 10.$

$\boldsymbol{R(14) = R(14-1) + R(14/2) = R(13) + R(7) = 10 + 3 = 13.}$

3. Теперь отдельно, «с нуля» расписываем каждый из шагов последовательности действий для получения из обязательного числа 2 конечного числа 29. При этом не забываем обнулять значения $R()$, соответствующие отсутствующему числу 25.

$R(14) = 1$ *{изначальное число — «в единственном экземпляре»}.*

$R(15) = R(15-1) + R(15/2) = R(14) = 1.$

$R(16) = R(16-1) + R(16/2) = R(15) + R(8) = 1$ *{сейчас мы «забываем» обо всех предыдущих расчётах — число R(8) для нас не существует!}.*

$R(17) = R(17-1) + R(17/2) = R(16) = 1.$

$R(18) = R(18-1) + R(18/2) = R(17) + R(9) = 1$ *{число R(9) для нас не существует}.*

$R(19) = R(19 - 1) + R(19/2) = R(18) = 1.$

$R(20) = R(20 - 1) + R(20/2) = R(19) + R(10) = 1$ *{число R(10) для нас не существует}.*

$R(21) = R(21 - 1) + R(21/2) = R(20) = 1.$

$R(22) = R(22 - 1) + R(22/2) = R(21) + R(11) = 1$ *{число R(11) для нас не существует}.*

$R(23) = R(23 - 1) + R(23/2) = R(22) = 1.$

$R(24) = R(24 - 1) + R(24/2) = R(23) + R(12) = 1$ *{число R(12) для нас не существует}.*

А теперь — внимание! — мы дошли до отсутствующего числа 25. Значение $R(25)$ в расчётах везде приравниваем к нулю.

$\boldsymbol{R(25) = 0.}$

$R(26) = R(26 - 1) + R(26/2) = R(25) + R(13) = 0$ *{число R(13) для нас не существует}.*

$R(27) = R(27 - 1) + R(27/2) = R(26) = 0.$

$R(28) = R(28 - 1) + R(28/2) = R(27) + R(14) = 0 + 1 = 1.$

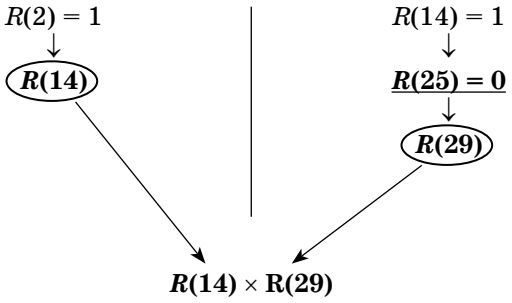
$\boldsymbol{R(29) = R(29 - 1) + R(29/2) = R(28) = 1.}$

4. Остаётся перемножить два полученных значения количеств программ, полученных на шагах 2 и 3:

$13 \times 1 = 13.$

Ответ: 13.

Схема решения задачи:

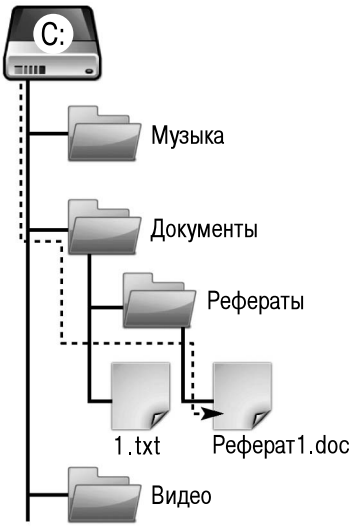


Раздел 6. **Архитектура компьютеров и компьютерных сетей**

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** — **C:\Документы\Рефераты** (показан пунктирной стрелкой);
- полное имя файла **Реферат1.doc** — **C:\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.

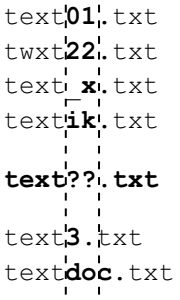
Примеры:

`*.*` — 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 имён 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`.

Задача 2. 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, 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 поочерёдной проверке каждой из предложенных масок (в вариантах ответа) на соответ-ствие 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 не обязательно) 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*.???`.

Задача 3. Для групповых операций с файлами используются маски имён файлов. Маска представляет собой последовательность букв, цифр и прочих допустимых в именах файлов символов, в которых также могут встречаться следующие символы:

• Символ «?» (вопросительный знак) означает ровно один произвольный символ.

• Символ «*» (звёздочка) означает любую последовательность символов произвольной длины, в том числе «*» может задавать и пустую последовательность.

В каталоге находится 6 файлов:

maverick.map
maverick.mp3
taverna.mp4
revolver.mp4
vera.mp3
zveri.mp3

Ниже представлено восемь масок. Сколько из них таких, которым соответствуют ровно три файла из данного каталога?

ver*.mp	***?ver?*.mp?**	**?*ver*.mp?***	***v*r*?.m?p***
???*???.mp*	**???*???.m***	***a*.*a***	***a*.*p***

Решение

Принцип решения задачи: «примерка» каждой маски к именам файлов и определение количества файлов, соответствующих маске. Маска, для которой будет отобрано ровно три файла (или маски), и есть ответ.

1) Маска ***ver*.mp*** предполагает, что где-то в имени файла содержится набор символов **ver**, а расширение начинается с символов **mp**. Значит, такая маска отберёт файлы **maverick.mp3**, **taverna.mp4**, **revolver.mp4**, **vera.mp3**, **zveri.mp3** — всего 5 файлов. Значит, эта маска нам *не годится*.

2) Маска ***?ver?*.mp?** отличается от предыдущей тем, что до и после символов **ver** обязательно должны стоять минимум по одному символу, а в расширении после символов **mp** может быть только один символ. Значит, такая маска отберёт файлы **maverick.mp3**, **taverna.mp4** и **zveri.mp3** (а имена **revolver.mp4** и **vera.mp3** не удовлетворяют этой маске, так как в них группа символов **ver** располагается с краю имён). Всего 3 файла. Значит, эта маска *подходит*.

3) Маска **?*ver*.mp?*** — минимум один символ должен быть перед **ver**, а в расширении после **mp** должно быть не менее одного символа. Такая маска отберёт файлы **maverick.mp3**, **taverna.mp4**, **revolver.mp4** и **zveri.mp3** — 4 файла. Данная маска *не годится*.

4) Маска ***v*r*?.m?p*** — не обращая внимания на маску имени, заметим: в расширении между **m** и **p** должен стоять хотя бы один какой-то символ. А у нас этому условию соответствует единственный файл **maverick.map**. Значит, эта маска тоже не годится.

5) Маска ***???*???.mp*** — ей удовлетворяют файлы с любыми именами не менее чем из 6 букв и с расширениями, начинающимися с **mp**. Значит, будут отобраны файлы **maverick.mp3**, **taverna.mp4** и **revolver.mp4** (у файлов **vera.mp3** и **zveri.mp3** имена слишком короткие, а расширение файла **maverick.map** не соответствует расширению в маске). Значит, данная маска тоже *подходит*.

6) Маска ***???*???.m*** — отличается от предыдущей тем, что здесь расширение может быть любым, лишь бы оно начиналось с **m**. Тогда в дополнение к файлам **maverick.mp3**, **taverna.mp4** и **revolver.mp4** в подборку попадёт и файл **maverick.map**. Всего 4 файла. Значит, эта маска *не годится*.

7) Маска ***a*.*a*** — соответствует любым файлам, у которых и в имени, и в расширении есть хотя бы одна буква **a**. Этому условию соответствует только один файл — **maverick.map**. Значит, эта маска тоже *не годится*.

7) Маска ***a*.*p*** — здесь в имени требуется хотя бы одна буква **a**, а в расширении — хотя бы одна буква **p**. Этому условию соответствуют файлы: **maverick.map**, **maverick.mp3**, **taverna.mp4** и **vera.mp3**. В этой подборке 4 файла, значит, данная маска тоже *не годится*.

Более удобно и наглядно можно записать решение в виде таблицы:

	ver.mp*	*?ver?*.mp?	?*ver*.mp?*	*v*r*?.m?p*	???*???.mp*	???*???.m*	*a*.*a*	*a*.*p*
maverick.map				+		+	+	+
maverick.mp3	+	+	+		+	+		+
taverna.mp4	+	+	+		+	+		+
revolver.mp4	+		+		+	+		
vera.mp3	+							+
zveri.mp3	+	+	+					
Кол-во файлов:	**5**	**3**	**4**	**1**	**3**	**4**	**1**	**4**

Итого нам подошли две маски — ***?ver?*.mp?** и **???*???.mp***.

Ответ: 2.

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

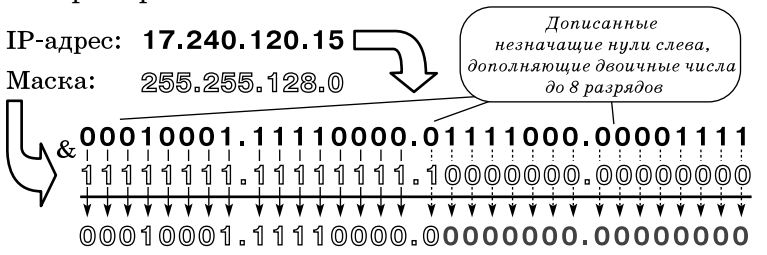
Если компьютер находится в составе локальной сети, которая в свою очередь подключена к сети Интернет, то возникает задача — определить IP-адрес всей локальной сети в целом (например, чтобы разослать пакет информации на *все* компьютеры данной локальной сети) и адрес конкретного компьютера внутри этой 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:

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:

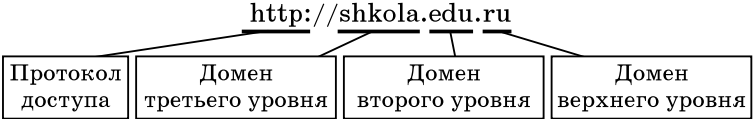
Маска: 255.255.128.0 $\Rightarrow$
11111111.11111111.10000000.00000000
Инвертированная маска:
00000000.00000000.01111111.11111111

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. 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 2. В терминологии сетей TCP/IP маской сети называется двоичное число, определяющее, какая часть IP-адреса узла сети относится к адресу сети, а какая — к адресу самого узла в этой сети. Обычно маска записывается по тем же правилам, что и IP-адрес — в виде четырёх байтов, причём каждый байт записывается в виде десятичного числа. При этом в маске сначала (в старших разрядах) стоят единицы, а затем с некоторого места — нули. Адрес сети получается в результате применения поразрядной конъюнкции к заданному IP-адресу узла и маске.

Например, если IP-адрес узла равен 231.32.255.131, а маска равна 255.255.240.0, то адрес подсети равен 231.32.240.0.

Для узла с IP-адресом 224.128.112.142 адрес сети равен 224.128.64.0. Чему равен третий слева байт маски? Ответ запишите в виде десятичного числа.

Решение

Здесь нам нужно искать не адрес сети, а наоборот, маску по заданным IP-адресу и адресу сети. Причём поскольку для байтов, которые в указанных адресах совпадают, байт маски очевидно равен 255, а там, где в адресе сети нулевой байт, байт маски так же очевидно тоже равен нулю, экзаменаторов интересует именно «ключевой» байт маски — третий слева. Так что задача для нас даже немного упростилась.

1) Преобразуем заданные третьи слева байты исходного IP-адреса и адреса сети в двоичные числа:

- $112_{10} = 1110000_2$;
- $64_{10} = 1000000_2$.

2) Для получения из исходного IP-адреса заданного адреса сети требовалась поразрядная конъюнкция (для каждого бита) IP-адреса и маски, т.е.:

```
  1110000
& *******
  -------
  1000000
```

Биты маски нам неизвестны. Но их очень легко определить:

- там, где значения битов IP-адреса и адреса сети оба равны 1, бит маски должен быть тоже равен 1;
- там, где бит IP-адреса равен 1, а бит адреса сети равен 0, бит маски должен быть равен 0.

Единственная неоднозначность — там, где биты IP-адреса и адреса сети оба равны нулю: тут бит маски может быть равен как 0, так и 1. Но устранить эту неоднозначность нетрудно, зная, что в маске при её просмотре слева направо может быть только один переход от единичных битов к нулевым, т.е., например, маска 11000110 недопустима. А мы видим, что уже со второго слева бита в маске должны начаться нули. Значит, и во всех «спорных» битах маски правее этого места тоже должны быть только нули.

3) Теперь нетрудно восстановить вышеприведенный «ребус» с конъюнкцией двоичных чисел:

```
  1110000
& 1000000
  1000000
```

4) Остаётся перевести найденную двоичную маску в десятичное число.

Для этого даже не понадобится никаких вычислений — полученная маска полностью совпала с двоичной записью байта адреса сети (64_{10}), значит, и искомый байт маски тоже равен 64.

Ответ: 64.

Задача 3. В сетях TCP/IP *маска сети* — это двоичное число, меньшее 2^{32}; в маске сначала (в старших разрядах) записаны единицы, а затем с некоторого бита — нули.

Маска определяет, какая часть IP-адреса относится к адресу подсети, а какая — к адресу конкретного компьютера (узла) в этой сети. Маска записывается по тем же правилам, что и IP-адрес, — в виде четырёх десятич-

ных чисел, каждое из которых соответствует одному байту и отделяется от других точкой. Адрес сети получается в результате применения поразрядной конъюнкции к заданному IP-адресу узла и маске.

Для узла с IP-адресом 167.57.252.220 адрес сети равен 167.48.0.0. Чему равен второй по счёту слева байт маски? Ответ нужно записать в виде десятичного числа.

Решение

Заметим: первый байт адреса сети совпадает с первым байтом IP-адреса — значит, первый байт маски равен 11111111_2; третий и четвёртый же байты адреса сети нулевые, значит, им соответствуют и нулевые байты маски. Второй же по счёту байт маски — самый «интересный», он должен содержать как единицы, так и нули. Именно поэтому в задаче требуется определить не всю маску, а только этот «ключевой» второй байт.

1) Переводим оба «ключевых» числа в двоичную систему счисления:

- $57_{10} = 111001_2$,
- $48_{10} = 110000_2$.

2) Записываем поразрядную конъюнкцию, в которой второй операнд неизвестен:

```
&  111001
   ??????
   110000
```

3) Сопоставляем первый операнд и результат:

- первый бит в обоих случаях равен 1, значит, соответствующий бит маски тоже равен 1;
- со вторым битом ситуация та же, значит, второй бит маски тоже равен 1;
- а вот третий бит в IP-адресе равен 1, а в адресе сети уже равен нулю, следовательно, соответствующий третий бит маски должен быть нулевым.

Четвёртый бит IP-адреса, как и четвёртый бит адреса сети, равен нулю. Тогда бит в маске может быть равен как 1, так и 0, — в обоих случаях конъюнкция с нулём даёт нуль.

Вспоминаем, что в маске сначала до какого-то разряда идут только единицы, а начиная с этого разряда — только нули. Поэтому если третий бит маски (как мы однозначно определили) равен нулю, то *все последующие* биты правее этого нуля тоже должны быть только нулями!

Поэтому маска получится такой:

110000_2 или 48_{10}.

Но этот ответ — неправильный!

Обязательно нужно вспомнить, что каждое из четырёх чисел, записанных в маске через точку, должно соответствовать одному байту, т.е. 8 битам. А у нас при переводе десятичных чисел 57 и 48 получились 6-битовые двоичные числа. Следовательно, их оба нужно дополнить до 8-битовых двумя незначащими нулями слева!

 Это — распространённая ошибка, когда учащиеся забывают, что каждое значение в маске должно представлять собой целый байт. Для получения правильного ответа нужно не забывать дописывать незначащие нули слева!

Тогда запись поразрядной конъюнкции должна иметь вид:

```
& 00111001
  ????????
  00110000
```

Теперь рассуждения о значениях битов маски тоже нужно начать с точно определяемых значений 3-го и 4-го слева битов: если соответствующие биты IP-адреса и адреса сети оба равны 1, то и соответствующие биты маски тоже равны 1.

Далее для 5-го слева бита значение бита маски тоже определяется однозначно: оно должно быть равно 0.

Вспомнив свойство маски, что в ней сначала до некоторого разряда записаны только единицы, а после этого разряда — только нули, определяем, что до найденных нами единичных битов тоже должны быть единицы, а после найденного нами нуля — нули. То есть маска будет такой: $11110000_2 = 240_{10}$. Это и есть правильный ответ.

Ответ: 240.

Задача 4. Для узла с IP-адресом 177.55.35.172 адрес сети равен 177.55.32.0. Чему равен наибольший возможный третий слева байт маски? (Ответ нужно записать в виде десятичного числа.)

Решение

Если в предыдущих задачах удавалось «однозначно решить неоднозначность» в распределении нулей и единиц в соответствующем байте маски, то здесь однозначно определить байт маски нельзя. В ней возможно несколько вариантов, именно поэтому в условии дано дополнительное уточнение — найти максимальный возможный байт маски.

1. Преобразуем десятичные значения третьих слева байтов IP-адресов в двоичную запись:

$$35_{10} = 00100011_2, \qquad 32_{10} = 00100000_2.$$

2. Записываем полученные значения «в столбик», где биты маски пока неизвестны:

```
&  00100011
   ????????
   00100000
```

3. Первые три бита маски определяются однозначно как единичные. Два последних бита тоже однозначно определяются как нулевые:

```
& 00100011
  111???00
  00100000
```

А вот с оставшимися тремя битами — неоднозначность. Они могут быть как единичными, так и нулевыми (и 0 & 1 = 0, и 0 & 0 = 0). То есть значение маски может быть в диапазоне от 11100000 до 11111100.

По условию, нужно найти наибольший байт маски. Тогда нам нужно значение 11111100.

 Если бы требовался наименьший возможный байт маски, то мы выбрали бы значение 11100000.

4. Остаётся преобразовать найденное двоичное значение байта маски в десятичное:

$11111100_2 = 252_{10}$.

Ответ: 252.

Задача 5. Для узла с IP-адресом 98.162.198.94 адрес сети равен 98.162.192.0. Для скольких различных значений маски это возможно?

Решение

В этой задаче требуется найти количество возможных различных значений маски, которые допустимы за счёт того самого неоднозначного определения значения маски, о котором говорилось в предыдущем задании.

1. Анализируя заданные значения IP-адреса узла и IP-адреса сети, замечаем, что разница между ними — в последних двух байтах. Именно их значения и требуется проанализировать.

2. Переводим десятичную запись значений этих байтов в двоичную:

$198 = 11000110_2$
$192 = 11000000_2$
$94 = 01011110_2$

3. Записываем два байта адреса сети под соответствующими двумя байтами адреса узла.

• Для первых по счёту слева направо двух битов (которые в обеих записях одинаковы и равны единицам) соответствующие биты маски однозначно должны быть единичными.

• Шестой слева бит адреса узла равен 1, а шестой бит (и последующие за ним) в адресе сети — нулевые. Следовательно, эти биты маски однозначно нулевые.

• Неоднозначность проявляется для трёх битов, которые и в адресе узла, и в адресе сети — нулевые: для них соответствующие биты маски могут быть как единичными (нулевой бит из адреса узла без изменения переписывается в адрес сети), так и нулевыми (маска «обнуляет нуль»).

Первая мысль — масок возможно столько, сколько возможно трёхзначных двоичных чисел (т. е. всех возможных комбинаций нулей и единиц в трёх битах). Но она неправильна: как мы помним, в маске после нулевого бита могут идти только нули. Поэтому возможные варианты масок получаются только «наращиванием» единиц слева направо.

11000110.01011110 *(IP-адрес узла)*
11000000.00000000 *(IP-адрес сети)*

Возможные маски:

11000000.00000000
11100000.00000000
11110000.00000000
11111000.00000000

А вот вариант

11111100.00000000

невозможен, иначе адрес сети был бы

11000100.00000000

Итого получаются четыре возможные IP-маски.

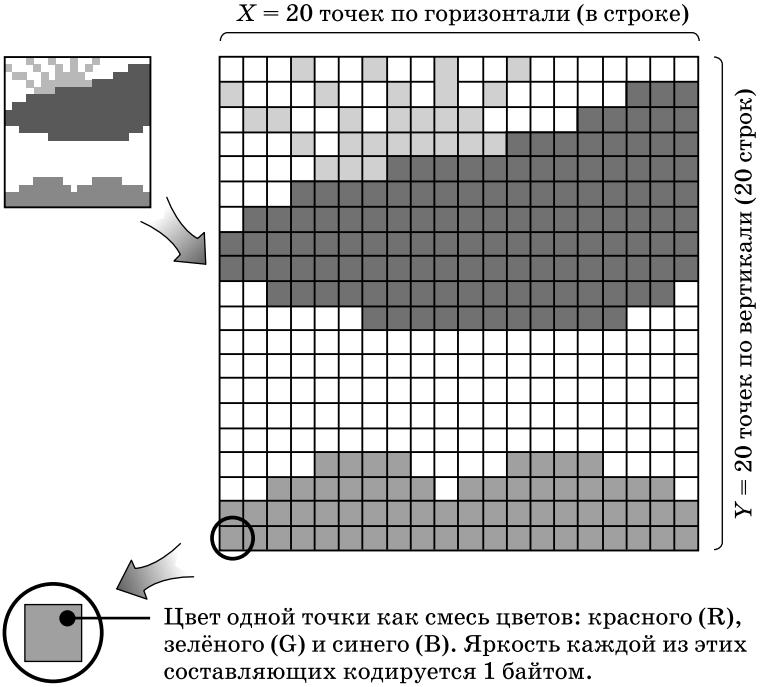
Ответ: 4

Раздел 7. Технология обработки звуковой и графической информации

Определение объёма и скорости передачи цифровой мультимедиа-информации

Конспект

Принципы цифрового кодирования растрового изображения

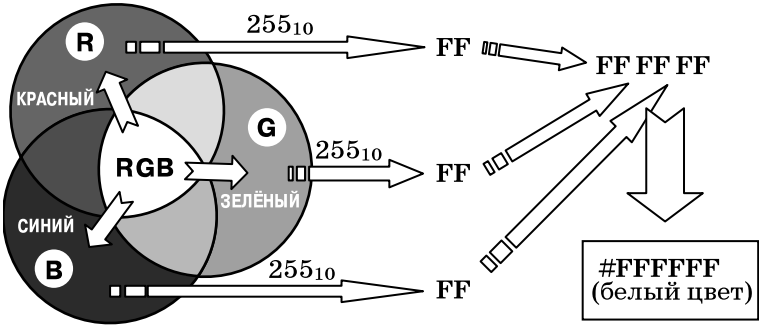


Цвет одной точки как смесь цветов: красного (R), зелёного (G) и синего (B). Яркость каждой из этих составляющих кодируется 1 байтом.
Для монохромного изображения цвет одной точки кодируется 1 битом.

Общий объём информации, байт = (цвет одной точки) × (кол-во точек в строке) × (кол-во строк) = 3 байта × 20 × 20 = 1200 байтов.

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-битовой RGB-модели** каждый из трёх основных цветов кодируется двоичным числом, разряды которого нумеруются от 0 до 7. То есть речь идёт о *8-битовом кодировании цветов в системе RGB*.

Red:

7	6	5	4	3	2	1	0

8 битов $\Rightarrow$ $2^8 = 256$ градаций

Green:

7	6	5	4	3	2	1	0

8 битов $\Rightarrow$ $2^8 = 256$ градаций

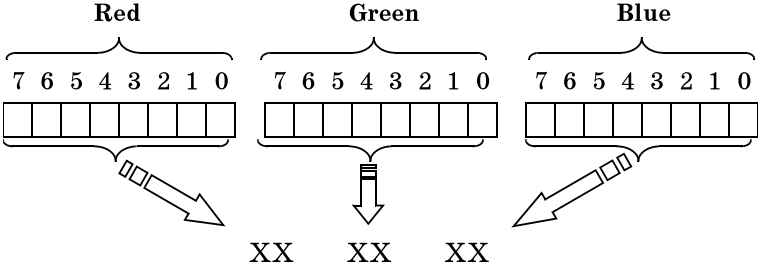
Blue:

7	6	5	4	3	2	1	0

8 битов $\Rightarrow$ $2^8 = 256$ 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-битовой 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***):



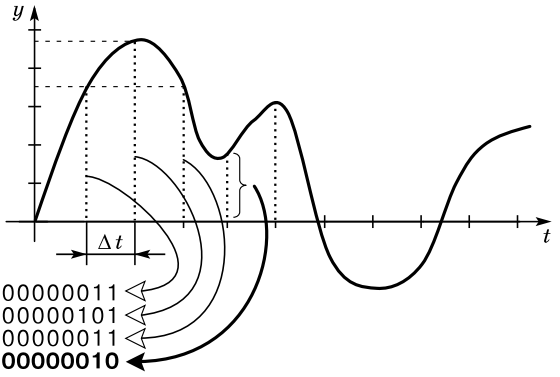
где стрелки обозначают перевод из двоичной в шестнадцатеричную систему счисления: исходное двоичное число (дополненное, если требуется, до 8 разрядов незначащими нулями слева) делится на две половины по 4 бита в каждой, а затем каждая такая «тетрада» битов заменяется одной шестнадцатеричной цифрой по 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 $= \underbrace{48\,000}_{\text{Частота дискретизации = кол-во замеров в секунду}} \times \underbrace{24\ (\text{бит})}_{\text{Разрешение = разрядность одного отсчёта}} \times \underbrace{60\ (\text{с})}_{\text{Длительность записи}} \times \underbrace{2}_{\text{Кол-во записываемых каналов}} =$

$= 138240000$ 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. Производилась двухканальная (стерео) звукозапись с частотой дискретизации 64 кГц и 24-битовым разрешением. В результате был получен файл размером 480 Мбайт, сжатие данных не производилось. Определите приблизительно, сколько времени (в минутах) проводилась запись. В качестве ответа укажите ближайшее к времени записи целое число, кратное 10.

Решение

1) Разрешение равно 24 бит.

2) Частота дискретизации равна 64 кГц — 64000 измерений в секунду.

3) Длительность записи (в секундах) примем равной x.

4) Запись двухканальная — объём информации умножается на 2.

5) Результирующий объём файла равен 480 Мб, или $480 \cdot 2^{23}$ бит.

(Не забываем, что все расчёты ведутся в битах и в секундах.)

6) Составляем уравнение:

$24 \times 64000 \times x \times 2 = 480 \cdot 2^{23}$.

Для удобства вычислений по возможности выделяем степени двойки:

$3 \cdot 2^3 \times 125 \cdot 2^9 \times x \times 2 = 15 \cdot 2^5 \cdot 2^{23}$;

$3 \times 5^3 \times x \times 2^{13} = 3 \cdot 5 \cdot 2^{28}$;

$25 \times x = 2^{15}$;

$x = 2^{15}/25 = 1310{,}72$ (секунд).

7) Переводим x в минуты:

$x \approx 1311 / 60 \approx 21{,}85$ (минут).

8) Округляем до ближайшего значения, кратного 10: $x \times 21{,}85 \approx 20$ (минут).

Ответ: 20.

Задача 2. Выполнена квадро (4-канальная) звукозапись с частотой дискретизации 32 кГц и 16-битовым разрешением. В результате получен файл размером 38 Мбайт, причём сжатие данных не производилось. Требуется приблизительно оценить, сколько времени (в минутах) производилась запись. В качестве ответа нужно указать ближайшее к полученному времени записи целое число минут.

Решение

Решение подобных задач сводится к записи одного-единственного уравнения, нужно только внимательно записать в него все составляющие:

количество каналов записи — 4,

частота — 32000 колебаний в секунду,

разрешение — 16 бит,

длительность — неизвестна и мы обозначим её как t;

получаемый размер файла равен 38 Мбайт = $= 38 \cdot 2^{23}$ бит.

Главное — ничего не забыть, обязательно преобразовать все величины к одним и тем же размерностям и правильно выполнить вычисления.

$4 \cdot 32000 \cdot 16 \cdot t = 38 \cdot 2^{23}$,

откуда $t = 155{,}648$. Это — в секундах. А у нас требуют указать длительность в минутах. Значит, полученное

значение надо обязательно разделить на 60. А также (согласно условию задачи) — округлить полученное дробное значение *до ближайшего целого*:

$t = 155{,}648/60 \approx 2{,}594 \approx 3$ минуты.

Ответ: 3.

Задача 3. Для хранения растрового изображения размером 128×240 пикселей отвели 30 Кбайт памяти. Каково максимально возможное число цветов в палитре?

Решение

Обозначим как x количество битов, приходящееся на один пиксель. Тогда общий объём памяти для хранения изображения равен $128 \cdot 240 \cdot x$, и он равен 30 Кбайт, или $30 \cdot 2^{13}$ бит. Из этого уравнения выразим x: $128 \cdot 240 \cdot x = 30 \cdot 2^{13} \Rightarrow x = 30 \cdot 2^{13} / (128 \cdot 240) = 8$.

Если для кодирования цвета пикселя отводится 8 бит, то максимальное количество цветов равно $2^8 = 256$.

Ответ: 256.

Задача 4. Фотокамера снимает растровые изображения размером 800×600 пикселей. При этом объём графического файла не может превышать 600 Кб. Упаковка данных не производится. Какое максимальное количество цветов может быть в палитре?

Решение

1. Зная разрешение изображения, можно вычислить количество пикселей в нём: $800 \times 600 = 480\,000$.

2. Максимальный объём графического файла равен 600 Кб — значит, на один пиксель может приходиться: $(600 \times 2^{13}$ бит$) / 480\,000 = 10$ бит (получаемое не целое частное 10,24 округляется до целого в **меньшую** сторону).

3. Этих 10 бит будет достаточно, чтобы закодировать $2^{10} = 1024$ цвета.

Ответ: 1024.

Наиболее распространённая ошибка при решении такой задачи — округление получаемого дробного количества бит в большую сторону.

Почему это делать нельзя?

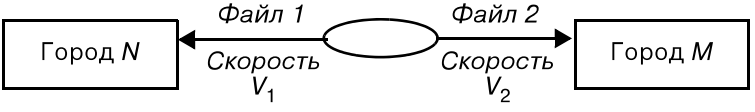
Потому что, хотя мы обязаны выполнить округление (количество битов не может быть дробным), если округлить значение в большую сторону, то при заданном числе пикселей мы тем самым увеличим получаемый объём графического файла сверх обозначенного максимума. Поэтому округление нужно выполнять до меньшего целого значения.

Задача 5. Звуковой фрагмент оцифровали и записали в файл без использования сжатия файлов. Этот файл передали в город *N* по каналу связи за полторы минуты. После этого тот же самый звуковой фрагмент оцифровали заново с разрешением в 4 раза меньше и частотой дискретизации в 8 раз больше, чем в первый раз. Новый файл передали в город *M* по второму каналу связи на полминуты быстрее, чем первый файл — в город *N*. Во сколько раз канал связи с городом *M* быстрее канала связи с городом *N*?

Решение

Здесь, кроме основ кодирования звуковой информации, потребуется вспомнить формулы скорости передачи данных.

1. Изобразим оба процесса передачи данных наглядно:



2. Пусть в первом случае объём файла равен *X*. Тогда:

— уменьшение разрешения в 4 раза позволяет уменьшить в те же 4 раза объём файла;

— увеличение частоты дискретизации в 8 раз соответственно увеличивает объём файла в те же 8 раз.

Следовательно, объём второго файла составит $X \cdot 8/4 = 2X$.

3. Скорость (пропускная способность) первого канала связи $V_1 = \frac{X}{90}$ (время выражаем в секундах).

Скорость (пропускная способность) второго канала связи $V_2 = \frac{2X}{60}$ (время тоже выражаем в секундах).

4. В соответствии с заданием ищем отношение $V_2/V_1 = \frac{\frac{2X}{60}}{\frac{X}{90}} = \frac{2X \cdot 90}{60 \cdot X} = \frac{2 \cdot 90}{60} = 3$. То есть канал связи с городом M быстрее канала связи с городом N в 3 раза.

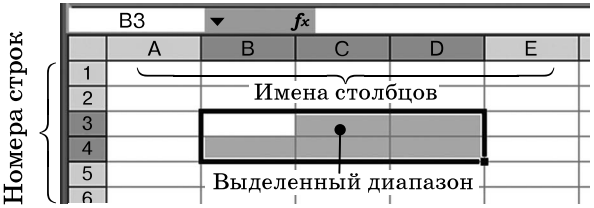
Ответ: 3.

Раздел 8. **Обработка числовой информации**

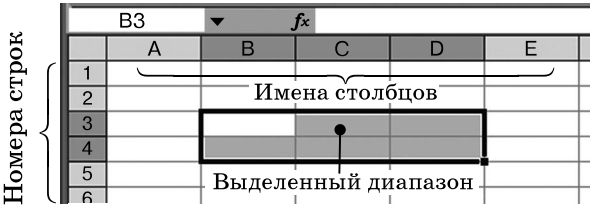
Электронные таблицы. Ссылки. Формулы

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

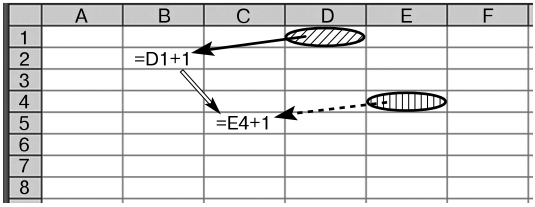
Ссылка на ячейку представляет собой запись имени ячейки. Ссылка на диапазон представляет собой запись имени диапазона.

Примеры:

=A3 + C5 — сложить числа в ячейках **A3** и **C5**;

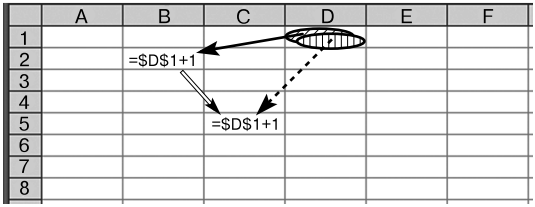
=СУММ(A3:C5) — функция вычисления суммы чисел во всём диапазоне **A3:C5**.

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**.



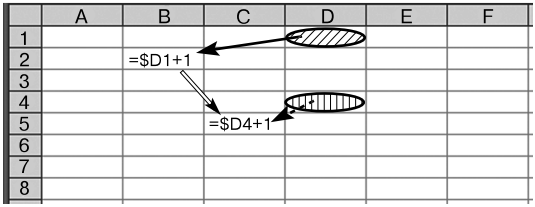
При копировании формулы из ячейки **B2** в ячейку **C5** ссылка меняется так, что всегда указывает на ячейку, расположенную на 2 столбца правее и на 1 строку выше *относительно* ячейки с формулой.

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.

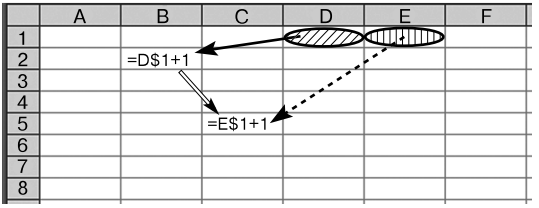


При копировании формулы из ячейки **B2** в ячейку **C5** ссылка не меняется и всегда указывает на одну и ту же ячейку.

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 распределённое 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 таблицы

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 таблицы

	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 таблицы

	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?

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.

Задача 2. Дан фрагмент электронной таблицы. Из ячейки C3 в одну из ячеек столбца D была скопирована формула. После копирования значение этой формулы стало равным 100.

В какую ячейку была скопирована формула? В ответе надо записать только номер строки, в которой находится эта ячейка.

	A	B	C	D
1	5	13		
2	6	12		
3	7	11	=$B3*A$4	
4	8	10		

Решение

Сначала определим, как меняется формула при её копировании в ячейки столбца D (с учётом абсолютной адресации):

	C	D
1		=$B1*B$4
2		=$B2*B$4
3	=$B3*A$4	=$B3*B$4
4		=$B4*B$4

Остаётся только вычислить произведения и сверить их с требуемым. Впрочем, и так сразу видно, что 100 — это 10*10, а подходящая формула — только в ячейке D4 (=$B4*B$4).

Ответ: 4.


Важно обратить внимание на то, что нужно внимательно читать условие задачи — в том числе о том, как надо записать ответ. Запись ответа в виде «D4» при автоматической проверке ответов будет признана за ошибку!

Задача 3. В ячейки электронной таблицы записаны числа:

	A	B	C	D	E
1					
2			3	2	1
3		1	2	3	4
4	5	4	3	2	1

В ячейку **A1** записали формулу **=$C3 − D$2.** После этого ячейку **A1** скопировали в ячейку **B2.** Какое число будет выведено в ячейке **B2**?

Примечание: знак $ используется для обозначения абсолютной адресации.

Решение

Достаточно простая задача на знание видов ссылок в электронных таблицах.

1. Исходная формула в ячейке **A1: =$C3 − D$2**, т.е. используются смешанные ссылки.

2. При копировании ячейки **A1** в **B2** в относительных ссылках (и в «относительных» частях смешанных ссылок) значения имён столбцов и номеров строк должны увеличиться на единицу, а абсолютные ссылки (и «абсолютные» части смешанных ссылок) остаются неизменными. Поэтому формула в ячейке **B2** после копирования примет вид: **=$C4 − E$2.**

3. Подставляя в полученную формулу значения из указанных в ней ячеек, определяем результат вычислений по этой формуле, который будет отображён в ячейке **B2**: 3 − 1 = 2.

Ответ: 2.

Задача 4. Известно, что в ячейке C3 записана формула — сумма двух других ячеек из диапазона A1: C3. Формулу из ячейки C3 скопировали в ячейки D3 и C4,

после чего значения в них стали равны 6 и 11 соответственно. Какое значение примет ячейка D4, если в неё также скопировать формулу из ячейки C3?

	A	B	C	D
1	1	2	5	
2	7	4	7	
3	1	1	9	
4				

Решение

1. Смотрим, какие из пар значений ячеек A1: C3 могут дать в сумме 9:

A2 + B1 = 7 + 2 = 9,

B1 + C2 = 2 + 7 = 9,

B2 + C1 = 4 + 5 = 9.

2. Проверяем каждый вариант, при этом учитывая, что в предполагаемой формуле могут использоваться и знаки абсолютной адресации ($).

Один из подходящих вариантов: формула

A$2 + $B1.

В ячейке C3 она даёт 7 + 2 = 9, при копировании в ячейку D3 формула преобразуется в B$2 + $B1 = 4 + + 2 = 6, а при копировании в ячейку C4 получаем формулу A$2 + $B2= 7 + 4 = 11, — что и требовалось.

Тогда при копировании в ячейку D4 имеем:

B$2 + $B2 = 4 + 4 = 8.

Ответ: 8.

Раздел 9. Технологии поиска и хранения информации

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.

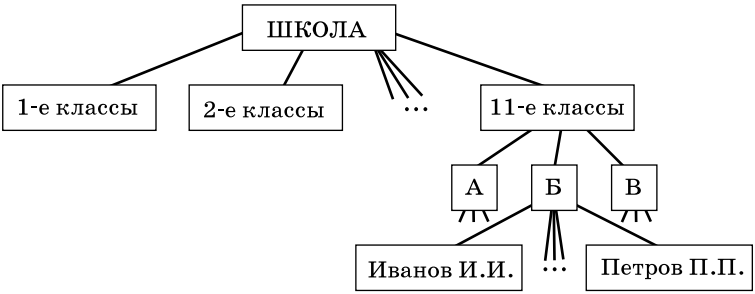
Система управления базами данных (СУБД) — комплекс программных и языковых средств для создания и модификации структуры БД, добавления, изменения, удаления, поиска и отбора данных, их представления на экране ПК и в печатном виде, разграничения прав доступа к информации и выполнения других операций с БД.

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.

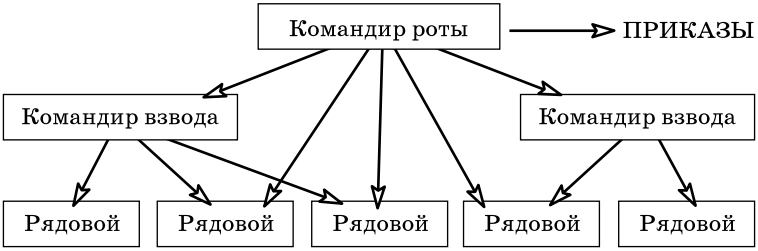
Классификацию БД обычно проводят по типам используемых в них **моделей данных:**

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 таблицы

(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 поочерёдном 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. В фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных фамилию бабушки Ивановой А.И.

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
…		

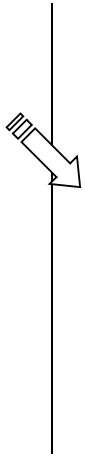


Таблица 2

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
…		

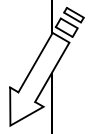


Таблица 2

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.

Задача 2. В фрагменте базы данных представлены сведения о родственных отношениях. Определите на основании приведённых данных ID 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
…	…

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
…		

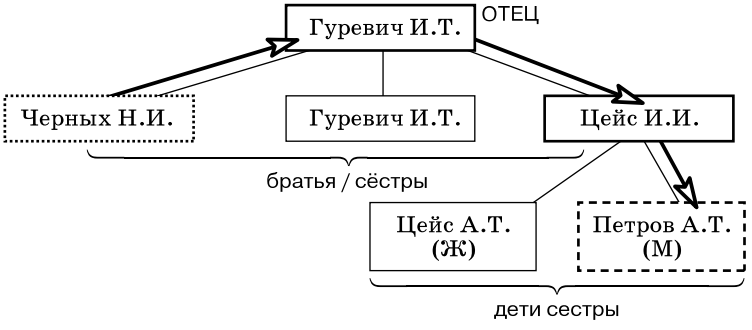
Таблица 2

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. По левой таблице определяем его ID: 23.

Ответ: 23.

Примечание. Для наглядности можно изобразить «генеалогическое древо» всех рассмотренных в задаче лиц с указанием стрелками пути поиска племянника.



Задача 3. В фрагменте базы данных представлены сведения о родственных отношениях. На основании приведённых данных определите, сколько прямых потомков (то есть детей и внуков) Кривич Л.П. упомянуты в таблице.

Таблица 1

ID	Фамилия_И.О.	Пол
2146	Кривич Л.П.	Ж
2155	Павленко А.К.	М
2431	Хитрук П.А.	М
2480	Кривич А.А.	М
2302	Павленко Е.А.	Ж
2500	Сокол Н.А.	Ж
3002	Павленко И.А.	М
2523	Павленко Т.Х.	Ж
2529	Хитрук А.П.	М
2570	Павленко П.И.	М
2586	Павленко Т.И.	Ж
2933	Симонян А.А.	Ж
2511	Сокол В.А.	Ж
3193	Биба С.А.	Ж
...	...	...

Таблица 2

ID_Родителя	ID_Ребёнка
2146	2302
2146	3002
2155	2302
2155	3002
2302	2431
2302	2511
2302	3193
3002	2586
3002	2570
2523	2586
2523	2570
2529	2431
2529	2511
2529	3193
...	...

Решение

Данная задача решается так же, как предыдущие, но здесь нужно, начиная от родителей, определять по базе данных сначала детей, потом внуков и т.д., подсчитывая их количество. (Аналогичным способом можно подсчитывать и количество других родственников.) Реально, как видно из решения, эта задача даже несколько упрощена по сравнению с прежними, особенно теми, где нужно было искать племянников, племянниц или другую родню из «боковых ветвей» генеалогического древа.

1) В таблице 1 ищем Кривич Л.П. и определяем её ID: 2146.

2) В правой таблице ищем все пары ID, в которых первое число (в левой колонке) равно 2146. Для них в правой колонке записаны ID 2302 и 3002. Это — дети Кривич Л.П.

3) В правой таблице ищем все пары ID, в которых первое число (в левой колонке) равно 2302 или 3002. Это — внуки Кривич Л.П.

4) Подсчитываем количество найденных в таблице 2 строк: 7 (двое детей и пятеро внуков).

Ответ: 7.

Задача 4. Даны фрагменты двух таблиц из базы данных. Каждая строка правой таблицы содержит информацию о ребёнке и об одном из его родителей. Информация представлена значением поля ID в соответствующей строке левой таблицы.

На основании имеющихся данных определите, у скольких людей из списка первый внук или внучка появились после достижения 60 полных лет. При вычислении ответа учитывайте только информацию из приведённых фрагментов таблиц.

ID	Фамилия И.О.	Пол	Год рождения
127	Петренко А.В.	М	1935
148	Петренко Д.И.	М	2000
182	Петренко Е.П.	Ж	1942
212	Петренко И.А.	М	1974
243	Петренко Н.Н.	Ж	1975
254	Штейн А.Б.	М	1982
314	Косых Е.А.	М	2006
404	Дулевич М.А.	Ж	1970
512	Тишко О.К.	Ж	1991
517	Дулевич В.К.	М	1996
630	Штейн Б.В.	М	1954
741	Петрова А.Е.	Ж	1958
830	Штейн А.Н.	Ж	1980
849	Косых Н.Н.	М	1939

ID Родителя	ID Ребёнка
127	212
182	212
212	148
243	148
254	314
127	404
182	404
404	512
404	517
630	254
741	254
830	314
849	243
849	830

Решение

Несмотря на кажущееся серьёзным усложнение, данная задача решается аналогично рассмотренным ранее. Нужно определять для каждого указанного в таблице родителя информацию о его внуках (как «детях детей» — двойной просмотр пар ID родителя — ребёнка). Далее нужно среди найденных внуков определить первого (с наименьшим годом рождения) и сопоставить год рождения внука с годом рождения бабушки/дедушки: разница между ними должна быть не менее 60 лет.

Начинаем с «детей». Для очередного ID ребёнка в правой таблице определяем ID его родителей. Затем найденные ID родителей (один или два номера) смотрим уже среди «детей» и ищем для них ID уже их родителей (т.е. бабушек и дедушек).

Удобнее всего составить отдельную таблицу всех ID дедушек/бабушек и их внуков:

Внук/внучка?	Родители	Бабушки/дедушки?
212	127	Нет данных
	182	
148	212	127
		128
	243	849
314	254	630
		741
404	127	Нет данных
	182	
512	404	127
		182
517	404	127
		182
254	630	Нет данных
	741	
314	830	849
243	849	Нет данных
830	849	Нет данных

Строки с пометкой «Нет данных» из дальнейшего рассмотрения, очевидно, исключаются.

К оставшимся строкам добавляем столбцы, в которые переписываем из левой таблицы годы рождения дедушек/бабушек и внуков/внучек. Разницу лет вычисляем как разность годов рождения внука/внучки и бабушки/дедушки, при этом отслеживаем: какой внук/внучка являются первыми (старшими — с наименьшим годом рождения), если их у данных бабушки/дедушки несколько.

Внук/внучка?	Родители	Бабушки/дедушки?	Год рождения бабушек/дедушек	Год рождения внука/внучки	Разница лет	Да/нет по возрасту бабушки/дедушки	Первый ли это внук/внучка?
148	212	127	1935	2000	65	+	– (не первый)
		128	Нет данных				
	243	849	1939		61	+	+
314	254	630	1954	2006	52	–	
		741	1958		48	–	
512	404	127	1935	1991	56	–	
		182	1942		49	–	
517	404	127	1935	1996	61	+	– (не первый)
		182	1942		54	–	
314	830	849	1939	2006	67	+	– (не первый)

Остаётся подсчитать количество записей (строк) со знаками «+» в последнем столбце. В нашем случае такой знак получился только один.

Ответ: 1.

Задача 5. Даны фрагменты двух таблиц из базы данных. Каждая строка таблицы Б содержит информацию (идентификационный номер ID) о ребёнке и одном из его родителей; полную информацию о каждом ребёнке или родителе можно найти по его ID в таблице А.

На основании этих таблиц определите, у скольких человек имеется брат, разница в возрасте с которым составляет не более 5 лет. При вычислении ответа учитывайте только информацию из приведённых фрагментов таблиц.

Таблица А

ID	Фамилия И.О.	Пол	Год рождения
866	Кравец Д.К.	Ж	1942
867	Тошич Б.Ф.	М	1938
879	Гонтарь В.А.	Ж	1998
885	Крон К.Г.	М	1984
900	Кислюк Л.А.	М	2012
904	Петраш А.И.	Ж	2010
911	Тошич А.Б.	Ж	1971
932	Петраш П.А.	Ж	2016
938	Тошич И.Б.	М	1974
949	Седых Г.Р.	Ж	1966
970	Кислюк А.П.	М	1987
995	Тошич Т.И.	М	2002
1017	Тошич П.И.	М	2005
1026	Мухина Р.Г.	Ж	1983
1041	Сайко М.А.	М	2010
1056	Кислюк П.А.	М	1994
...	...	...	...

Таблица Б

ID Родителя	ID Ребёнка
866	911
866	938
867	911
867	938
911	879
938	995
938	1017
1026	900
949	885
949	970
949	1056
885	904
885	932
970	1041
...	...

Решение

Здесь нам нужно искать братьев (но не сестёр!) и определять их разницу в возрасте.

Проще всего сначала искать в таблице Б всех детей для каждого родителя, а затем находить информацию по этим детям в таблице А и определять, подходят ли они под заданные условия.

Для наглядности составим таблицу:

ID родителей	ID детей	Пол детей	Возраст детей	Разница в возрасте	Подходит ли под условие задачи?
866	911	Ж	1971	1974 – 1971 = = 3 (≤ 5)	**есть такой брат**
	938	М	1974		сестра
867	911	Ж	1971	1974 – 1971 = = 3 (≤ 5)	*по этим детям решение уже найдено*
	938	М	1974		
911	879	*единственный ребёнок*			братьев нет
938	995	М	2002	2005 – 2002 = = 3 (≤ 5)	**есть такой брат**
	1017	М	2005		**есть такой брат**
1026	900	*единственный ребёнок*			братьев нет
949	885	М	1984	1987 – 1984 = = 3 (≤ 5)	**есть такой брат**
	970	М	1987	1994 – 1987 = = 7 (> 5)	**есть такой брат**
	1056	М	1994	1994 – 1984 = = 10 (> 5)	братья старше 5 лет
885	904	Ж	*братьев в семье нет*		сестра
	932	Ж			сестра
970	1041	*единственный ребёнок*			братьев нет

Нужно помнить, что при наличии в семье нескольких братьев у каждого из них есть один или несколько братьев, а если в семье есть мальчик и девочка, то у мальчика братьев нет, но у девочки (сестры) брат есть!

Подсчитываем в правой колонке нашей таблицы «положительные ответы» и делаем вывод: брата с разницей в возрасте менее 5 лет имеют 5 человек (с ID: 911, 995, 1017, 885 и 970; повторов номеров нет — значит, никого мы дважды не посчитали).

Ответ: 5.

Поиск информации в сети Интернет. Поисковые запросы

Конспект

Поисковый запрос для поисковой системы в Интернете представляет собой ключевое слово или несколько ключевых слов, соединённых 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:

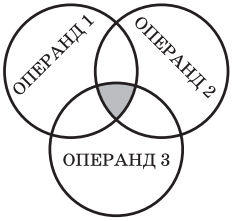
[1] Язык поисковых запросов, реализованный на почтовых сервисах (Яндекс, Google и пр.), включает также ряд других операций с ключевыми словами, позволяющих осуществлять поиск более гибко.

- **операция «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;
- **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 за-

даче 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;
- 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

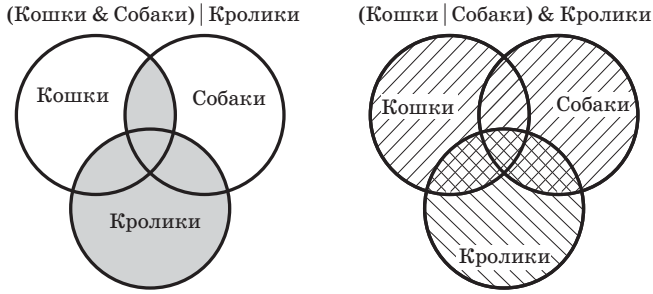
(кошки | собаки) & кролики

В первом случае будут найдены документы, в которых есть *оба* слова — «кошки» и «собаки», и к ним будут добавлены *все* документы со словом «кролики».

---

[1] Если требуется, наоборот, выстроить поисковые запросы по порядку ***уменьшения*** количества найденных документов, то задача решается аналогично, но расположение запросов в списке будет обратным.

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».



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**) множе-

ства, 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 операци-

ей 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»

Продолжение таблицы

Запрос	Уравнение	Комментарий
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)

Следовательно, 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:

$$\begin{cases} ❶ + 100 + ❸ + 100 + 200 + 100 + ❼ = 2600 \\ ❶ + 100 + 100 + 200 = 1300 \\ 100 + ❸ + 100 + 100 = 800 \\ \boxed{\begin{array}{l} ❺ = 200 \\ ❷ = 100 \\ ❻ = 100 \end{array}} \end{cases}$$

Искомая сумма:
$100 + 200 + 100 + ❼$

Из второго и третьего уравнений определяются значения ❶ и ❸ и подставляются в первое уравнение:

$$\begin{cases} 900 + 100 + 500 + 100 + 200 + 100 + ❼ = 2600 \\ \boxed{\begin{array}{l} ❶ = 900 \\ ❸ = 500 \end{array}} \end{cases}$$

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. 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:

Складываются 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.

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	Кол-во найденных страниц, тыс.
Паскаль & (Си & Бейсик | Кобол)	350
Паскаль & Кобол	187
Паскаль & Си & Бейсик & Кобол	48

Какое количество страниц (в тысячах) будет найдено по запросу

Паскаль & Си & Бейсик ?

Считаем, что все запросы выполнялись почти одновременно и набор страниц, содержащих искомые слова, за это время не изменялся.

Решение

«Традиционное» решение такой задачи — построение кругов Эйлера. Но здесь — целых четыре переменных, а числовых данных явно недостаточно, чтобы выполнить полное решение. Как же быть?

Обратим внимание на запрос *Паскаль* & (*Си* & *Бейсик* | *Кобол*) и раскроем скобки в этом логическом выражении: *Паскаль* & *Си* & *Бейсик* | *Паскаль* & *Кобол*.

А теперь посмотрим на запрос *Паскаль* & *Си* & *Бейсик* & *Кобол*. Правда, похоже? И там, и там слева стоит «*Паскаль* & *Си* & *Бейсик*». А если сравнить со вторым запросом — *Паскаль* & *Кобол*, то мы увидим: в первом запросе справа от «|» записано «*Паскаль* & *Кобол*» (как во втором запросе), а в третьем запросе правая часть — только «*Кобол*». Но ведь существует «правило поглощения»: $A \& B \& A = A \& B$ (повторяющийся операнд A можно не повторять). А значит, верно и обратное: любой операнд можно повторить (записав через тот же знак операции) сколько угодно раз, не меняя значение логического выражения!

Воспользуемся этим и повторим в третьем запросе слово *Паскаль*: *Паскаль* & *Си* & *Бейсик* & *Паскаль* & & *Кобол*. Значение выражения не изменилось, но зато его теперь можно представить в виде: (*Паскаль* & *Си* & & *Бейсик*) & (*Паскаль* & *Кобол*).

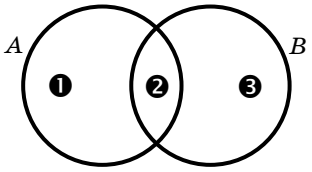
Итак, четыре наших запроса (включая искомый) теперь выглядят так:

Паскаль & *Си* & *Бейсик* | *Паскаль* & *Кобол*
Паскаль & *Кобол*
(*Паскаль* & *Си* & *Бейсик*) & (*Паскаль* & *Кобол*)
Паскаль & *Си* & *Бейсик*

Совершенно очевидно, что можно выполнить «макроподстановку»:

A = *Паскаль* & *Си* & *Бейсик*,
B = *Паскаль* & *Кобол*
Тогда наши запросы примут вид:
A | ***B*** (350 тыс.стр.),
B (187 тыс.стр.),
A & ***B*** (48 тыс.стр.),
A — искомый.

Теперь мы получили задачу всего с двумя логическими переменными (и соответственно с двумя кругами Эйлера):



$A \mid B$	❶ + ❷ + ❸	350 тыс. стр.
B	❷ + ❸	187 тыс. стр.
A & B	❷	48 тыс. стр.
A	❶ + ❷	*Требуется найти*

Составляем уравнения:

$$\begin{cases} ❶ + ❷ + ❸ = 350; \\ ❷ + ❸ = 187; \\ ❷ = 48. \end{cases} \Rightarrow \begin{cases} ❶ + 48 + ❸ = 350; \\ 48 + ❸ = 187; \end{cases} \Rightarrow \begin{cases} ❶ + ❸ = 302; \\ ❸ = 139; \end{cases}$$

$$\Downarrow$$

$$❶ + 139 = 302;$$

Отсюда легко вычислить, что ❶ = 163, а ❶ + ❷ = = 163 + 48 = 211.

Следовательно, по запросу *Паскаль & Си & Бейсик* будет найдено 211 тыс. стр.

Ответ: 211.

Задача 3. 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».

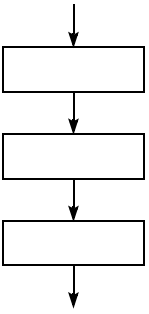
Ответ: 4321.

Условный оператор. Циклы

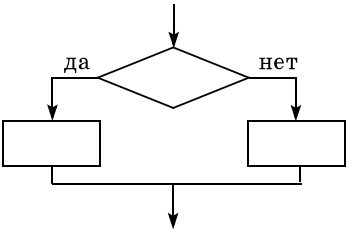
Конспект

Основные алгоритмические конструкции

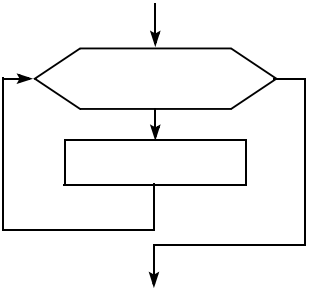
Линейный алгоритм



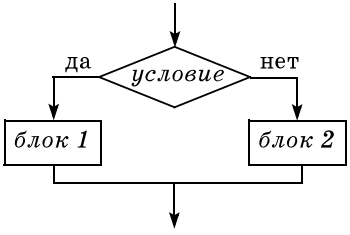
Алгоритм с ветвлением



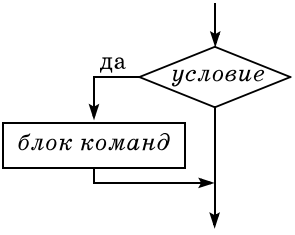
Циклический алгоритм



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.

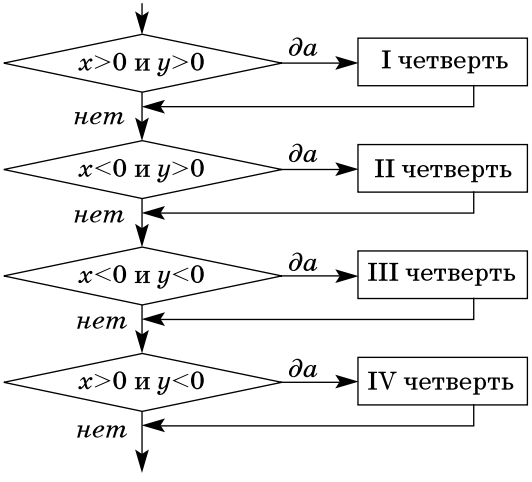
Полные и неполные условия в цепочке операторов `if`

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:

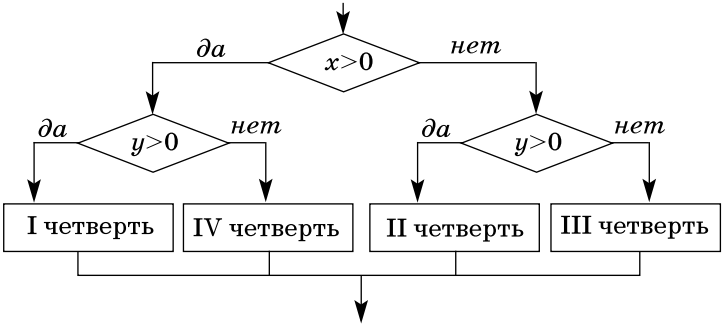
```
if (x>0) and (y>0) then writeln('I четверть');
if (x<0) and (y>0) then writeln('II четверть');
if (x<0) and (y<0) then writeln('III четверть');
if (x>0) and (y<0) then writeln('IV четверть');
```



2. Использование частичных условий

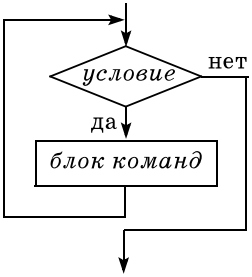
Частичное условие является неполным и определяет несколько возможных ситуаций (например, условие $x > 0$ указывает, что точка расположена в правой полуплоскости). Для определения конкретной ситуации нужно использовать ещё одно ветвление, проверяющее оставшиеся условия (дополняющие уже проверенное до полного набора условий), например, условие $y > 0$, проверяемое в ветви `then` оператора `if` с условием $x > 0$, однозначно определяет принадлежность точки I 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;
```



Цикл с предусловием
(цикл ПОКА)

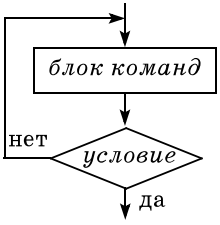
Обеспечивает выполнение блока команд, составляющего тело цикла, пока условие остаётся истинным (выход из цикла — по ложности условия).



Условие проверяется *до* начала выполнения цикла, поэтому возможна ситуация, что тело цикла не будет выполнено ни разу.

Цикл с постусловием
(цикл ДО)

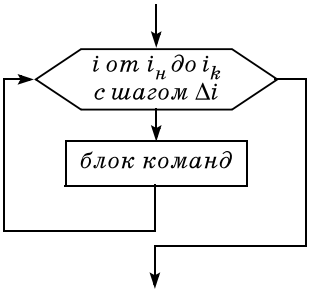
Обеспечивает выполнение команд, составляющих тело цикла, пока условие не станет истинным (цикл выполняется, пока условие ложно).



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 неизвестно 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

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> := <iн> to
<iк> do
  begin
  <блок команд>
  end;
или
for <i> := <iн> to
<iк> do <команда>;
	for <i> := <iк>
	downto <iн> do
	  begin
	  <блок команд>
	  end;
	или
	for <i> := <iк>
	downto <iн> 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:

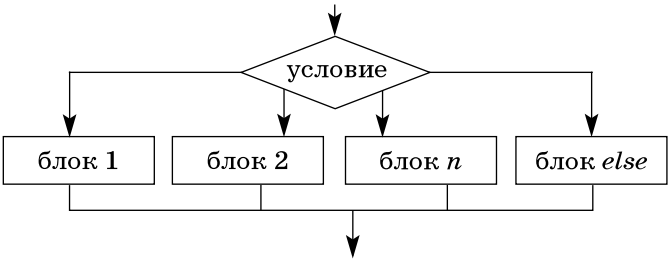
- циклы не должны «пересекаться», т.е. внутренний цикл должен начинаться и полностью завершаться внутри тела внешнего цикла;
- во внутреннем и во внешнем циклах ДЛЯ должны использоваться разные переменные-счётчики.

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.

Оператор языка Паскаль	Описание
`break`	**Оператор прерывания** — выполнение данного оператора прекращает выполнение цикла и передаёт управление на следующий оператор после цикла.

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 про-

грамме. 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. Ниже приведён фрагмент программы. При каком наименьшем введенном числе a после выполнения программы значение переменной c будет равно 60?

```
readln(a);
b := 50;
a := a * 2 - b;
if a < b then
  c := b - 10
else
  c := b + 10;
```

Решение

В отличие от предыдущей задачи здесь требуется не просто формально «протрассировать» заданную программу, а проанализировать алгоритм её работы и понять, какими могут быть значения введённого числа a.

1) Изменение значения переменной b выполняется однократно (одним из операторов в ветвях `then` или `else`).

2) Изначально $b = 50$, а нам нужно, чтобы это значение стало равным 60. Следовательно, нам нужно для этого выполнить оператор `c:=b+10` в ветви `else`.

3) Чтобы выполнялась ветви `else`, требуется, чтобы к моменту выполнения оператора `if` уже пересчитанное значение a (`a:=a*2-b`) было не меньше (т.е. больше или равно) b, т.е. числа 50.

4) Решаем полученное неравенство:

$a \cdot 2 - 50 \geqslant 50; \quad \Rightarrow \quad 2a \geqslant 100; \quad \Rightarrow \quad a \geqslant 50.$

5) Это *почти* уже ответ. Нам же нужно определить, какое возможное значение a является наименьшим. Так как неравенство нестрогое, это значение $a = 50$.

Ответ: 50.

Задача 3. Определите, что будет напечатано в результате работы следующего фрагмента программы:

Бейсик	Паскаль

Dim k, s As Integer
s = 0
k = 0
While s < 1024
s = s + 10
k = k + 1
End While
Console.Write(k)
	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	**Алгоритмический язык**

{
int к, s;
s = 0;
к = 0;
while (s < 1024) {
s = s + 10;
к = k + 1;
}
printf("%d", k); }
	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

Продолжение таблицы

Операторы	*s*	*k*
`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 4. Ниже приведена программа. При каком наибольшем введённом числе d после выполнения программы будет напечатано 55?

```
var n, s, d: integer;
begin
  readln(d);
  n := 0;
  s := 0;
  while s <= 365 do
  begin
    s := s + d;
    n := n + 5;
  end;
  write(n);
end.
```

Решение

В предыдущей задаче требовалось определить конечное значение вычисляемой переменной, а теперь наоборот, нужно найти по заданному конечному значению исходное.

1) Анализируем алгоритм:

в цикле ПОКА задано условие выполнения цикла — до тех пор, пока значение s не превышает 365;

внутри цикла значение переменной s увеличивается на заданное значение d, а значение переменной n увеличивается на 5;

по завершении цикла на экран выводится значение переменной n.

2) Тогда, согласно условию, значение n равно 55. Учитывая, что изначально переменная n обнуляется, это означает, что цикл выполнился $55/5 = 11$ раз.

3) Учитывая, что переменная s изначально тоже обнуляется, получаем, что за эти 11 проходов цикла переменная s увеличивается на $11 \cdot d$ и становится больше 365 (условие прекращения работы цикла). Тогда получаем неравенство:

$11d > 365$, откуда $d > 365/11$, т.е. $d > 33^2/_{11}$, или, учитывая, что числа — целые, $d > 34$.

4) Чтобы теперь оценить верхнюю границу интервала возможных значений d, надо учесть, что при слишком большом значении d цикл мог бы прерваться ещё после 10 проходов (а то и раньше). Значит, нам нужно определить значение d, на единицу меньшее, чем минимальное значение d, обеспечивающее выполнение 10 проходов цикла. В этом случае неравенство будет таким:

$10d > 365$, тогда $d > 365/10$, или $d > 36{,}5$.

5) Рассуждая аналогично п. 3, можно заключить, что минимальное значение d, при котором цикл завершится через 10 проходов, равно 37 (ближайшее целое число, большее 36,5). Нам же требуется значение d, на единицу меньшее. Это число 36.

Таким образом, диапазон возможных значений d, обеспечивающих выполнение 11 проходов цикла, составляет [34 .. 36], а искомое максимальное значение d, при котором будет выведено число 55, равно 36.

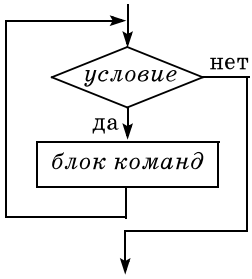
Ответ: 36.

Циклы: анализ алгоритмов

Конспект

Цикл с предусловием
(цикл ПОКА)

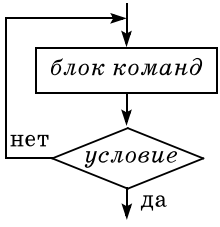
Обеспечивает выполнение блока команд, составляющего тело цикла, пока условие остаётся истинным (выход из цикла — по ложности условия).



Условие проверяется *до* начала выполнения цикла, поэтому возможна ситуация, что тело цикла не будет выполнено ни разу.

Цикл с постусловием
(цикл ДО)

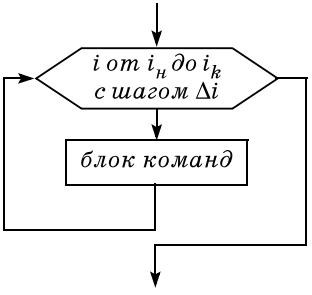
Обеспечивает выполнение команд, составляющих тело цикла, пока условие не станет истинным (цикл выполняется, пока условие ложно).



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.

Конструкции циклов в языке Паскаль

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>;

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. Ниже записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: a и b. Укажите наименьшее из таких чисел x, при вводе которых алгоритм печатает сначала 13, а потом 5.

```
var x, a, b, c: integer;
begin
  readln(x);
  a := 0;
  b := 10;
  while x>0 do
   begin
    c := x mod 10;
    a := a+c;
    if c<b then b := c;
    x := x div 10;
   end;
  writeln(a);
  write(b)
end.
```

Решение

Очевидно, здесь из числа x сначала выделяется последняя цифра, а в переменной a, таким образом, накапливается сумма цифр исходного числа, после чего в числе x очередная последняя цифра отбрасывается. То есть программа выполняет «разборку» заданного числа на отдельные цифры. Но здесь нет «обычного» счётчика проходов цикла. Зато появился оператор `if`.

В этом условном операторе выполняется проверка: если только что выделенная цифра меньше текущего значения b (а вначале так и будет: исходное $b = 10$ и больше, чем любая возможная десятичная цифра), то эта цифра перезапоминается в b. А дальше такое перезапоминание будет производиться, только если очередная (т.е. более «старшая» по разрядам) цифра числа x будет меньше, чем более «младшие» (запомненные в b ранее).

Теперь смотрим, что программа выводит на экран. Сначала это число a — сумма цифр числа x, равная 13. А потом — число b, указывающее что одна из этих цифр — цифра 5, причём она — наименьшая из всех цифр в числе x (однако количество этих цифр нам неизвестно).

Можно ли по этим данным определить число x? Да, можно! Если одна из цифр равна 5, то, вычтя её из суммы (13), мы получаем число 8. Это сумма всех остальных цифр числа x. Какими же они могут быть и сколько их может быть? Перебираем варианты:

- $1 + 7$ — невозможно: тогда в b была бы запомнена наименьшая по величине цифра 1, а не 5; то же касается и вариантов $2 + 6$, $3 + 5$, $4 + 4$ и всех им «симметричных» (например, $5 + 3$) — во всех них есть хотя бы одна цифра, меньшая 5;
- $1 + 1 + 6$, $1 + 2 + 5$ и все прочие возможные варианты с тремя цифрами: тоже невозможны по той же причине;
- очевидно, точно так же невозможны и любые другие варианты, в которых остаток суммы, равный 8, раскладывается на несколько цифр.

Следовательно, остаётся одна-единственная возможность: число x — двузначное и содержит цифры 5 и 8.

А какое из подобных чисел — наименьшее? Очевидно, 58 (а не 85)!

Ответ: 58.

Задача 2. Имеется алгоритм. Получив на вход число x, он печатает два числа a и b. Нужно указать наибольшее из чисел x, при вводе которого алгоритм напечатает сначала 2, а потом 26.

```
var x, a, b: integer;
begin
    readln(x);
    a : = 0; b : = 0;
    while x > 0 do
        begin
            a := a+1;
            b := b + (x mod 100) ;
            x := x div 100;
    end;
writeln(a);
write(b);
end.
```

Решение

Вновь проанализируем алгоритм. В операциях mod и div вторым операндом является число 100, а не 10. Следовательно, теперь в цикле обрабатываются не отдельные цифры исходного числа, а числа, составленные из пар таких цифр.

Значение переменной *a* равно 2. Значит, в исходном числе (раз оно разбивается на пары) может быть или четыре цифры, или три (вторая по счёту выделяемая пара — неполная). Нам, по условию, нужно искать наибольшее из возможных чисел — значит, нужно пытаться брать в рассмотрение число из наибольшего количества цифр — из четырёх.

Сумма числа, которое составлено из двух последних цифр, и числа, составленного из первых двух цифр, равна 26. Но максимально возможная сумма десятичных цифр — это 18 (9 + 9). Нам нужно получить как можно большее возможное значение x, а значит, наибольшие цифры должны стоять в нём в начале числа. Предположим, что эти первые цифры и есть две девятки. Тогда второе число из пары цифр будет равно 26 − 18 = 8.

Задача практически решена. Остаётся записать два найденных двузначных числа так, чтобы составить из них требуемое число x: 9908 (не забываем во второй паре дописать нуль, чтобы превратить «одинокую» восьмёрку в двузначное число).

Ответ: 9908.

Задача 3. Ниже на пяти языках записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа *a* и *b*. Укажите *наибольшее* из таких чисел x, при вводе которых алгоритм печатает сначала 4, а потом 20.

```
var x, a, b: integer;
begin
  readln(x);
  a := 0; b := 0;
  while x > 0 do
```

```
  begin
    a := a + 1;
    if x mod 2 = 0 then
      b := b + x mod 10;
    x := x div 10;
  end;
  writeln(a); write(b);
end.
```

Решение

Анализируем алгоритм.

Очевидно, что переменная *a* представляет собой счётчик количества проходов цикла.

В самом цикле сначала выполняется проверка чётности числа (т.е. фактически — чётности его последней цифры). Если число (и последняя его цифра) чётно, то последняя цифра числа выделяется (`x mod 10`) и прибавляется к переменной *b*. После этого, уже вне условного оператора, из числа *x* отбрасывается последняя цифра (`x div 10`). Таким образом, в программе выполняется подсчёт суммы чётных цифр введённого числа.

Поскольку в результате на экран выводится пара чисел 4 и 20, делаем вывод, что количество цифр в исходном числе (определяющее количество проходов цикла) равно 4, а сумма чётных цифр в этом числе равна 20.

Остаётся найти наибольшее из возможных четырёхзначных чисел, в котором сумма чётных цифр равна 20.

Вспоминаем правило: в большем числе цифры расположены по убыванию.

Поэтому нам «выгодно» выделить в составе суммы 20 одно число — чётное и соответствующее наибольшему из возможных значению чётной десятичной цифры (очевидно, это 8). Оставшуюся часть суммы (12) проверяем: может ли она соответствовать какой-либо чётной цифре. В нашем случае она превышает максимально возможное значение чётной цифры (8). Поэтому мы снова выделяем из этой суммы чётное и наибольшее из воз-

можных значение 8. Остаётся число 4. Таким образом, мы разбиваем сумму 20 на три слагаемых: 8+8+4.

Таким образом, мы определили три чётные цифры, входящие в исходное число. Оставшаяся четвёртая цифра должна быть нечётной. И поскольку нам требуется наибольшее исходное число из возможных, а нечётные цифры на подсчёт суммы не влияют, мы можем выбрать эту цифру равной 9.

Итак, мы имеем четыре цифры: 8, 8, 4 и 9. Остаётся расположить их по убыванию, чтобы получить число 9884. Оно и будет наибольшим из возможных, которое в результате работы указанной программы даст на экране значения 4 и 20.

Ответ: 9884.

Задача 4. Ниже на пяти языках записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа a и b. Укажите *наименьшее* из таких чисел x, при вводе которых алгоритм печатает сначала 6, а потом 18.

```
var x, a, b: integer;
begin
  readln(x);
  a := 0; b := 0;
  while x > 0 do
  begin
    a := a + 1;
    if x mod 2 = 0 then
      b := b + x mod 10;
    x := x div 10;
  end;
  writeln(a); write(b);
end.
```

Решение

Как и в предыдущей задаче, алгоритм — тот же самый: переменная a — это счётчик количества проходов цикла и, следовательно, количество цифр в исходном чис-

ле (6 прохода — четырёхзначное число), а в переменной b подсчитывается сумма чётных цифр числа. Однако теперь нужно найти наименьшее из возможных четырёхзначных чисел, в котором сумма чётных цифр равна 18.

Вспоминаем правило: в меньшем числе цифры расположены по возрастанию, при этом самая первая цифра должна быть ненулевая, наибольшие цифры должны быть «сгруппированы» в конце числа (справа), а начальные цифры (кроме самой первой) «выгодно» заполнять нулями.

Наша сумма (18) превышает возможное наибольшее значение чётной цифры (8). Пробуем выделить из этой суммы слагаемое, равное 8 (которое возьмём в качестве последней цифры). Остаётся 10. Это число тоже превышает возможное наибольшее значение чётной цифры, поэтому из него мы снова выделяем слагаемое 8 (это будет предпоследняя цифра числа). Остаётся число 2, которое может являться чётной десятичной цифрой. Таким образом, мы разбили сумму 18 на три слагаемых: 8+8+2.

Три цифры исходного числа нам известны. Осталось определить ещё две. Одну из них берём ненулевую, наименьшую возможную нечётную — это единица. Она в создаваемом числе будет самой первой, а на подсчёт суммы не влияет. Ещё одну цифру «заполняем» нулём.

Остаётся записать полученные цифры (8, 8, 2, 1, 0) в таком порядке, чтобы получилось наименьшее из возможных пятизначных чисел. Это число 10288.

Ответ: 10288.

Задача 5. Имеется алгоритм. Получив на вход число x, он печатает два числа a и b. Нужно указать наименьшее из таких чисел x, при вводе которого алгоритм напечатает сначала 2, а потом 3.

```
var x, a, b, c: integer;
begin
    readln(x);
    a := 0;
```

```
    b := 0;
    while x > 0 do
    begin
        c := x mod 2
        if c = 0 then
            a := a + 1
        else
            b := b + 1
        x := x div 10
    end;
    writeln(a);
    write(b);
end.
```

Решение

Здесь алгоритм совершенно другой. Подсчёт количества проходов цикла не ведётся вообще. В переменную c записывается остаток от деления текущего значения числа x на 2, а затем если $c = 0$, то на единицу увеличивается значение a, а если $c = 1$, то на единицу увеличивается значение b (которые в итоге и выводятся на экран). А после этого у исходного числа x отбрасывается последняя цифра.

Нетрудно догадаться, что данный алгоритм, последовательно выделяя из исходного числа отдельные цифры, определяет чётность каждой такой цифры и подсчитывает в переменной a количество чётных, а в переменной b — количество нечётных цифр. По условию, чётных цифр в этом числе — 2, а нечётных — 3.

Не менее очевидно, что сумма итоговых значений a и b даёт нам общее количество цифр в числе: $2 + 3 = 5$.

Итак, нам нужно найти наименьшее из возможных пятизначных чисел, в котором две чётные цифры и три нечётные. Какие это могут быть цифры и как они расположены в числе?

Очевидно, что для получения наименьшего возможного числа надо брать и наименьшие возможные цифры. Наименьшая нечётная цифра — это единица. А наимень-

шая чётная? Если учесть, что остаток от деления нуля на 2 тоже равен нулю, то в качестве наименьшей чётной цифры у нас в таких задачах выступает нуль.

Итак, в числе x должно быть три единицы и два нуля. В каких разрядах их расположить? Самой первой цифрой, очевидно, должна стоять ненулевая — то есть в старшем разряде будет единица. А остальные цифры располагаем по правилу: в наименьшем возможном числе цифры по разрядам нужно располагать по возрастанию слева направо (чтобы в старших разрядах, «вес» которых максимален, оказывались самые маленькие цифры). Значит, после «лидирующей» обязательной единицы (которая определяет пятизначность числа) сначала надо записать оба нуля, а в конце — две оставшиеся единицы. Получим число 10011 (обратим внимание — десятичное).

Ответ: 10011.

Задача 6. Имеется алгоритм. Получив на вход число x, он печатает два числа a и b. Нужно указать наибольшее из таких чисел x, при вводе которого алгоритм напечатает сначала 3, а потом 4.

```
var x, a, b, c: integer;
begin
    readln(x);
    a := 0;
    b := 0;
    while x > 0 do
    begin
        c := x mod 2
        if c = 0 then
            a := a + 1
        else
            b := b + 1
        x := x div 10
    end;
    writeln(a);
     write(b);
end.
```

Решение

Нетрудно найти, что в искомом числе x имеется три чётные цифры и четыре нечётные, а само число — семизначное.

Теперь нам требуется найти наибольшее возможное число, значит и цифры для него надо выбирать наибольшие из возможных. Самая большая нечётная цифра — это 9, а самая большая чётная цифра — 8.

Остаётся только правильно записать эти три восьмерки и четыре девятки. В наибольшем числе из возможных цифры записываются слева направо по убыванию (чтобы в старших разрядах, «вес» которых максимален, оказывались самые большие цифры). Тогда искомое число — 9999888.

Ответ: 9999888.

Задача 7. Имеется алгоритм. Получив на вход число x, он печатает два числа a и b. Нужно указать наибольшее из таких чисел x, при вводе которого алгоритм напечатает сначала 2, а потом 3.

```
var x, a, b, c: integer;
begin
    readln(x);
    a := 0;
    b := 0;
    while x > 0 do
    begin
        c := x mod 2
        if c = 0 then
            a := a + 1
        else
            b := b + 1
        x := x div 8
    end;
    writeln(a);
    write(b);
end.
```

Решение

Смотрим на алгоритм. Вместо оператора $x := x \text{ div } 10$, который, как мы знаем, отбрасывает в десятичном числе последнюю цифру, у нас записано $x := x \text{ div } 8$.

Первая идея, которая приходит в голову, — это пытаться отслеживать кратность при каждом очередном делении на 8 и исходя из этого пытаться определить количество цифр. Но, как легко догадаться, такая интерпретация представленного алгоритма слишком сложна и малонаглядна.

А что, если мысленно представить исходное число x так, чтобы и при выполнении операции div 8 в числе отбрасывалась последняя цифра? Это вполне возможно, если число x понимать как записанное в восьмеричной системе счисления!

Вот это и есть «ключ» к решению любых задач такого типа (где в операторе $x := x \text{ div } \ldots$ стоит не 10, а другое число). Остальная же часть решения подобна рассмотренным выше.

По условию, в числе x (которое мы рассматриваем в восьмеричном формате!) две чётные цифры и три нечётные. Значит, всего их пять. Число требуется наибольшее из возможных — значит, цифры надо брать наибольшие из возможных и располагать их в числе по правилу «большие — слева».

Самая большая чётная восьмеричная цифра — это 6. Самая большая нечётная восьмеричная цифра — 7. Тогда наибольшее пятизначное число, составленное из этих цифр — это 77766.

Остаётся только вспомнить, что мы рассматривали и «расписывали» число в восьмеричном виде, а в задаче-то имелось в виду десятичное число x. Значит, нужно восьмеричное значение 77766 преобразовать в десятичный формат: $77766_8 = 32758_{10}$.

Ответ: 32758.

Задача 8. Считав число *x*, приведённая ниже программа печатает два числа: *a* и *b*. Укажите **наименьшее** из чисел *x*, при вводе которого программа напечатает сначала 2, а потом 4.

```
var x, a, b: longint;
begin
  readln(x);
  a:= 0;
  b:= 0;
  while x > 0 do begin
    if x mod 2 = 0 then a:= a + 1
    else b:= b + x mod 10;
    x:= x div 10;
  end;
  writeln(a);
  write(b);
end.
```

Решение

1. Анализируем программу:

• цикл с условием `while x > 0 do` и выполняемый в цикле оператор `x:= x div 10` — это хорошо знакомый учащимся алгоритм «разбора» числа на отдельные цифры справа налево (от младших разрядов к старшим);

• операция `if x mod 2 = 0 then a:= a + 1` при помощи `mod` определяет, является ли последняя цифра числа *x* чётной, и если да, то счётчик *a* увеличивается на 1, т.е. в переменной *a* определяется количество чётных цифр в числе *x*;

• если же последняя цифра числа нечётна, то срабатывает ветвь `else` и в переменной *b* вычисляется сумма таких нечётных цифр.

2. Если программа выводит сначала число 2, а затем число 4, то это означает, что в числе *x* было две чётные цифры (причём нули тоже считаются чётными цифрами), но о количестве нечётных цифр ничего не говорит-

ся (!), а сумма нечётных цифр равна 4 (но чётные цифры в подсчёт не входят!).

3. Нам нужно найти наименьшее такое число. Значит, требуется:

- минимально возможное количество нечётных цифр, сумма которых равна 4;
- две чётные, но наименьшие из возможных цифры (допустимо — два нуля, с условием, что они не могут стоять в начале числа);
- наибольшие цифры в числе нужно размещать правее.

4. Сумму 4 надо представить как сложение двух нечётных цифр, — это можно сделать только одним способом: 1 + 3.

5. Тогда наименьшее возможное число — 1003.

Ответ: 1003.

Если требовалось бы найти наибольшее возможное число, то для этого нужно было бы:

- максимально возможное количество нечётных цифр, сумма которых равна 4;
- две чётные, но наибольшие из возможных цифры;
- наибольшие цифры надо в числе размещать левее.

Число 4 в данном случае можно разложить на нечётные цифры так: 1 + 1 + 1 + 1. А наибольшие возможные чётные цифры — это 88. Тогда искомое число — это 881111.

Задача 9. 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 3, а потом число 16.

```
var x, a, b: integer;
  begin
  readln(x);
  a := 0;
  b := 1;
  while x > 0 do
```

```
    begin
      a:= a + 1;
      b:= b + (10 - (x mod 10));
      x:= x div 10;
    end;
    writeln(a);
    write(b);
end.
```

Решение

1. Проанализировав текст программы, нетрудно понять, что алгоритм обработки в ней заложен самый простой: исходное число раскладывается на десятичные цифры; переменная a — это счётчик количества проходов цикла, т.е. значение a определяет, сколько цифр было в изначальном числе; переменная же b — это сумма..., но чего именно?

2. Раньше в таких задачах суммировались получаемые цифры. Теперь же к b прибавляется значение `(10 - (x mod 10))`, которое, очевидно, представляет собой дополнение очередной цифры исходного числа до 10. То есть, например, если очередная цифра равна 3, то к b прибавляется число 7.

3. По условию, программа выводит сначала число 3, а затем — число 16. Значит, после выполнения программы $a = 3$, а $b = 16$. Следовательно, исходное число содержало 3 цифры. Но как определить эти цифры по заданному значению b?

Пусть искомые три цифры равны соответственно x, y и z (считая справа налево, в том порядке, в каком они выделяются из числа). Тогда получаемое значение b равно:

$b = (10 - x) + (10 - y) + (10 - z) = 10 + 10 + 10 - x - y - z = 30 - (x + y + z)$, откуда искомую сумму цифр $(x + y + z)$ можно определить как $30 - b$ (получаемое при работе программы значение b нужно вычесть из 10, умноженного на количество цифр в числе, т.е. на значение a). Итоговая формула: $сумма\ цифр = 10 \times a - b$.

4. Следовательно, надо найти минимально возможное трёхзначное число, сумма цифр которого равна 30 – 16 = 14. Удобнее всего разложить эту сумму на две цифры, из которых одна наименьшая из возможных, т.е. как 9 + 5, и записать эти цифры по возрастанию (большая цифра — слева). А между ними добавить в качестве третьей цифры минимально возможную цифру 0.

Ответ: 509.

Задача 10. 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 2, а потом число 6.

```
var x, a, b: longint;
begin
  readln(x);
  a := 0; b := 0;
  while x > 0 do begin
    if x mod 2 > 0 then a := a + 1
      else b := b + x mod 6;
    x := x div 6;
  end;
  writeln(a); write(b);
end.
```

Решение

1. Анализируя текст программы, видим:

- a и b — это переменные-счётчики,
- исходное число разбивается на цифры в шестеричной системе,
- если число нечётно (последняя цифра в шестеричной системе нечётна), то $a = a + 1$, следовательно, a — это количество нечётных шестеричных цифр,
- иначе к b прибавляется сама очередная чётная шестеричная цифра, т.е. в b накапливается сумма чётных шестеричных цифр.

2. Выводятся значения $a = 2$, $b = 6$. Значит, в числе есть две нечётные шестеричные цифры, а сумма чётных шестеричных цифр равна 6.

3. Число требуется найти наименьшее как по модулю, так и по количеству разрядов.

Поэтому две нечётные цифры берём наименьшие из возможных — две единицы.

Число 6 же представим как сумму чётных шестеричных цифр 2 и 4 (так как цифра 6 невозможна, а другие варианты дают больше разрядов).

4. В результате имеем цифры 1, 1, 2, 4. Минимально возможное составленное из них число — это 1124 в шестеричной записи.

Переводим это шестеричное число в десятичную систему счисления:

$1124_6 = 6^3 + 6^2 + 2 \times 6 + 4 = 268$.

Ответ: 268

Задача 11. Определить наибольшее двузначное натуральное число, при вводе которого программа выведет на экран число 0.

```
var i, n: longint;
begin
  i := 0;
  readln(n);
  while (n > 0) do
  begin
    i := i + n mod 8;
    n := n div 8;
  end;
  writeln(i mod 7);
end.
```

Решение

1. Анализируем алгоритм:

1) судя по числовым значениям в операциях `div` и `mid`, введённое число разбирается на цифры в восьмеричной системе;

2) в переменной *i* накапливается сумма выделенных из числа восьмеричных цифр;

3) выводится на экран не сама эта сумма, а остаток от её деления на 7, и согласно условию, этот остаток должен быть равен нулю.

Итак, требуется найти **наибольшее двузначное десятичное число, у которого сумма цифр в восьмеричной записи кратна 7**.

2. Весь диапазон двузначных десятичных чисел — от 10 до 99. Поскольку нам нужно наибольшее возможное число, просмотр этого диапазона мы начинаем с 99 и далее по убыванию.

Построим таблицу:

Десятичное число	Восьмеричная запись	Сумма восьмеричных цифр	Остаток от деления суммы на 7
99	143	8	1
98	142	7	**0**

Итак, мы нашли такое двузначное десятичное число *n*, для которого сумма цифр восьмеричной записи кратна 7. Это найденное число и есть ответ.

Ответ: 98

Очевидно, что переводить десятичное число в другую систему счисления (в данном случае в восьмеричную) нужно только один раз. Далее мы каждый раз уменьшаем десятичное значение на единицу, и соответствующее восьмеричное значение тоже достаточно уменьшать на 1. Только нужно при этом помнить правила вычитания в соответствующей системе счисления: например, в восьмеричной системе $140_8 - 1 = 137_8$.

Задача 12. Определить наименьшее четырёхзначное натуральное число, при вводе которого программа выведет на экран число 0.

```
var i, n: longint;
begin
   i:= 0;
```

```
    readln(n);
    while (n > 0) do
    begin
      i:= i + n mod 16;
      n:= n div 16;
    end;
    writeln(i mod 15);
end.
```

Решение

Задача похожа на предыдущую.

1. Анализ алгоритма:

- введённое число разбирается на цифры в шестнадцатеричной системе;
- в переменной *i* накапливается сумма выделенных из числа шестнадцатеричных цифр;
- выводится остаток от деления этой суммы на 15, и согласно условию, этот остаток должен быть равен нулю.

Вывод: требуется найти **наименьшее четырёхзначное десятичное число, у которого сумма цифр в шестнадцатеричной записи кратна 15**.

2. Весь диапазон четырёхзначных десятичных чисел — от 1000 до 9999. Нам нужно наименьшее возможное число, поэтому просмотр этого диапазона начинаем с 1000 и далее по возрастанию.

Строим таблицу (помним, чему равны значения шестнадцатеричных цифр: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15):

Десятичное число	Шестнадцатеричная запись	Сумма шестнадцатеричных цифр	Остаток от деления суммы на 15
1000	3E8	3 + 14 + 8 = 25	10
1001	3E9	3 + 14 + 9 = 26	11
1002	3EA	3 + 14 + 10 = 27	12
1003	3EB	3 + 14 + 11 = 28	13
1004	3EC	3 + 14 + 12 = 29	14
1005	3ED	3 + 14 + 13 = 30	**0**

Мы нашли такое четырёхзначное десятичное число n, для которого сумма цифр шестнадцатеричной записи кратна 15. Это найденное число — и есть ответ.

Ответ: 1005.

Задача 13. Определить количество двухзначных натуральных чисел, при вводе которых программа выводит на экран число 0.

```
var i, n: longint;
begin
   i:= 0;
   readln(n);
   while (n > 0) do
   begin
      i:= i + n mod 5;
      n:= n div 5;
   end;
   writeln(i mod 4);
end.
```

Решение

Эта задача немного сложнее. Здесь потребуется просматривать весь диапазон допустимых чисел.

1. Анализ алгоритма:

- введённое число разбирается на цифры в пятеричной системе;
- в переменной i накапливается сумма выделенных из числа пятеричных цифр;
- выводится остаток от деления этой суммы на 4, и согласно условию, этот остаток должен быть равен нулю.

Вывод: требуется найти **количество двухзначных десятичных чисел, у которых сумма цифр в пятеричной записи кратна 4.**

2. Весь диапазон двухзначных десятичных чисел — от 10 до 99. Начнём просмотр этого диапазона с 10 и далее по возрастанию.

Начнём строить таблицу, но, в отличие от предыдущих двух задач, найдя нулевой остаток от деления, продолжаем запись:

Десятичное число	Пятеричная запись	Сумма пятеричных цифр	Остаток от деления суммы на 4
10	20	2	2
11	21	3	3
12	22	4	**0**
13	23	5	1
14	24	6	2
15	30	3	3
16	31	4	**0**
17	32	5	1
18	33	6	2
19	34	7	3
20	40	4	**0**
21	41	5	1
22	42	6	2
23	43	7	3
24	44	8	**0**
25	50	5	1
...			

Нетрудно заметить закономерность: впервые нулевой остаток встречается для $n = 12$, а далее ситуация повторяется для каждого четвёртого числа. Остаётся выписать все такие двузначные числа в ряд, каждый раз увеличивая значение на 4, и подсчитать их количество:

12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96 (а число 100 уже нам не годится — оно уже трёхзначное!).

Количество этих чисел равно 22. Это и есть ответ.

Ответ: 22.

Задача 14. Укажите наименьшее из чисел x, при вводе которого программа выведет на экран пятизначное число.

```
var x, k, m, n: integer;
begin
  readln(x);
  m:= x;
  n:= 0;
  while x > 0 do begin
    k:= x mod 4;
    n:= 10*n + k;
    x:= x div 4
  end;
  n:= n + m;
  writeln(n);
end.
```

Решение

1. Начнём с анализа алгоритма:

- введённое число x запоминается в переменной m,
- значение n изначально равно нулю,
- в цикле сначала от числа справа отделяется очередная цифра (причём в **четверичной** системе счисления — см. разбор предыдущих задач), затем эта цифра прибавляется к n, а после этого обработанная цифра четверичной записи числа отбрасывается,
- после завершения цикла к n ещё прибавляется ранее запомненное в переменной m исходное число x.

При этом очередная выделенная цифра четверичной записи прибавляется к предыдущему значению n, умноженному на 10. К чему это приводит?

Попробуем рассмотреть работу этого фрагмента на примере.

Пусть $x = 39_{10}$, что в четверичной системе счисления равно 213_4. Выполняем трассировку алгоритма для четверичной записи числа.

		x	n	k
Исходные значения		213	0	–
1-й проход цикла	`k:=x mod 4`	213	0	**3**
	`n:=10*n + k`	213	$0 \times 10 + 2 = \mathbf{3}$	3
	`x:=x div 4`	**21**	3	3
2-й проход цикла	`k:=x mod 4`	21	3	**1**
	`n:=10*n + k`	21	$3 \times 10 + 1 = \mathbf{31}$	1
	`x:=x div 4`	**2**	31	1
3-й проход цикла	`k:=x mod 4`	2	31	**2**
	`n:=10*n + k`	2	$31 \times 10 + 2 = \mathbf{312}$	2
	`x:=x div 4`	**0**	312	2
Цикл завершён				

Итак, для исходного числа 39_{10}, равного 213_4, мы получили в итоге n, равное 312_4.

Следовательно, рассматриваемый алгоритм (три рассмотренных оператора в цикле), по сути, определяет четверичное представление исходного десятичного числа, а затем переписывает эти четверичные цифры в обратном порядке (!) уже как цифры нового десятичного числа n.

2. По условию, нам нужно найти наименьшее возможное число x, которое при таком преобразовании даёт пятизначное число n. Но при таком преобразовании наименьшее значение x даст и наименьшее значение n.

Поэтому найдём сначала наименьшее возможное значение n, соответствующее условию задачи, т. е. пятизначное, но без учёта последнего оператора `n:= n + m;`.

Можно предположить, что это число 10000. Но вспомним, что рассматриваемый алгоритм «переворачивает» исходное число справа налево (меняет порядок цифр на обратный). И для числа n, равного 10000, исходное четверичное представление числа x тогда оказалось бы равным 00001, или просто 1. А в этом случае значение n не получилось бы пятизначным.

Поэтому минимальное значение n, которое нам подходит, равно 10001. Тогда (учитывая его «симметричность») такое четверичное представление соответствует исходному десятичному числу x, равному: $4^4 + 1 = 257_{10}$.

Итак, для исходного значения x, равного 257, по завершении выполнения цикла n будет равно 10001, и это действительно минимально возможное пятизначное число. Но у нас ещё остаётся не рассмотренным оператор `n := n + m;`. После его выполнения $n = 10001 + 257 = 10258$, и это число остаётся пятизначным.

Но теперь нужно ещё проверить: не могло ли быть так, что ещё меньшее исходное значение x по завершении цикла давало четырёхзначное значение n, и оно становилось пятизначным уже в результате прибавления к нему самого исходного числа.

Поэтому повторяем наше решение, выбрав меньшее значение n. А поскольку брать $n = 10000$ мы не можем (ранее было уже указано, почему), то мы берём предыдущее ему значение $n = 3333$.

Для такой четверичной записи исходное значение $x = 3{\times}4^3 + 3{\times}4^2 + 3{\times}4 + 3 = 255$. Проверим, станет ли сумма предполагаемого числа n и исходного значения x пятизначной? $3333 + 255 = 3588$. Нет, число осталось только четырёхзначным. Значит, верным является наше первое предположение, и минимально возможное значение x, дающее в итоге пятизначное значение n, равно 257.

Ответ: 257.

 В данной задаче завершающий оператор `n := n + m;` не повлиял на решение. Но выполнять описанную выше дополнительную проверку всё-таки нужно. Скажем, если бы «финальным» оператором было `n := n + 27*m;`, то у нас действительно бы четырёхзначное число 3333 после такого суммирования становилось бы пятизначным, и правильным ответом было бы уже $x = 255$, а не 257.

Задача 15. Укажите наименьшее число x, для которого программа выводит число, большее 10200.

```
var x, k, m, n: integer;
begin
   readln(x);
   m:= x; n:= 0;
   while x > 0 do begin
      k:= x mod 5;
      n:= 10*n + k;
      x:= x div 5
   end;
   n:= n + m;
   writeln(N);
end.
```

Решение

1. Анализ алгоритма:

- введённое число x запоминается в переменной m,
- значение n изначально равно нулю,
- в цикле сначала от числа справа отделяется очередная цифра (в **пятеричной** системе счисления), эта цифра прибавляется к n, а затем обработанная цифра пятеричной записи числа отбрасывается,
- после завершения цикла к n ещё прибавляется ранее запомненное в переменной m исходное число x.

При этом очередная выделенная цифра пятеричной записи прибавляется к предыдущему значению n, умноженному на 10, т.е. происходит переписывание пятеричных цифр уже как десятичных и в обратном порядке.

2. Основная идея решения — в том, что хотя к полученному числу n и прибавляется исходное число x, но из-за того, что пятеричная запись более «громоздка» (требует больше цифр), а число x ищется минимальное, мы предполагаем, что этот «довесок» будет не очень большим.

3. Число, с которым в условии нужно сравнивать результат (10200) — пятизначное, а x, согласно нашему предположению, много к этому числу не добавляет. Значит, получаемое число n тоже должно быть пяти-

значным. (Можно это проверить. Наибольшее возможное четырёхзначное число, состоящее из пятеричных цифр, равно 4444. При этом $4444_5 = 624_{10}$. В этом случае получаемое значение $n = 4444 + 624 = 5068 < 10200$. И всё меньшие значения x тоже можно не проверять — они ещё меньше.)

4. Пробуем взять первое (самое маленькое) допустимое пятизначное число — 10001 (число 10000 не подходит, так как при его «переворачивании» — записи в обратном порядке — получилось бы «00001», т.е. не пятизначное число 1).

Число 10001_5 в десятичном представлении равно $5^4 + 1 = 626_{10}$.

В «перевёрнутой» записи тоже получается число 10001, на этот раз понимаемое как десятичное.

$10001 + 626 = 10627 > 10200$, что и требовалось.

Ответ: 626.

Задача 16. Укажите наименьшее число x, для которого программа выводит число, большее 10300.

```
var x, k, m, n: integer;
begin
   readln(x);
   m:= x; n:= 0;
   while x > 0 do begin
      k:= x mod 4;
      n:= 10*n + k;
      x:= x div 4
   end;
   n:= n + m;
   writeln(N);
end.
```

Решение

1. Анализ алгоритма показывает, что отличие от предыдущей задачи только в том, что рассматривается представление исходного числа в четверичной системе счисления.

2. Так же, как и в прошлый раз, предполагаем, что добавление в конце программы числа x значение числа n меняет не сильно.

3. Число, с которым в условии нужно сравнивать результат (10300) — пятизначное, тогда получаемое число n тоже пятизначное. (Проверяем: $4444 + 255 = 4699 < 10300$.)

4. Если не учитывать прибавления x, то можно взять наименьшее значение n, большее 10300, равным 10301. Тогда «перевёрнутая» запись числа равна 10301, и если считать её за пятеричное число, то получим $x = 4^4 + 3*4^2 + 1 = 305$.

5. Конечно, это — ещё не ответ. «Правильное» число x будет меньше, поскольку получаемое число должно быть больше заданного 10300 только уже после прибавления x. Поэтому теперь перебором ищем наименьшее значение, которое удовлетворяет условиям задачи. Для этого строим таблицу, в которой сначала записываем итоговое (уже «перевёрнутое») пятизначное число, затем его исходную записи до «переворачивания», потом десятичный эквивалент этой записи, считая её четверичной, а в заключение — вычисленное итоговое значение n.

Предполагаемое «перевёрнутое» десятичное число n (до прибавления x)	**«Исходное» четверичное число n**	**x**	**Сумма «перевёрнутого» десятичного n и исходного x**	**Сумма больше 10300?**
10300	*Недопустимо (при «переворачивании» не получается пятизначное число)*			
10233	33201	993	11226	да
10232	23201	737	10969	да
10231	13201	481	10712	да
10230	*Недопустимо (при «переворачивании» не получается пятизначное число)*			
10223	32201	929	11152	да
10222	22201	673	10895	да
10221	12201	417	10638	да

Уже сейчас можно заметить:

• число в левой колонке не должно заканчиваться нулём,

• наименьшие значения получаются, если число в левой колонке заканчивается единицей, а не цифрой 2 или 3.

Поэтому далее заполняем таблицу только для таких чисел.

10211	11201	353	10564	да
10201	10201	289	10490	да
10131	13101	465	10596	да
10121	12101	401	10522	да
10111	11101	337	10448	да
10101	10101	273	10374	да
10031	13001	449	10480	да
10021	12001	385	10406	да
10011	11001	321	10332	да
10001	10001	257	10258	**нет**

Итак, мы нашли значение x, для которого в последней графе получается «нет».

Очевидно, брать ещё меньшее число бесполезно: значение 10000 недопустимо, а далее уже получаются четырёхзначные числа, которые, как мы уже показали в начале решения, нам непригодны.

Попробуем взять в качестве x число, на 1 большее найденного «критического» значения 257. Проверяем: $258_{10} = 10002_4$; «переворачиваем» — получаем 20001; прибавляем исходное число 258 — получаем 20259 $>$ $>$ 10300.

Ответ: 258.

Задача 17. Укажите наименьшее такое число x, большее 100, при вводе которого программа выводит число 15.

```
var x, L, M: integer;
begin
  readln(x);
  L := x - 30; M := x + 30;
  while L <> M do
    if L > M then
      L := L - M
    else
      M := M - L;
  writeln(M);
end.
```

Решение

Вообще говоря, речь идёт об алгоритме Евклида для вычисления НОД двух чисел. Однако решить задачу можно и просто, исходя из общего анализа алгоритма.

Из условия цикла `while` понятно, что значение M, по условию равное 15, выводится, когда оба числа (и L, и M) равны друг другу.

Тогда очевидно, что на предыдущем шаге цикла какое-то из чисел было равно $15 + 15 = 30$. Именно потому после вычитания из большего значения (30) меньшего (15) в результате мы получили тоже 15.

Ещё на один шаг раньше большее число могло быть ещё на 15 больше, т.е. $30 + 15 = 45$.

Повторяя эти рассуждения и «двигаясь» от конца вычислений к началу, прибавляем к большему числу по 15: $45 + 15 = 60$; $60 + 15 = 75$; $75 + 15 = 90$; $90 + 15 = 105$. Однако 105 — это и есть наименьшее число, большее 100. Значит, возможно, это и есть ответ.

Проверяем этот ответ, учитывая, что для $x = 105$ значение $L = 105 - 30 = 75$, а $M = 105 + 30 = 135$: вычисляем НОД (75, 135). Он равен 15. Следовательно, наше решение верно.

Ответ: 105.

Если при проверке предполагаемого ответа выяснится, что получается не то значение НОД, которое нужно по условию, то мы просто берём следующее по порядку значение

в нашей цепочке. В данном случае — прибавляем к предполагавшемуся значению 105 ещё 15, получая $x = 120$, и для него снова проверяем НОД вычисленных значений L и M.

Задача 18. Получив на вход число x, этот алгоритм печатает число M. Известно, что $x > 100$. Укажите наименьшее такое (т.е. большее 100) число x, при вводе которого алгоритм печатает 11.

```
var x, L, M: integer;
begin
  readln(x);
  L:= x - 21;
  M:= x + 12;
  while L M do
   if L > M then
    L:= L - M
   else
    M:= M - L;
  writeln(M);
end.
```

Решение

Главное отличие здесь — в том, что перед началом цикла к исходному числу прибавляются и из него вычитаются **разные** значения. В этом случае ход решения несколько иной.

1. Известно, что в результате оба числа — и L, и M — равны 11, и это число — есть НОД исходных чисел L и M. Следовательно, изначально оба они должны делиться на 11.

2. Как и в предыдущем случае, выпишем все возможные значения L, начиная с заданного числа 11 и каждый раз прибавляя это число 11, пока не получим первое же значение, превышающее 100:

11; 22; 33; 44; 55; 66; 77; 88; 99; 110.

3. Если, согласно программе, $L = x - 21$, то $x = L + 21$. Выбираем из только что выписанного ряда такое значе-

ние L, что при прибавлении к нему числа 21 получается наименьшее из возможных чисел, бóльших 100. Если мы возьмём 77, то $77 + 21 = 98$ — мало, а вот $88 + 21 = 109$ — это то, что нам надо. Тогда число 109 — это предполагаемое значение x.

4. Опять же, согласно программе, $M = x + 12$. Тогда предполагаемое значение $M = 109 + 12 = 121$.

5. Для полученной пары чисел $L = 88$ и $M = 121$ вычисляем значение НОД: оно равно 11. Следовательно, наше решение верно, и ответом является $x = 109$.

Ответ: 109.

Если бы на пятом шаге решения мы получили бы другое значение НОД, то нужно было бы взять из ранее записанного ряда следующее, большее, значение L (99), повторить для него вычисление предполагаемых значений x и M, определить для полученных L и M значение НОД и сверить с заданным, и так — до нахождения нужного x.

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 поочерёдную 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 максимумом/минимумом, и если этот элемент больше/меньше предполагаемого максимума/минимума, то значение этого элемента запоминается в качестве нового предполагаемого максимума/минимума. До-

полнительно 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 во 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).

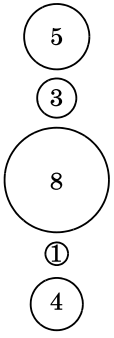
Сортировка массива

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 бóльшими значениями; при сортировке по убыванию — наоборот).

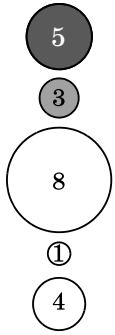
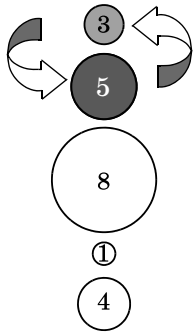
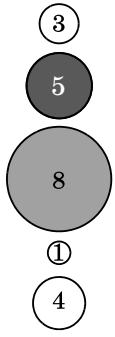
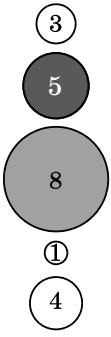
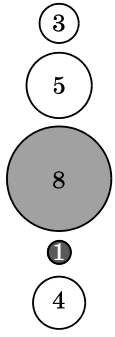
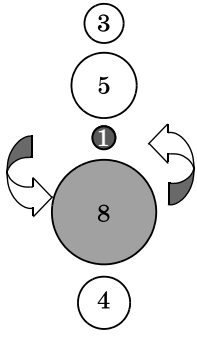
Существует несколько методов сортировки массивов, различающихся сложностью их алгоритма и скоростью работы: сортировка перемешиванием, сортировка выбором, сортировка вставками, быстрая сортировка, пирамидальная, «гномья», слиянием, деревом, сортировка Шелла, но наиболее понятной и наглядной является **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.

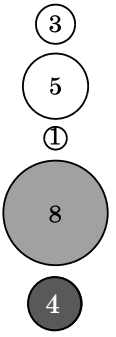
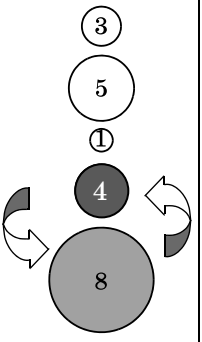


Если бы в условии было всего два элемента (например, числа 5 и 3, с которых начинается наш массив), то достаточно было бы сравнить их друг с другом, и если первое из этих чисел больше второго, то поменять их местами.

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 таблицы

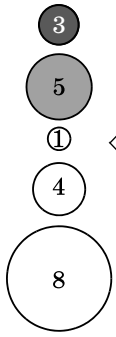
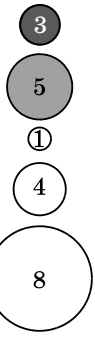
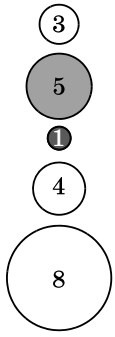
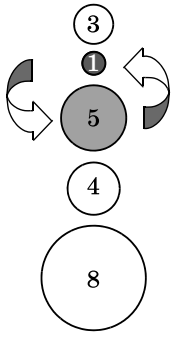
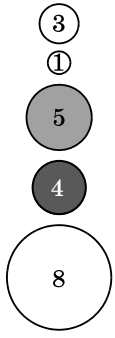
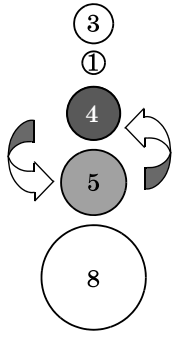
№ цикла	№ шага	№№ выбранных элементов (i и $i+1$)	До обмена элементов	После обмена элементов
1	4	4 и 5		

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.

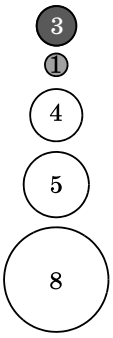
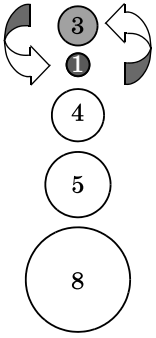
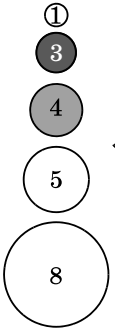
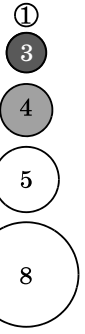
Остаётся повторить цикл ещё раз!

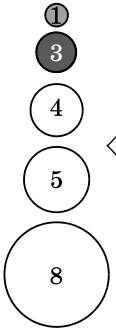
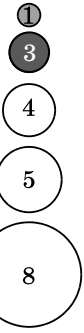
Зная, что последнее число уже стоит на своём месте и его можно исключить из сравнений, меняется значение цикловой переменной от 1 до 3 (т.е. для произвольного массива — от 1 до $(n - 2)$).

№ 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 бóльшие, словно камешки, опускаются вниз («тонут»). Именно поэтому данный метод сортировки и получил своё название.

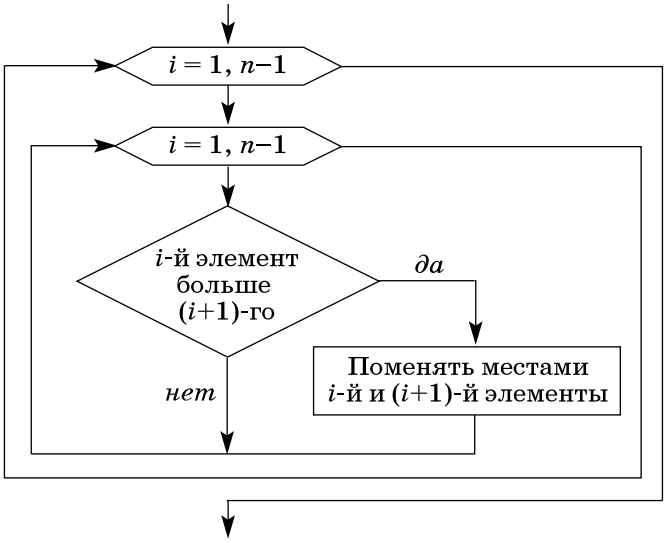
Обобщив только что разобранные действия на случай произвольного массива из n элементов получается, что:

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 всё 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

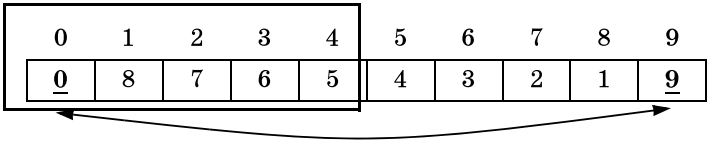
4. Изменение массива выполняется стоящей в теле второго цикла парой операторов:

```
k := A[i];
A[i] := A[9-i];
A[9-i] := k;
```

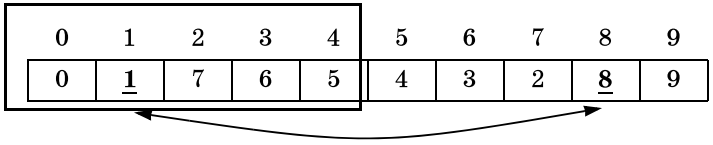
Эта алгоритмическая конструкция — обмен местами двух элементов (i-го и ($9-i$)-го) через буферную переменную k.

Эта операция выполняется последовательно для всех значений i:

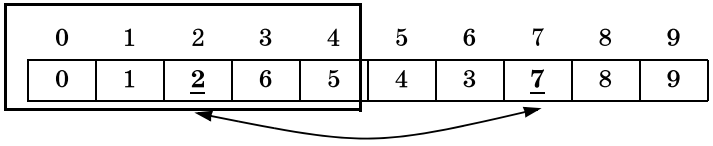
- $i = 0$:



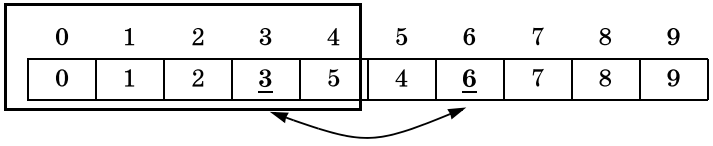
- $i = 1$:



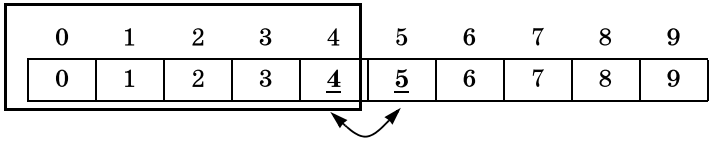
- $i = 2$:



- $i = 3$:



- $i = 4$:



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.

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 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

```
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
```

Паскаль

```
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

```
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];
}
```

Алгоритмический язык

```
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:

- $i = 0$:

0	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	**0**

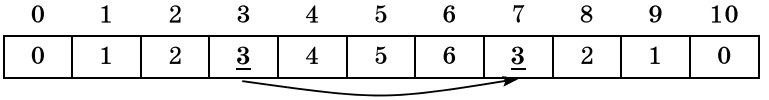
- $i = 1$:

0	1	2	3	4	5	6	7	8	9	10
0	**1**	2	3	4	5	6	7	8	**1**	0

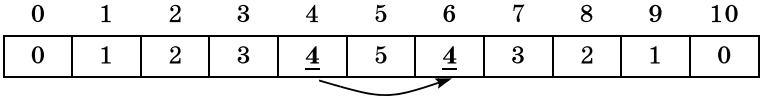
- $i = 2$:

0	1	2	3	4	5	6	7	8	9	10
0	1	**2**	3	4	5	6	7	**2**	1	0

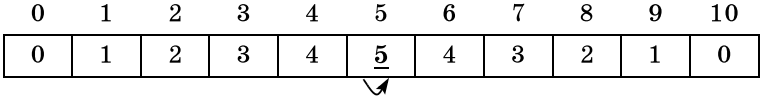
- $i = 3$:



- $i = 4$:

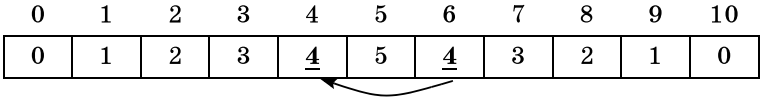


- $i = 5$:



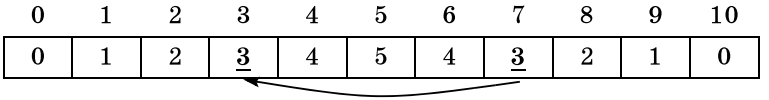
(При обмене элемента самого с собой никаких изменений не происходит.)

- $i = 6$:

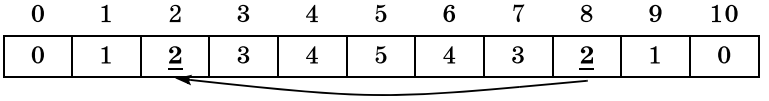


При дальнейших операциях массив не изменяется.

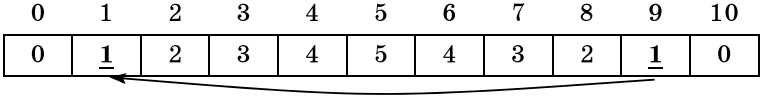
- $i = 7$:



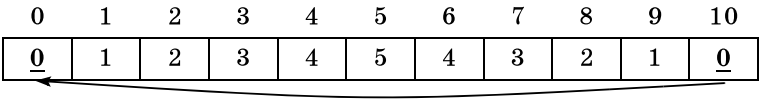
- $i = 8$:



- $i = 9$:



- $i = 10$:



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.

Задача 3. Имеется одномерный целочисленный массив M с индексами от 0 до 9. Значения его элементов изначально равны 16, 17, 13, 18, 15, 11, 12, 10, 19, 14 (M[0] = 16, M[1] = 17 и т.д.).

Требуется определить значение переменной x после выполнения фрагмента программы:

```
x := 0;
for i := 1 to 9 do
 begin
  if M[i] < M[0] then
   begin
    x := x + 1;
    t := M[i];
    M[i] := M[0];
    M[0] := t;
   end;
 end;
```

Решение

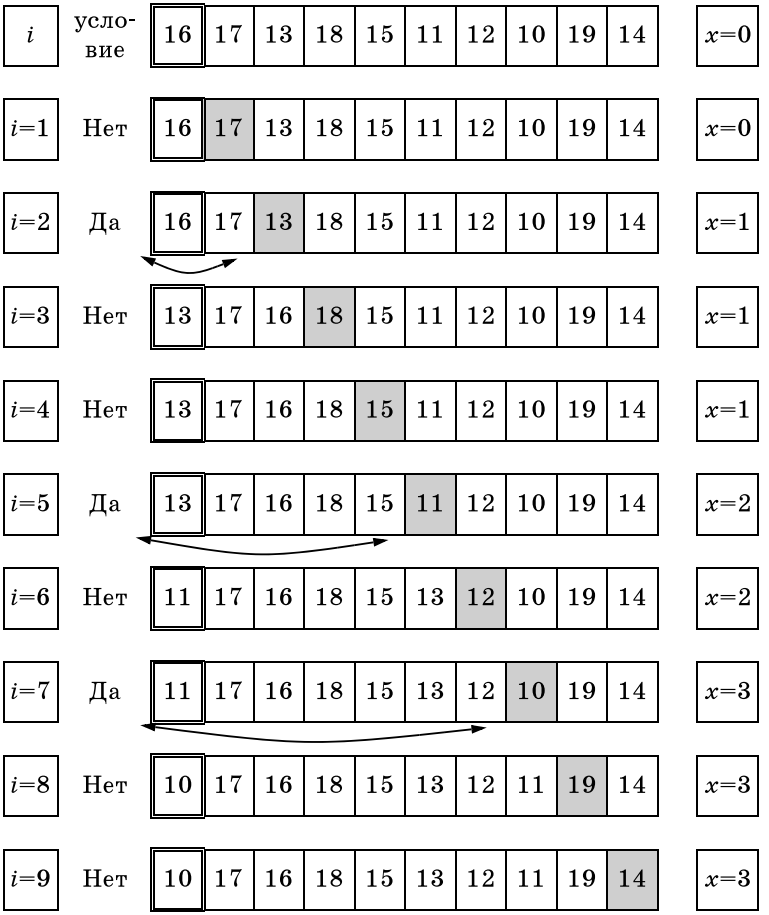
1. Анализируем алгоритм.

В цикле выполняется перебор элементов массива с 1-го по 9-й, при этом нулевой элемент циклом не затрагивается.

На каждом шаге цикла если очередной (i-й) элемент меньше нулевого, то эти элементы меняются местами.

Переменная x — это счётчик, который подсчитывает количество таких обменов.

2. Выполняем «прогонку» программы:



Ответ: 3.

Задача 4. В программе описан одномерный целочисленный массив. Ниже представлен фрагмент программы, обрабатывающей этот массив:

```
s := 0;
n := 8;
for i := 0 to n - 4 do begin
  s := s + A[i] - A[i+3]
end;
```

До выполнения этого фрагмента в массиве находились четырёхзначные нечётные натуральные числа. Какое наименьшее значение может иметь переменная s после выполнения данной программы?

Решение

1) Расписываем вычисление суммы в виде одной строки, помня, что значение i меняется в цикле от 0 до 6 (т.е. до 10 − 4):

$$s = 0 + A[0] - A[3] + A[1] - A[4] + A[2] - A[5] + \\ + A[3] - A[6] + A[4] - A[7] + A[5] - A[8] + A[6] - A[9].$$

2) Теперь сокращаем одинаковые переменные (элементы массива), записанные со знаками «плюс» и «минус»:

$$s = A[0] - \cancel{A[3]} + A[1] - \cancel{A[4]} + A[2] - \cancel{A[5]} + \cancel{A[3]} - \\ - \cancel{A[6]} + \cancel{A[4]} - A[7] + \cancel{A[5]} - A[8] + \cancel{A[6]} - A[9].$$

Получаем: $s = A[0] + A[1] + A[2] - A[7] - A[8] - A[9]$.

3) Получаемая сумма (разность) должна, по условию, иметь наименьшее значение. А сами элементы массива — это четырёхзначные нечётные натуральные числа, т.е. числа в диапазоне от 1001 до 9999. Следовательно, те элементы массива, которые записаны в выражении со знаком «плюс», должны быть наименьшими из возможных, а элементы массива со знаком «минус» должны быть наибольшими из возможных. Значит, нужно выбрать их следующими: $A[0] = A[1] = A[2] = 1001$, а $A[7] = A[8] = A[9] = 9999$.

4) Вычисляем значение s при этих значениях элементов массива:

$$s = 1001 + 1001 + 1001 - 9999 - 9999 - 9999 = \\ = 3 \cdot (1001 - 9999) = -26994.$$

Ответ: −26994.

Операции с массивами: обработка данных

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.

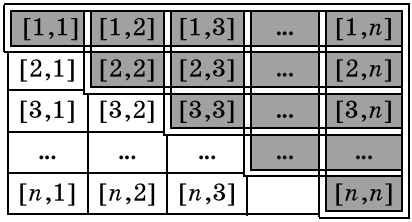
[1,1]	[1,2]	[1,3]	...	[1,n]
[2,1]	[2,2]	[2,3]	...	[2,n]
[3,1]	[3,2]	[3,3]	...	[3,n]
...	...	...	...	...
[n,1]	[n,2]	[n,3]		[n,n]

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.

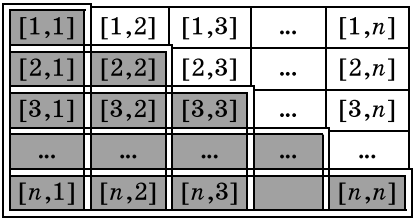
[1,1]	[1,2]	[1,3]	...	[1,n]
[2,1]	[2,2]	[2,3]	...	[2,n]
[3,1]	[3,2]	[3,3]	...	[3,n]
...	...	...	...	...
[n,1]	[n,2]	[n,3]		[n,n]

Верхняя матрица — это совокупность элементов, расположенных на главной диагонали и выше неё. Эти элементы имеют значения индексов: [i,j], где i меняется от 1 до n (размер матрицы), а j — меняется во вложен-

ном 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 неё. 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.



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 указываются 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 заверше-

нии 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.)

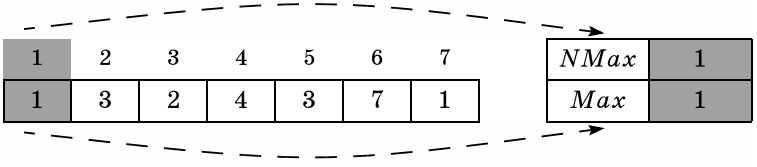
В цикле (для многомерного массива — во вложенных циклах) производится перебор элементов массива (полный или частичный — для фрагмента массива).

- **При поиске максимума/минимума** за предполагаемый максимум/минимум берётся первый элемент массива либо константа, заведомо меньшая/бóльшая 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.

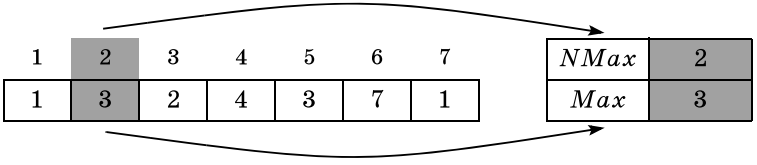
Поиск максимума	Поиск минимума

`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;`

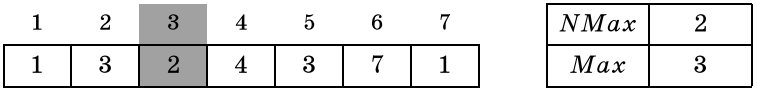
Пример: поиск максимума в массиве (1,3,2,4,3,7,1).
1) первоначально:



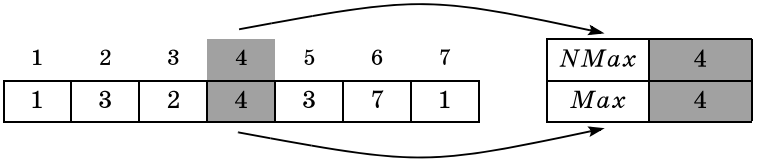
2) $i = 2$: $Mas[2] = 3$ больше, чем Max:



3) $i = 3$: $Mas[3] = 2$ меньше, чем Max:



4) $i = 4$: $Mas[4] = 4$ больше, чем Max:



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

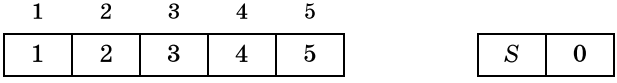
При вычислении суммы/произведения вначале переменной, выделенной для накопления суммы/произведения, присваивается инициализационное значение (нуль — для суммы, единица — для произведения). Затем в цикле (либо во вложенных циклах) выполняется перебор элементов массива и текущее значение суммы/произведения складывается/умножается на текущий элемент массива.

Вычисление суммы	Вычисление произведения

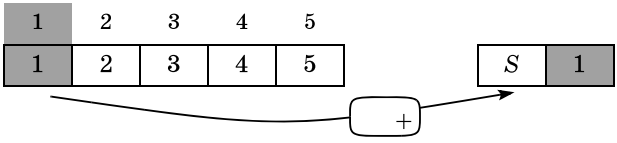
`S := 0;`
`for i := 1 to N do`
`  S := S + Mas[i];`
	`P := 1;`
	`for i := 1 to N do`
	`  P := P * Mas[i];`

Пример: вычисление суммы элементов массива (1,2,3,4,5).

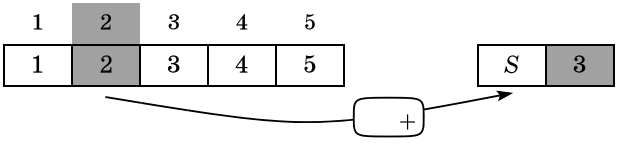
1) первоначально:



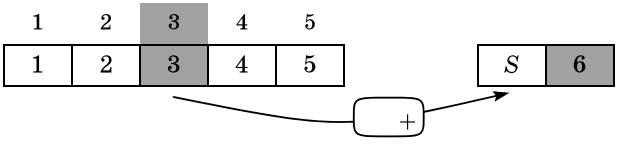
2) $i = 1$:



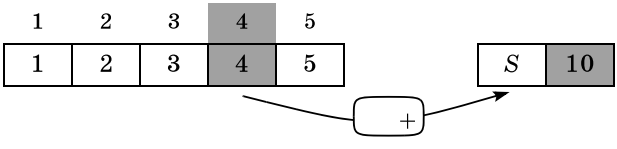
3) $i = 2$:



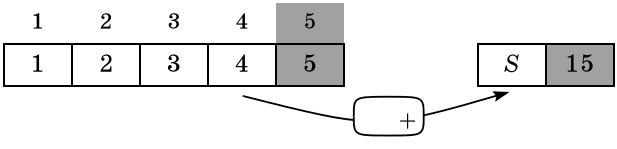
4) $i = 3$:



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 *бóльшую*, чем возможное самое большое значение элементов массива, удовлетворяющих заданному условию, а в качестве предполагае-

мого *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 *неудовлетворяющая* 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

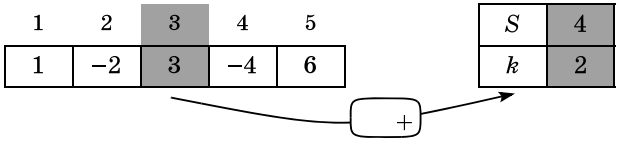
S	1
k	1

3) $i = 2$: $Mas[2] = -2$ — отрицательное число:

1	2	3	4	5
1	−2	3	−4	6

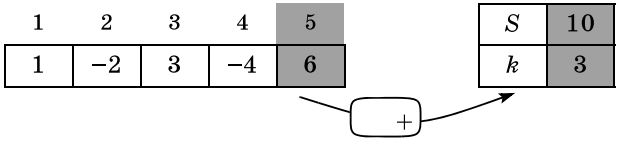
S	1
k	1

4) $i = 3$: $Mas[3] = 3$ — положительное число:



5) $i = 4$: $Mas[4] = -4$ — отрицательное число:

6) $i = 5$: $Mas[5] = 6$ — положительное число:



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

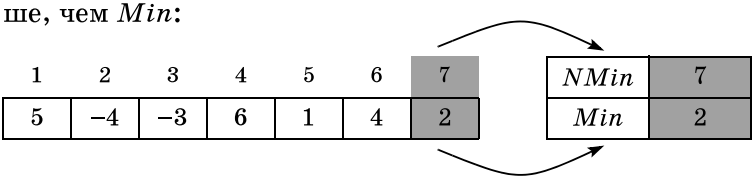
NMin	4
Min	6

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

NMin	**6**
Min	**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

NMin	7
Min	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:

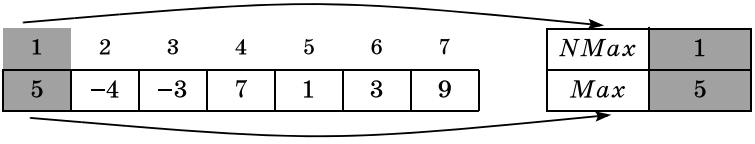
```
Max := 0; // поскольку требуется поиск
          // максимума среди положительных
          // элементов, нуль является
          // значением,заведомо меньшим любого
          // значения элемента массива,
          // удовлетворяющего условию
          // положительности
for i := 1 to N do
  if (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:

$NMax$	7
Max	9

Пример 4: минимум среди положительных элементов массива (5, –6, –3, 9, 1, 3, 6), кратных трём, 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

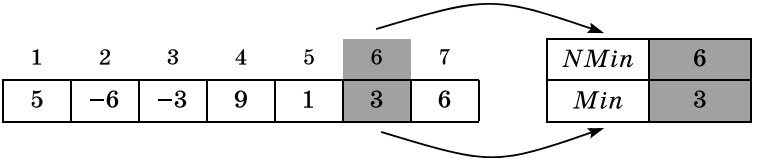
NMin	4
Min	9

6) $i = 5$: $Mas[5] = 1$ — положительное, но не кратное 3:

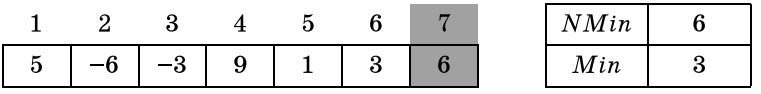
1	2	3	4	5	6	7
5	−6	−3	9	1	3	6

NMin	4
Min	9

7) $i = 6$: $Mas[6] = 3$ — положительное, кратное 3, меньше, чем Min:



8) $i = 7$: $Mas[7] = 6$ — положительное, кратное 3, но больше, чем Min:



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.
```

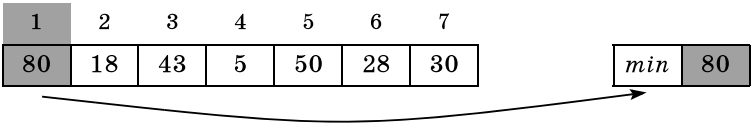
Пример для массива (80, 18, 43, 5, 50, 28, 30).

1) первоначально:

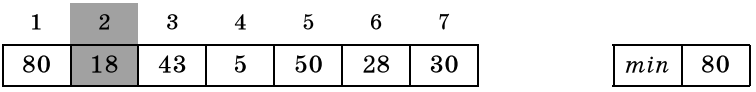
1	2	3	4	5	6	7
80	18	43	5	50	28	30

min	101

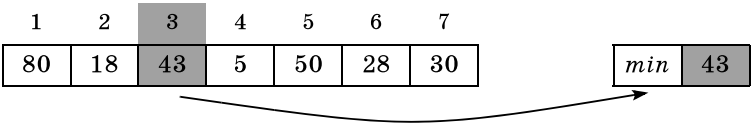
2) $i = 1$: $Mas[1] = 80$ — больше 20 и меньше, чем min:



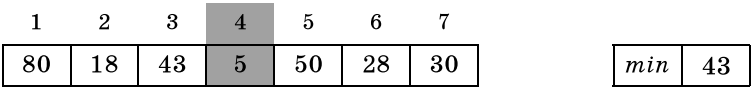
3) $i = 2$: $Mas[2] = 18$ — меньше 20:



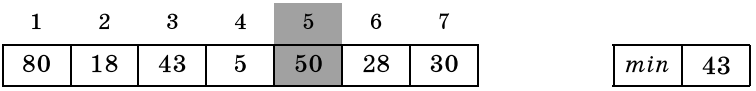
4) $i = 3$: $Mas[3] = 43$ — больше 20 и меньше, чем min:



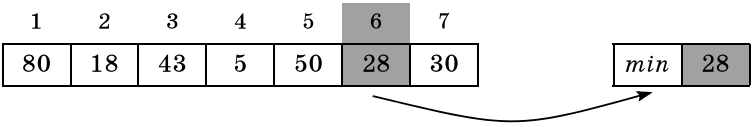
5) $i = 4$: $Mas[4] = 5$ — меньше 20:



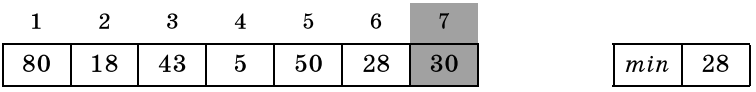
6) $i = 5$: $Mas[5] = 50$ — больше 20, но больше, чем min:



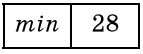
7) $i = 6$: $Mas[6] = 28$ — больше 20 и меньше, чем min:



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 нечётное значение. Гарантируется, что в исходном массиве хотя бы один элемент имеет нечётное значение.

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, x, y: integer;`
`s: real;`
`begin`
`for i := 1 to N do readln(a[i]);`
…
`end.`
	`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	**Естественный язык**

`#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]);`
…
`}`
	`Объявляем массив A из 30 элементов.`
	`Объявляем целочисленные переменные I, X, Y.`
	`Объявляем вещественную переменную S.`
	`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 переменные, аналогичные переменным, используемым в алгоритме, записанном на естественном языке, с учётом 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 одном из языков программирования алгоритм, позволяющий найти и вывести порядковый номер минимального значения среди нечётных положительных элементов массива.

Гарантируется, что в исходном массиве есть хотя бы один элемент, значение которого нечётно 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.
```

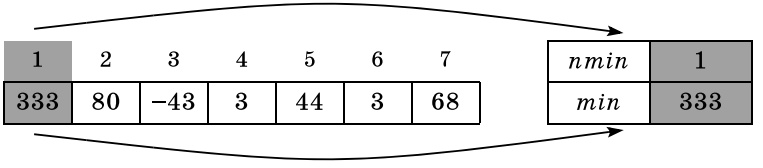
Пример для массива (333, 80, −43, 3, 44, 3, 68).

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

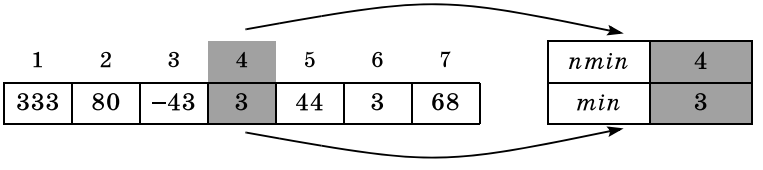
nmin	1
min	333

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

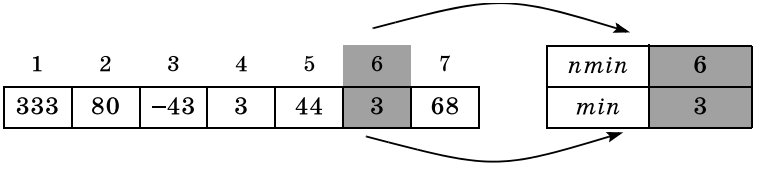
nmin	4
min	3

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

Задача 4. Дан массив, содержащий неотрицательные целые числа. Требуется определить и вывести на экран:

— максимальный чётный элемент, если количество чётных элементов не меньше, чем количество нечётных;

— максимальный нечётный элемент, если количество нечётных элементов больше, чем чётных.

Например, для массива (1, 2, 3, 4, 5) в качестве ответа нужно вывести число 5, так как нечётных чисел больше, чем чётных.

Напишите на изучаемом вами языке программирования программу для решения этой задачи. Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но можно не использовать часть описанных переменных. В качестве ответа нужно записать фрагмент программы, который должен быть на месте многоточия.

Бейсик	Python

CONST N = 2000
DIM A(N) AS INTEGER
DIM I,J,K,M AS INTEGER
FOR I = 1 TO N
  INPUT A(I)
NEXT I
…
END
	// допускается использовать
	// целочисленные переменные
	// j, k, m
	a = []
	N = 2000 // менять
	// значение N нельзя
	for i in range(0, n):
	  a.append(int(input()))
	…

Алгоритмический язык

```
алг
нач
  цел N = 2000
| Изменять значение
| этой переменной
| нельзя
  целтаб a[1:N]
  цел i, j, k, m
  нц для i от 1 до N
    ввод a[i]
  кц
  …
кон
```

Паскаль

```
const N = 2000;
var a: array [1..N] of
                integer;
i, j, k, m: integer;
begin
  for i:=1 to N do
    readln(a[i]);
  …
end.
```

Си

```
#include <stdio.h>
#define N 2000
int main(){
 int a[N];
 int i, j, k, m;
for (i = 0; i < N; i++)
 scanf("%d", &a[i]);
  …
  return 0;
}
```

Основная сложность при решении данной задачи следующая. Чтобы за один проход цикла определять количества чётных и нечётных элементов и максимальные значения чётных и нечётных элементов, требуется пять переменных (две — для хранения количеств, две — для хранения предполагаемых максимумов и счётчик цикла). А разрешается использовать только четыре объявленные переменные.

Чтобы обойти это ограничение, можно или реализовать несколько проходов по массиву (поскольку требование оптимальности по времени работы программы в данном типе задач не выдвигается), или применить небольшую хитрость — подсчитывать только количество, например, чётных элементов, так как количество нечётных легко определить как разность длины массива и вычисленного количества чётных.

Решение (*способ 1*)

```
program z;
const N = 2000;
var a: array [1..N] of integer;
     i, j, k, m: integer;
begin
  for i := 1 to N do
    readln(a[i]);

  j := 0; k := 0;
  for i := 1 to N do
    if (a[i] mod 2 = 0) then j := j + 1
// подсчет кол-ва четных
                          else k := k + 1;
// подсчет кол-ва нечетных

  m:=0; // предполагаемый максимум
  if j >= k then begin
    for i := 1 to N do
      if (a[i] mod 2 = 0) and (a[i] > m)
                    then m := a[i]; end
  else begin
    for i := 1 to N do
      if (a[i] mod 2 = 1) and (a[i] > m)
                   then m := a[i]; end;
  writeln('Max= ',m);
end.
```

Решение (*способ 2*)

```
program z;
const N = 2000;
var a: array [1..N] of integer;
     i, j, k, m: integer;
begin
  for i := 1 to N do
    readln(a[i]);
```

```
   j := 0; k := 0; m := 0;
   for i:=1 to N do
      if (a[i] mod 2 = 0) then begin
       j := j + 1;  // подсчет кол-ва четных
         if a[i] > k then k := a[i];
         // максимум среди четных
         end;
      else begin
         if a[i] > m then m := a[i];
         // максимум среди нечетных
         end;
   if j >= N - j then writeln('Max= ', k)
   // максимум среди четных
               else writeln('Max= ', m);
               // максимум среди нечетных

end.
```

Задача 5. Дан массив, содержащий 5000 натуральных чисел, не превышающих 10000. Требуется найти и вывести количество элементов этого массива, тринадцатеричная запись которых содержит ровно четыре знака и первая цифра больше третьей.

Исходные данные объявлены так, как показано ниже. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать только часть из описанных переменных.

```
Const
N=5000;
var a: array [1..N] of integer;
        i, s, k: integer;
```

```
begin
   for i:=1 to N do readln(a[i]);
   . . .
end.
```

Решение

На первый взгляд решение должно быть очень сложным — сначала перевести исходное *десятичное* число, хранящееся в массиве, в тринадцатеричную (!) систему счисления, потом проверить, сколько в этой записи получится цифр (операция вычисления длины символьной строки), потом выделить требуемые цифры и сравнить их как числа. Но на самом деле всё гораздо проще, эта задача решается полностью аналогично таким, где условие сформулировано для десятичной системы счисления.

1. Как проверить, что исходное десятичное число в *тринадцатеричной* системе счисления имеет ровно 4 знака?

Надо сравнить число с наименьшим и наибольшим возможными тринадцатеричными значениями, имеющими ровно 4 цифры. Очевидно, это числа 1000_{13} и $CCCC_{13}$.

(Напомним: в системах счисления с основанием больше 10 используются «цифры»: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15 и т.д., — а в тринадцатеричной системе счисления значения цифр ограничены 12-ю.)

Теперь вспомним, что исходное число у нас — десятичное. Значит, найденные «граничные» тринадцатеричные значения надо пересчитать в десятичные:

$1000_{13} = 13^3 = 2197_{10}$,

$CCCC_{13} = 10000_{13} - 1 = 13^4 - 1 = 28561 - 1 = 28560$.

Теперь записываем соответствующую часть условия, проверяющего, что в числе ровно 4 знака:

```
(A[i] >= 2197) and (A[i] <= 28560)
```

2. Как выделить из исходного *десятичного* числа первую и третью цифры *тринадцатеричной* записи?

Это можно тоже сделать по аналогии с «десятичной» задачей — используя операцию вычисления остатка от деления и операцию целочисленного деления исходного числа — только уже на основание заданной системы счисления в соответствующей степени.

2.1) Чтобы выделить первую («старшую») цифру числа, используем целочисленное деление на 13 в степени 3 (т.е. целочисленное деление на число 1000_{13}); это значение мы уже вычисляли:

```
A[i] div 2197
```

2.2) Чтобы выделить третью цифру (во второй позиции справа), нам нужно сначала получить часть числа, где эта цифра будет «старшей», т.е. вычислить остаток от деления исходного числа на 100_{13}, а потом выделить эту оставшуюся «старшей» цифру операцией целочисленного деления на 10_{13}. Только эти тринадцатеричные значения надо сначала пересчитать в десятичные:

$100_{13} = 13^2 = 169_{10}$,

$10_{13} = 13_{10}$.

Тогда операция выделения из исходного числа третьей цифры имеет вид:

```
(A[i] mod 169) div 13
```

3. Соответствующая часть условия, проверяющая, что первая цифра в тринадцатеричной записи больше третьей, будет такой:

```
(A[i] div 2197) > ((A[i] mod 169) div 13)
```


Проще всего сначала записать «типовые» операции с десятичными числами, потом «вспомнить», что каждое десятичное значение — это на самом деле число в соответствующей системе счисления, а затем пересчитать получившиеся «недесятичные» константы в десятичные числа. Например, для этой задачи последовательность действий будет такой:

Первая цифра 4-значного числа:		Третья цифра 4-значного числа:
`A[i] div 1000` ←	было бы для десятичного значения:	→ `(A[i] mod 100) div 10`
`A[i] div 1000`$_{13}$ ←	для 13-ричной системы счисления	→ `(A[i] mod 100`$_{13}$`) div 10`$_{13}$
`A[i] div 2197` ←	с пересчётом в десятичные значения	→ `(A[i] mod 169) div 13`

4. Остаётся записать весь оператор `if`, проверяющий заданные условия, а в его ветви `then` записать вычисление количества.

```
k := 0;          // обнуляем переменную-счетчик
for i:=1 to N do
    if (A[i] >= 2197) and (A[i] <= 28560) and
    (A[i] div 2197) > ((A[i] mod 169) div 13)
    then k := k + 1;
writeln(k);  // вывод результата
```

Для облегчения решения подобных задач полезно будет выучить следующую таблицу для четырёхзначных чисел и систем счисления с основаниями от 2 до 16 (при её изучении рекомендуется начать с наиболее понятного и наглядного случая — десятичной системы счисления):

Основание системы счисления	Нижняя граница четырёх-значности	Верхняя граница четырёх-значности	Выделение цифр (слева направо)			
			1-я цифра	2-я цифра	3-я цифра	4-я цифра
2	8	15	A[i] div 8	(A[i] mod 8) div 4	(A[i] mod 4) div 2	A[i] mod 2
3	27	80	A[i] div 27	(A[i] mod 27) div 9	(A[i] mod 9) div 3	A[i] mod 3
4	64	255	A[i] div 64	(A[i] mod 64) div 16	(A[i] mod 16) div 4	A[i] mod 4
5	125	624	A[i] div 125	(A[i] mod 125) div 25	(A[i] mod 25) div 5	A[i] mod 5
6	216	1295	A[i] div 216	(A[i] mod 216) div 36	(A[i] mod 36) div 6	A[i] mod 6
7	343	2400	A[i] div 343	(A[i] mod 343) div 49	(A[i] mod 49) div 7	A[i] mod 7
8	512	4095	A[i] div 512	(A[i] mod 512) div 64	(A[i] mod 64) div 8	A[i] mod 8

9	729	6560	A[i] div 729	(A[i] mod 729) div 81	(A[i] mod 81) div 9	A[i] mod 9
10	1000	10000 – 1 = = 9999	A[i] div 1000	(A[i] mod 1000) div 100	(A[i] mod 100) div 10	A[i] mod 10
11	1331	14640	A[i] div 1331	(A[i] mod 1331) div 121	(A[i] mod 121) div 11	A[i] mod 11
12	1728	20735	A[i] div 1728	(A[i] mod 1728) div 144	(A[i] mod 144) div 12	A[i] mod 12
13	2197	28560	A[i] div 2197	(A[i] mod 2197) div 169	(A[i] mod 169) div 13	A[i] mod 13
14	2744	38415	A[i] div 2744	(A[i] mod 2744) div 196	(A[i] mod 196) div 14	A[i] mod 14
15	3375	50624	A[i] div 3375	(A[i] mod 3375) div 225	(A[i] mod 225) div 15	A[i] mod 15
16	4096	65535	A[i] div 4096	(A[i] mod 4096) div 256	(A[i] mod 256) div 16	A[i] mod 16

Задача 6. Дан массив, содержащий положительные целые числа, не превышающие по значению 32768. Программа должна определить количество элементов этого массива, десятичная и восьмеричная запись которых содержат одинаковое количество цифр.

Исходные данные объявлены, как показано в тексте ниже. Запрещается использовать не описанные в нём переменные, но можно использовать не все описанные переменные.

```
const N = 5000;
var a: array [1..N] of integer;
     b, i, k, l, m: integer;
begin
   for i := 1 to N do
     readln(a[i]);
   …
   end.
```

Решение

В предыдущей задаче уже требовалось сформулировать в программе условие, определяющее, что в анализируемом числе (элементе массива) содержится заданное количество цифр, например, три. Это условие было записано в операторе `if` в виде сравнений с граничными значениями — наименьшим и наибольшим числами с нужным количеством цифр в соответствующей системе счисления. Следовательно, для выявления числа с некоторым заданным количеством цифр в конкретной системе счисления требуется записать два граничных условия. Нам же требуется определять и сравнивать **все** возможные количества цифр (от одной до пяти — в соответствии с указанным максимально возможным значением элементов массива) и в двух системах счисления — десятичной и восьмеричной. Тогда потребуется в `if` записывать по два граничных условия для каждого количества цифр и для каждой системы счисления, согласно схеме:

Если ((одна десятичная цифра) И (одна восьмеричная цифра)) ИЛИ ((две десятичные цифры) И (две восьмеричные цифры)) ИЛИ ((три десятичные цифры) И (три восьмеричные цифры)) ИЛИ ((четыре десятичные цифры) И (четыре восьмеричные цифры)) ИЛИ ((пять десятичных цифр) И (пять восьмеричных цифр))

Итого — 20 операций сравнения! Получается слишком громоздко!

В других задачах мы уже встречали алгоритм, позволяющий разбивать число на цифры в соответствующей системе счисления и подсчитывать количество таких цифр! Вот им-то и воспользуемся для решения данной задачи.

Приведём сразу требуемый листинг с соответствующими построчными комментариями.

`k:= 0;`	Счётчик искомого количества чисел, в которых одинаковы количества цифр в десятичной и восьмеричной системе счисления.
`for i:=1 to N do begin`	Цикл перебора всех элементов массива.
`  b:=a[i];`	Нужно сохранить копию значения текущего элемента массива, так как наш алгоритм подсчёта количества цифр «разрушает» число.

Определяем количество десятичных цифр

`  m:=0;`	Счётчик количества десятичных цифр.
`  while b > 0 do begin`	Цикл подсчёта количества цифр в числе.
`    m:=m+1;`	Увеличиваем количество цифр на 1.
`    b:=b div 10`	Отбрасываем последнюю **десятичную** цифру числа.

`end;`	Конец цикла подсчёта количества цифр.
`b:=a[i];`	Сохраняем копию значения текущего элемента массива.

Определяем количество восьмеричных цифр

`l:=0;`	Счётчик количества восьмеричных цифр.
`while b>0 do begin`	Цикл подсчёта количества цифр в числе.
`  l:=l+1;`	Увеличиваем количество цифр на 1.
`  b:=b div 8`	Отбрасываем последнюю **восьмеричную** цифру числа.
`end;`	Конец цикла подсчёта количества цифр.

Зная количества десятичных и восьмеричных цифр, сравниваем их

`if l=m then k:=k+1`	Если количества десятичных и восьмеричных цифр равны, увеличиваем на 1 счётчик количества элементов массива.
`end;`	Конец цикла перебора элементов массива.
`writeln(k)`	Вывод результата.

 Если в используемом языке программирования есть операции / функции / методы для работы с символьными и строковыми данными, в частности, для преобразования числа в его символьную запись в требуемой системе счисления и для подсчёта длины текстовой строки, то можно воспользоваться ими для решения данной задачи. Но, например, на языке Паскаль такие функции предусмотрены только для работы с десятичной системой счисления, а значит, реализовать решение таким способом не удастся.

Задача 7. Дан массив положительных целых чисел, не превышающих 1000. Нужно уменьшить все элементы, которые больше 100, на одно и то же значение, при этом минимальный из них должен стать равным 100.

Решение

Задача достаточно проста в программной реализации, но требует очень внимательного прочтения и понимания условия.

1. Минимальный элемент среди элементов больше 100, должен стать равным 100. Значит, сначала нужно найти значение такого элемента. Для этого используем следующий программный фрагмент:

```
m:=1001;
for i:=1 to N do
  if (a[i]>100) and (a[i]<m) then m:=a[i];
```

2. Далее нужно определить значение разницы, на которую надо уменьшить найденное минимальное значение m, чтобы оно стало равно 100:

```
k:=m-100;
```

3. Теперь надо снова просмотреть весь массив, и все числа, большие 100, уменьшить на эту только что вычисленную разницу. Ранее найденный минимальный элемент при этом автоматически станет равным 100.

```
for i:=1 to N do
  if a[i]>100 then a[i]:=a[i]-k;
```

4. Наконец, остаётся только организовать вывод полученного массива на экран.

Записать полный текст программы читателям предлагается самостоятельно.

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.

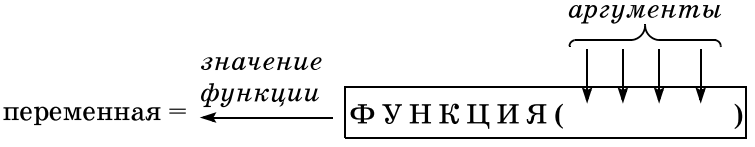
Иногда в виде подпрограмм выделяют фрагменты программного кода, выполняющие какие-либо типовые действия, — например, реализующие какие-то элементы интерфейса с пользователем. Это позволяет создавать отдельные *библиотеки подпрограмм*, облегчая этим работу программисту: ему уже не нужно отвлекаться на реализацию подобных «мелочей» и можно сосредоточиться непосредственно на сути решаемой вычислительной задачи, а библиотека гото-

вых 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 передаётся 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, заключённого 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 этих 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 воз-

вращает 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:

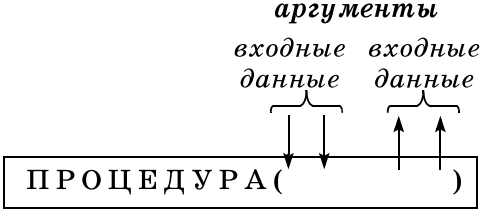
- записывается имя функции, а после него в скобках через запятую — список *фактических параметров*: константы, имена переменных, выражения, а также другие вызовы функций, значения которых нужно передать в функцию для выполнения вычислений. При этом количество и типы таких фактиче-ских параметров должны 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 подпро-

грамму 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 соответствующих 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 неё как значения или как ссылки. В чём 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.

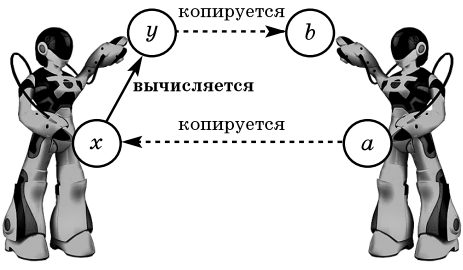
4. Результаты вычислений, которые процедура должна вернуть в основную программу, должны быть записаны в качестве значения соответствующего формального параметра.

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



Значение переменной *a* в основной программе **не изменяется**

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 в основной программе встречается 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		

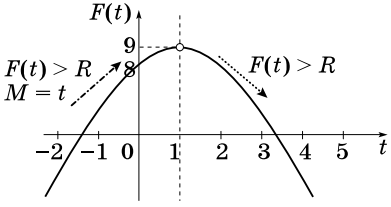
и т.д.

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 ⩽ *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 ⩽ *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**.

Задача 2. Определите, какое число будет напечатано в результате работы следующей программы (для вашего удобства программа представлена на четырёх языках):

Бейсик	**Паскаль**

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
	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 таблицы

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`

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:

$(x - 1) \cdot (x - 3) = 0 \Rightarrow x_1 = 1, x_2 = 3.$

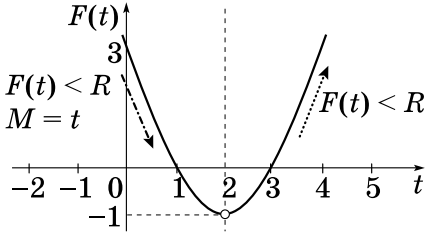
Чтобы определить направление ветвей параболы, нужно определить знаки значений заданной функции в трёх точках, одна из которых расположена внутри интервала между найденными корнями, а две другие — вне этого интервала:

- $x = 0 \Rightarrow \rightarrow = (0 - 1) \cdot (0 - 3) = 3$ (знак «+»);
- $x = 2 \Rightarrow \rightarrow = (2 - 1) \cdot (2 - 3) = -1$ (знак «−»);
- $x = 4 \Rightarrow \rightarrow = (4 - 1) \cdot (4 - 3) = 3$ (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 переза-

пись 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**.

В подобных задачах 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.

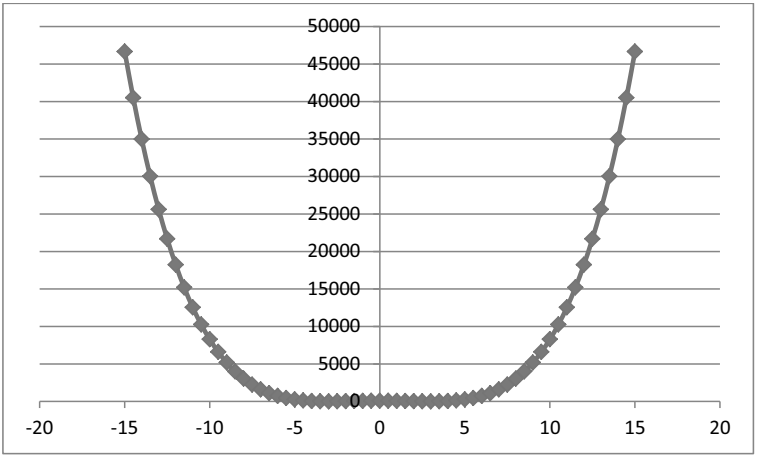
Задача 3. Определить, какое число будет напечатано в результате выполнения программы:

```
var a, b, t, M, R: integer;
Function F(x: integer): integer;
begin
  F := (x*x - 9)*(x*x - 9) + 3
end;
begin
  a := -15; b := 15;
  M := a; R := F(a);
  for t := a to b do
    if (F(t) < R) then begin
      M := t; R := F(t)
    end;
write(M*R)
end.
```

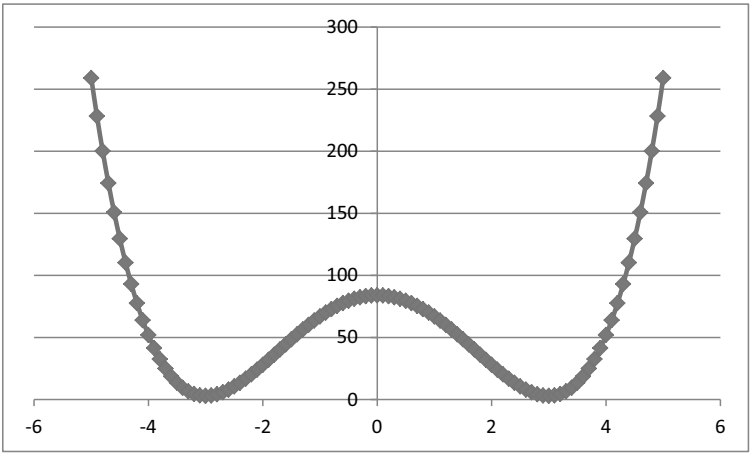
Решение

Отличие от предыдущих задач — в том, что функция представляет собой биквадратное выражение (x в четвёртой степени). Парабола для такой функции получается немного другая, чем для квадратного выражения.

Вот как соответствующий график выглядит при его построении в программе Excel в заданном диапазоне от −15 до 15 с шагом 0,5:



Кажется, что это всё та же парабола с единственным минимумом. Но если выполнить построение в большем масштабе — для диапазона от −5 до 5 и с шагом 0,1, то картина получается совершенно иная:



То есть для выражения с четвёртой степенью график имеет локальный максимум и два симметрично расположенных относительно него минимума. Соответственно это нужно учесть при анализе поведения предлагаемой программы.

1. Коэффициент при x^4 (если выполнить перемножение скобок) — положительный, значит, ветви параболы направлены вверх.

2. В выражении функции $(x \times x - 9) \times (x \times x - 9) + 3$ легко определить, какие точки графика соответствуют минимумам: скобки перемножаются одинаковые — значит, эти точки расположены симметрично относительно оси Y, а приравняв одну из скобок нулю: $(x \times x - 9) = 0$, получаем, что $x^2 = 9$, тогда значения x равны −3 и 3. Локальный максимум же (учитывая симметрию графика относительно оси Y) соответствует $x = 0$.

3. Прибавление константы 3 всего лишь «сдвигает» график на 3 единицы вверх. Это пока можно не учитывать.

4. В программе движение по графику производится слева направо. При этом переприсваивание значений переменных производится, если текущее значение функции меньше предыдущего, причём неравенство в условном операторе записано **строгое**. Следовательно, программа выполняет **поиск первого по счёту минимума функции** (при повторном обнаружении такого же минимального значения переприсваивание производиться не будет — неравенство строгое!).

5. Первый по счёту при движении по графику слева направо минимум соответствует значению x (или t, что для нас то же самое), равному -3. Подставив это значение в выражение функции, получаем, что $F(t)$ равно 3.

6. В переменной M запоминается значение t найденной точки, а в переменной R — соответствующее значение функции, следовательно, в нашем случае $M = -3$, а $R = 3$.

Теперь особенно внимательно смотрим на то, что именно должна выводить программа.

В данной задаче программа выводит произведение значений M и R, и оно равно -9.

Ответ: -9.

Задача 4. Определите, какое значение H нужно ввести, чтобы число, напечатанное в результате выполнения следующего алгоритма, было наименьшим.

```
var a, b, t, M, R, H : integer;
Function F(H, x: integer):integer;
  begin
    F := 11*(x - H)*(x - H) + 13;
  end;
begin
  readln(H);
  a := -10;
  b := 50;
  M := a;
  R := F(H,a);
```

```
    for t := a to b do begin
      if (F(H,t) > R) then
      begin
        M := t;
        R := F(H,t);
      end;
    end;
    write(R);
  end.
```

Решение

В предыдущих задачах требовалось для функции одной переменной F(x), представляющей собой квадратное уравнение, найти, какое число выводится в результате. А здесь — функция с двумя переменными F(H,x), которая представляет собой квадратное уравнение с параметром H, и искать надо значение этого параметра, такое, чтобы выводимое значение было наименьшим.

Прежде всего заметим, что работа в языке программирования (в данном случае в Паскале, но это верно и для других языков — по крайней мере для большинства из них) с функциями с одной и с двумя переменными полностью идентична: в обоих случаях в функцию передаются значения переменных, а результат возвращается через саму функцию F( ). Поэтому сосредоточимся на собственно решении.

1) Анализируем алгоритм.

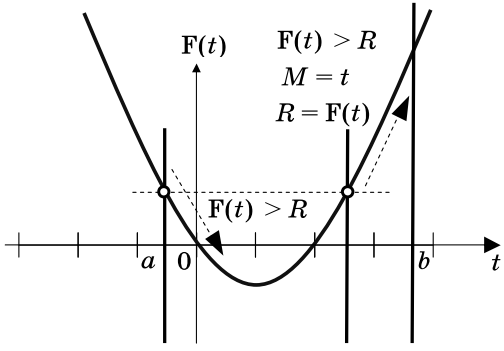
- В цикле идёт перебор значений t от заданных ранее $a = -10$ до $b = 50$. При этом в переменной M запоминается сначала начальное значение a, а потом (при срабатывании условного оператора) — текущее значение t, а в переменной R хранится предыдущее значение функции F(H, x).
- Если с увеличением значения t получаемое значение F( ) увеличивается (и в результате оказывается больше предыдущего значения F(t), запомненного в переменной R), текущее значение t за-

носится в переменную M, а значение функции в этой точке запоминается в R.

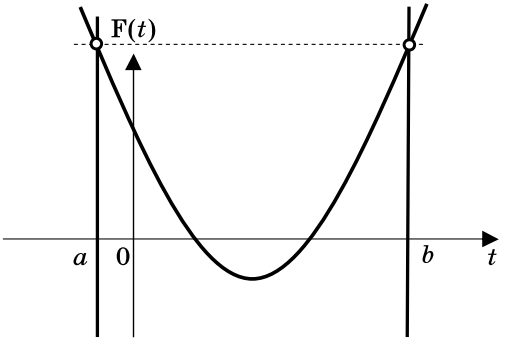
- До каких пор это будет продолжаться? Посмотрим на выражение, записанное в составе функции F( ). Выполним в нём перемножение скобок. Получим:

$$11 \cdot (x - \text{Н}) \cdot (x - \text{Н}) + 13 = 11x^2 - 22x\text{Н} + (11\text{Н}^2 + 13).$$

Это явно запись квадратного уравнения (если приравнять это выражение нулю), графиком которого является парабола. Поскольку перед x^2 стоит знак «плюс», ветви этой параболы направлены вверх. И мы в своих вычислениях «движемся» по левой ветви этой параболы сверху вниз. Значит, условный оператор в цикле не будет работать до тех пор, пока мы не пройдём вершину параболы (минимум на её графике) и не поднимемся по правой ветви настолько, что значение выражения $11t^2 - 22t\text{Н} + (11\text{Н}^2 + 13)$ (замена переменной x на t происходит при переходе из основной программы в функцию) в некоторой точке t превысит его значение в исходной точке a. А после этого, пока мы «движемся» по правой ветви параболы вверх, у нас значение R будет всё расти вплоть до окончания цикла по достижении точки b (см. примерный рисунок). Следовательно, конечное значение R будет равно значению выражения $\text{F}(\text{Н}, b) = 11b^2 - 22b\text{Н} + (11\text{Н}^2 + 13)$ либо, если расположить параболу так, что указанное значение меньше, чем исходное ($\text{F}(\text{Н}, a) = 11a^2 - 22a\text{Н} + (11\text{Н}^2 + 13)$), значение R будет равно этому исходному.



2) Как нужно расположить параболу, чтобы получить минимально возможное значение R? И каким оно будет? Очевидно, что если ветвь `then` начнёт работать, то это приведёт к увеличению значения R. Следовательно, наименьшее значение R (которое и выводится на экран в результате работы программы), будет достигаться, если ветвь `then` так ни разу и не сработает (и будет равным значению в исходной точке a). Для этого требуется, чтобы в точках $a = -10$ и $b = 50$ значения выражения, записанного в функции, были равными:



3) Остаётся вычислить значение H такое, чтобы получить нужное расположение параболы. Для этого надо составить уравнение в виде:

$$11a^2 - 22a\text{H} + (11\text{H}^2 + 13) = \\ = 11b^2 - 22b\text{H} + (11\text{H}^2 + 13)$$

или, подставляя известные нам значения $a = -10$ и $b = 50$:

$$11 \cdot (-10)^2 - 22 \cdot (-10)\text{H} + (11\text{H}^2 + 13) = \\ = 11 \cdot 50^2 - 22 \cdot 50\text{H} + (11\text{H}^2 + 13).$$

Очевидно, одинаковые слагаемые $(11\text{H}^2 + 13)$ сразу сокращаются. Также можно разделить обе его части на 11. Получаем уравнение первого порядка:

$$(-10)^2 - 2 \cdot (-10)\text{H} = 50^2 - 2 \cdot 50\text{H}.$$

Решаем его:

100 + 20H = 2500 – 100H;
100H + 20H = 2500 – 100;
120H = 2400; H = 20.
Ответ: 20.

Задача 5. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 if n < 6 then begin
   F(n + 1);
   F(n*2)
 end
end;
```

Чему равна сумма чисел, выведенных на экран при вызове **F(2)**?

Табличный метод решения. Суть его в том, что мы, начиная с исходного значения *n* (в данном случае — 2), составляем таблицу, в которой расписываем и то, что выводится на экран, и требуемые рекурсивные вызовы процедуры с соответствующими новыми значениями *n*, которые тоже помещаем в нашу таблицу... и так до тех пор, пока не дойдём до условия прекращения рекурсии. А после этого мы снова проходим по составленной таблице уже в обратном порядке, вычисляя возвращаемые при рекурсивных вызовах значения и подставляя эти значения в предыдущие строки таблицы.

Вспомним основную часть рассматриваемого рекурсивного алгоритма:

```
writeln(n);
 if n < 6 then begin
   F(n + 1);
   F(n*2)
 end
```

Распишем построение таблицы для этой задачи подробно.

Начинаем с заданного начального значения $n = 2$ и не забываем, что значение n выводится на экран в любом случае (так как команда `writeln(n)` располагается вне оператора `if`).

Получающиеся новые вызовы процедуры F будем выделять серым фоном.

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2

Теперь добавляем в таблицу строки для $n = 3$ и $n = 4$:

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2
3	F(4)	F(6)	3
4	F(5)	F(8)	4

В таблице появились новые вызовы процедуры F для $n = 5$, 6 и 8. Добавляем в таблицу и их. Но не забываем, что для значений 6 и 8 уже срабатывает условие завершения рекурсии и никаких вызовов F для них уже не будет. Но само значение n на экран по-прежнему выводится!

n	F(n+1)	F(n*2)	Вывод на экран
2	F(3)	F(4)	2
3	F(4)	F(6)	3
4	F(5)	F(8)	4
5	F(6)	F(10)	5
6	–	–	6
8	–	–	8

Наконец, добавляем последнюю строку для F(10):

n	F(*n*+1)	F(*n**2)	Вывод на экран
2	F(3)	F(4)	2
3	F(4)	F(6)	3
4	F(5)	F(8)	4
5	F(6)	F(10)	5
6	–	–	**6**
8	–	–	**8**
10	–	–	**10**

Видим, что никаких новых рекурсивных вызовов процедуры F больше нет.

Можно приступать ко второй части построения таблицы — к просмотру её строк с вызовами F построчно снизу вверх. При этом в каждой очередной строке сразу будем вычислять сумму выводимых на экран чисел, прибавляя к уже записанному значению то, что должно выводиться за счёт указанных в таблице рекурсивных вызовов.

Полученное очередное значение суммы является результатом данного вызова F и затем, в свою очередь, подставляется в предыдущую строку таблицы.

И так — пока мы не дойдём до самой первой строки, содержащей указанный в условии задачи изначальный вызов рекурсивной функции.

n	F(*n*+1)	F(*n**2)	Вывод на экран
5	**F(6)**	***F(10)***	5 + **6** + ***10*** = 21
6	–	–	**6**
8	–	–	8
10	–	–	***10***

Аналогично поступаем с предыдущей строкой для $n = 4$:

n	F(n+1)	F(n*2)	Вывод на экран
4	**F(5)**	***F(8)***	$4 + \mathbf{21} + \boldsymbol{8} = 33$
5	F(6)	F(10)	21
6	–	–	6
8	–	–	***8***
10	–	–	10

Теперь добавляем строку для предыдущего значения $n = 3$ (стрелки уже рисовать не будем):

n	F(n+1)	F(n*2)	Вывод на экран
3	**F(4)**	***F(6)***	$3 + \mathbf{33} + \boldsymbol{6} = 42$
4	F(5)	F(8)	**33**
5	F(6)	F(10)	21
6	–	–	***6***
8	–	–	8
10	–	–	10

И наконец, добавляем самую первую строку для $n = 2$:

n	F(n+1)	F(n*2)	Вывод на экран
2	**F(3)**	***F(4)***	$2 + \mathbf{42} + \boldsymbol{33} = \underline{\mathbf{77}}$
3	F(4)	F(6)	**42**
4	F(5)	F(8)	***33***
5	F(6)	F(10)	21
6	–	–	6
8	–	–	8
10	–	–	10

Полученное при последнем сложении число **77** — это и есть ответ.

Ответ: 77.

Задача 6. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 if n < 7 then begin
   writeln(n);
   F(n+1);
   F(n+2);
   F(n*2)
 end
end;
```

Чему равна сумма чисел, выведенных на экран при вызове F(2)?

Решение

Уже познакомившись с решением предыдущей задачи, будем записывать решение более коротко.

Таблицу начинаем составлять, начиная с исходного значения $n = 2$. Теперь в ней будет три графы для записи вложенных рекурсивных вызовов, а в графе «Вывод на экран» не забываем указывать два значения, если условие в `if` истинно.

n	F(n+1)	F(n+2)	F(n*2)	Вывод на экран
2	F(3)	F(4)	F(4)	2 + 2
3	F(4)	F(5)	F(6)	3 + 3
4	F(5)	F(6)	F(8)	4 + 4
5	F(6)	F(7)	F(10)	5 + 5
6	F(7)	F(8)	F(12)	6 + 6
7	–	–	–	7
8	–	–	–	8
9	–	–	–	9
10	–	–	–	10
12	–	–	–	12

А теперь идём по таблице в обратном порядке (снизу вверх):

n	F(n+1)	F(n+2)	F(n*2)	Вывод на экран
2	F(3)	F(4)	F(4)	2 + 2 + 232 + + 121 + 121 = **478**
3	F(4)	F(5)	F(6)	3 + 3 + 121 + + 66 + 39 = 232
4	F(5)	F(6)	F(8)	4 + 4 + 66 + + 39 + 8 = 121
5	F(6)	F(7)	F(10)	5 + 5 + 39 + + 7 + 10 = 66
6	F(7)	F(8)	F(12)	6 + 6 + 7 + 8 + + 12 = 39
7	–	–	–	7
8	–	–	–	8
9	–	–	–	9
10	–	–	–	10
12	–	–	–	12

Ответ: 478.

Задача 7. Имеется следующий рекурсивный алгоритм:

```
procedure F(n: integer);
begin
 writeln(n);
 n := n + 2;
 if n < 8 then begin
    F(n+2);
    F(n+3);
end
end;
```

Чему равна сумма чисел, выведенных на экран при вызове F(1)?

Решение

Здесь вне оператора `if` добавился оператор увеличения значения n, но нам составлять таблицу это не помешает — нужно лишь не забывать, что в условии оператора `if` и в последующих вложенных вызовах F участвует уже измененное значение n.

n	Новое n	F(n+2)	F(n+3)	Вывод на экран
1	3	F(5)	F(6)	1
5	7	F(9)	F(10)	5
6	8	–	–	6
9	11	–	–	9
10	12	–	–	10

В графу «Вывод на экран» попадают первоначальные значения n, так как вывод производится до изменения n. Что будет выводиться, если команда вывода будет стоять после операции `n := n + 2` или внутри оператора `if`, читатели, наверное уже определили сами.

Теперь после пути «туда», подобно небезызвестному хоббиту, возвращаемся «обратно»:

n	Новое n	F(n+2)	F(n+3)	Вывод на экран
1	3	F(5)	F(6)	1 + 24 + + 6 = **31**
5	7	F(9)	F(10)	5 + 9 + + 10 = 24
6	8	–	–	6
9	11	–	–	9
10	12	–	–	10

Ответ: 31.

Задача 8. Дан рекурсивный алгоритм:

```
procedure F(n: integer);
begin
writeln('*');
 if n > 0 then begin
    writeln('*');
    F(n-2);
    F(n div 2);
    F(n div 2);
 end
end;
```

Сколько символов «звёздочка» будет напечатано на экране при выполнении вызова F(7)?

Решение

Все различие здесь — в том, что в графе «Вывод на экран» числа будут обозначать не сами значения, выводимые на экран, а количества выводимых туда «звёздочек» (при первичном заполнении — отмечаем, что **по одной** в каждой команде вывода). И, конечно же, не забываем, что здесь имеются две команды вывода на экран. Кроме того, здесь имеются два одинаковых рекурсивных вызова `F(n div 2)`, и чтобы про них не забыть, в таблице мы запишем две соответствующие графы — пусть даже информация в них будет дублироваться.

n	F(*n*−2)	F(*n* div 2)	F(*n* div 2)	Вывод на экран
7	F(5)	F(3)	F(3)	1 + 1
5	F(3)	F(2)	F(2)	1 + 1
3	F(1)	F(1)	F(1)	1 + 1
2	F(0)	F(1)	F(1)	1 + 1
1	F(−1)	F(0)	F(0)	1 + 1
0	–	–	–	1
−1	–	–	–	1

А теперь идём обратно:

n	F(*n*−2)	F(*n* div 2)	F(*n* div 2)	Вывод на экран
7	F(5)	F(3)	F(3)	1 + 1 + 45 + + 17 + 17 = **81**
5	F(3)	F(2)	F(2)	1 + 1 + 17 + + 13 + 13 = = 45
3	F(1)	F(1)	F(1)	1 + 1 + 5 + + 5 + 5 = 17
2	F(0)	F(1)	F(1)	1 + 1 + 1 + + 5 + 5 = 13
1	F(−1)	F(0)	F(0)	1 + 1 + 1 + + 1 + 1 = 5
0	–	–	–	1
−1	–	–	–	1

Ответ: 81.

Задача 9. Имеется рекурсивный алгоритм F.

```
procedure F(n: integer);
begin
 if n > 0 then begin
  F(n - 3);
  F(n div 3);
  write(n)
 end
end;
```

Требуется записать подряд без пробелов и разделителей все числа, которые будут напечатаны на экране при выполнении вызова F(9). Числа должны быть записаны в том же порядке, в котором они выводятся на экран.

Решение

1. Составляем таблицу вызовов функций (эти вызовы функций для наглядности также выделены цветом в тексте программы в условии задачи):

Исходный вызов	Генерируемые им вызовы	
	F(n−3)	*F(n div 3)*
F(9)	F(6)	F(3)
F(6)	F(3)	F(2)
F(3)	F(0)	F(1)
F(2)	F(−1)	F(0)
F(1)	F(−2)	F(0)

2. Добавляем к этой таблице ещё одну графу справа — для выводимых данных. Заполняем эту графу *снизу вверх*, при этом сначала записываются значения, выводимые генерируемыми вызовами (по порядку *слева направо*), а затем к ним дописывается значение, выводимое исходной функцией. Также не забываем, что выводятся только значения функций с $n > 0$.

Исходный вызов	Генерируемые им вызовы		Выводимые значения	Комментарий
	F(n – 3)	***F(n div 3)***		
F(9)	**F(6)**	*F(3)*	**1326** *13* 9	Выводятся значения для **F(6)**, *F(3)* и для исходной функции
F(6)	**F(3)**	*F(2)*	**13** *2* 6	Выводятся значения для **F(3)**, *F(2)* и для исходной функции
F(3)	**F(0)**	*F(1)*	*1* 3	Выводятся значения для *F(1)* и для исходной функции
F(2)	**F(–1)**	**F(0)**	2	Выводится только значение для исходной функции
F(1)	**F(–2)**	**F(0)**	1	Выводится только значение для исходной функции

Получаемая в результате последовательность выводимых значений записана в первой ячейке правого столбца (для исходного вызова функции): **1326139.**

Ответ: 1326139.

Задача 10. В программе содержатся две рекурсивные функции **F** и **G**:

```
function F(n: integer): integer;
begin
  if n > 3 then
    F := F(n - 2) + G(n - 3)
```

```
    else
      F := n - 1;
end;
function G(n: integer): integer;
begin
    if n > 3 then
      G := G(n - 2) + F(n - 3);
    else
      G := n + 1;
    end;
```

Чему равно значение, вычисленное при выполнении вызова F(8)?

Решение

Отрабатываем программу для $n = 8$. При этом таблицу трассировки удобнее строить из двух частей (соответствующих вызовам F(n) и G(n)) и решать задачу в два прохода — сначала на прямом проходе мы только записываем вызовы функций, а затем на обратном проходе, в обратном порядке снизу вверх, записываем получаемые значения.

F(n)	Условие	F(n)	G(n)	Условие	G(n)
F(8)	8>3 – Да	F(8) = F(6)+G(5)	G(5)	5>3 – Да	G(5) = G(3)+F(2)
F(6)	6>3 – Да	F(6) = F(4)+G(3)	G(3)	3>2 – Да	G(3) = G(1)+F(0)
F(4)	4>3 – Да	F(4) = F(2)+G(1)	G(1)	1>2 – Нет	G(1) = 1+1 = **2**
F(2)	2>3 – Нет	F(2) = 2 – 1 = **1**			
F(0)	0>3 – Нет	F(0) = 0 – 1 = **–1**			

F(n)	G(n)
F(8) = 4+2 = **6**	G(5) = 1+1 = **2**
F(6) = 3+1 = **4**	G(3) = 2–1 = **1**
F(4) = 1+2 = **3**	G(1) = **2**
F(2) = **1**	
F(0) = **–1**	

Ответ: **6**.

Другой возможный вариант решения — построение дерева вызовов функций, однако этот способ решения более громоздкий.

Задача 11. Имеются алгоритмы рекурсивных процедур F и G. Определите сумму чисел, выведенных на экран при выполнении вызова F(93).

```
procedure F(n:integer); forward;
procedure G(n:integer); forward;

procedure F(n: integer);
 begin
  if n mod 5 > 3 then G(n div 5)
  else G(n div 2)
 end;

procedure G(n: integer);
 begin
  write(n, ' ');
  if n > 1 then F(n-1);
 end;
```

Решение

Этот новый тип задач даже проще, чем ранее рассмотренные. Для решения составим таблицу, начиная отсчёт с заданного значения n (промежуточные вычисления для определения выполняемых вызовов F(n) и G(n) выполним «в уме»):

F(n)	F(93)	F(45)	F(21)	F(9)	**Вызовы прекращены**
G(n)	G(46)	G(22)	G(10)	G(1)	**Итоговая сумма:**
вывод на экран	46	22	10	1	46+22+10+ +1 = 79

Ответ: 79.

Задача 12. Определите количество различных значений входной переменной k, при которых программа выдаёт тот же ответ, что и при входном значении $k = 64$. Значение $k = 64$ также включается в подсчёт.

```
var k, i : longint;
function f(n: longint) : longint;
  begin
    f := n * n
  end;
begin
  readln(k);
  i := 12;
  while (i>0) and (f(i)>=k) do
    i := i-1;
  writeln(i)
end.
```


В новых задачах такого типа выражение, записанное в подпрограмме-функции, может быть любым, а цикловая переменная может как уменьшаться, так и увеличиваться, но принцип решения таких задач остаётся тем же. В любом случае первое неравенство составляется для значения *i*, при котором цикл ещё выполняется (последний раз), и в нём записывается в точности такой знак неравенства, как в условии цикла. Второе неравенство составляется для значения *i*, при котором цикл уже не выполняется, и знак этого неравенства — противоположный первому (в том числе по строгости/нестрогости).

Решение (*первый способ*)

1) Анализируем алгоритм. Функция используется здесь только для выполнения вычислений, поэтому для упрощения анализа программы можно записать соответствующее выражение прямо взамен вызова функции:

```
readln(k);
i := 12;
while (i>0) and (i*i>=k) do
  i := i-1;
writeln(i)
```

2) Теперь выполним трассировку программы для заданного значения $k = 64$:

i	$i \cdot i$	Условие	k
12	144	$12 > 0;\ 144 \geqslant 64$	64
11	121	$11 > 0;\ 121 \geqslant 64$	
10	100	$10 > 0;\ 100 \geqslant 64$	
9	81	$9 > 0;\ 81 \geqslant 64$	
8	64	$8 > 0;\ 64 \geqslant 64$	
7	49	$7 > 0;\ 49 < 64$	


Обратите внимание: при $i = 8$ условие выполнения цикла всё ещё истинно, поэтому в теле цикла в последний раз выполняется уменьшение значения переменной i. Но сразу после этого условие цикла становится ложным, выполнение цикла прекращается, а на экран выводится уменьшенное на 1 значение i, равное 7.

Итак, для $k = 64$ будет выведено число 7.

3) Если увеличить значение k на единицу, то при $i = 8$ получаем:

i	$i \cdot i$	Условие	k
9	81	$9 > 0;\ 81 \geqslant 65$	65
8	64	$8 > 0;\ 64 < 65$	

То есть условие цикла будет нарушено уже при $i = 8$ (на экран вместо числа 7 будет выведено число 8). То же будет и при дальнейшем увеличении k. Следовательно, $k = 64$ — это наибольшее возможное значение.

4) А что будет, если уменьшать k? Очевидно, что программа будет работать так же, как и при $k = 64$, пока k не станет равно 49 (т.е. значению функции f при $i = 7$). В этом случае программа будет работать следующим образом:

i	$i \cdot i$	Условие	k
12	144	$12 > 0;\ 144 \geqslant 49$	49
11	121	$11 > 0;\ 121 \geqslant 49$	
10	100	$10 > 0;\ 100 \geqslant 49$	
9	81	$9 > 0;\ 81 \geqslant 49$	
8	64	$8 > 0;\ 64 \geqslant 49$	
7	49	$7 > 0;\ 49 \geqslant 49$	
6	36	$6 > 0;\ 36 < 49$	

В результате на экран вместо числа 7 будет выведено число 6.

Следовательно, уменьшать значение k можно только до 50.

5) Делаем вывод: программа работает одинаково (с выводом на экран числа 7) при значениях k из интервала от 50 до 64. Таких значений — $(64 - 50 + 1) = 15$ (единица прибавляется, так как мы определяем количество чисел в диапазоне, в который входят обе его границы).

Решение (*второй способ*)

1) Так же, как и в предыдущем случае, начинаем с трассировки программы. Это позволяет определить «граничные» значения i: последнее, при котором цикл ещё выполняется ($i = 8$), и первое, при котором цикл перестаёт выполняться ($i = 7$).

2) В соответствии с этим записываем два неравенства, левые части которых представляют собой выражение из условия цикла (оно же — выражение, записанное в теле функции f) с подстановкой найденных значений i, а в правых частях записывается переменная k:

- $8 \cdot 8 \geqslant k$ (так как при $i = 8$ это условие цикла истинно);

- $7 \cdot 7 < k$ (так как при $i = 7$ условие $i \cdot i \geqslant k$ ложно, поэтому знак в нём меняется на противоположный с заменой нестрогого неравенства на строгое).

3) Получаем систему неравенств и решаем её:

$$\begin{cases} 8 \cdot 8 \geqslant k; \\ 7 \cdot 7 < k; \end{cases} \Rightarrow \begin{cases} 64 \geqslant k; \\ 49 < k; \end{cases} \Rightarrow \begin{cases} k \leqslant 64; \\ k > 49. \end{cases}$$

4) Решением этой системы неравенств является интервал $k = (49,64]$. Обратим внимание на то, что в этом интервале левая граница в него не входит (соответствующее неравенство — строгое).

5) Определяем количество значений k, входящих в этот интервал. Оно равно $(64 - 49) = 15$.

Ответ: 15.

Задача 13. Определите наименьшее возможное значение входной переменной x, при котором программой будет получен ответ 31.

```
var k, i : longint;
function f(n: longint): longint;
begin
  f := n * n * n;
end;
function g(n: longint): longint;
begin
   g := n*n;
end;
begin
   readln(x);
   i := 1;
   while f(i) <= x*g(i) do
      i := i+1;
   writeln(i);
end.
```

Решение

1. Учитывая, что функции f(i) и g(i) не обращаются друг к другу, можно просто подставить соответствующие выражения в основную программу вместо вызова этих функций:

```
readln(x);
i := 1;
while i*i*i <= x*i*i do
   i := i + 1;
writeln(i);
```

2. Очевидно, что вывод результата 31 означает, что выполнение цикла `while` будет остановлено, когда условие $31^3 \leqslant x \cdot 31^2$ ложно, хотя условие $30^3 \leqslant x \cdot 30^2$ всё ещё истинно.

Тогда можно получить следующую систему неравенств:

$$\begin{cases} 30^3 \leqslant x \cdot 30^2; \\ 31^3 > x \cdot 31^2. \end{cases}$$

3. Решаем эту систему неравенств:

$$\begin{cases} 30^3 \leqslant x \cdot 30^2; \\ 31^3 > x \cdot 31^2; \end{cases} \Rightarrow \begin{cases} 30 \leqslant x; \\ 31 > x; \end{cases} \Rightarrow \begin{cases} x \geqslant 30; \\ x < 31. \end{cases}$$

Решение очевидно: $x = 30$.

Ответ: 30.

Задача 14. Укажите количество различных значений входной переменной k, при которых программа выдаёт тот же ответ, что и при входном значении $k = 55$. Само значение 55 тоже включается в подсчёт.

```
var k, i: integer;
Function F(x: integer): integer;
 begin
  F := x*x
 end;
```

```
begin
 readln(k);
 i := 1;
 while (F(i) <= k) do i := 2*i;
 if 2*F(i) - k > F(i-1) + k then writeln(i*2)
 else writeln(i)
end.
```

Решение

1. Анализируем алгоритм:

— функция F выполняет возведение переданного ей значения в квадрат;

— цикл работает «вхолостую», меняя только значение цикловой переменной (умножая её на 2), пока истинно условие $F(i) \leqslant k$;

— уже после завершения цикла выполняется оператор `if`: проверяется условие $2 \cdot F(i) - k > F(i-1) + k$ и, в зависимости от его истинности или ложности, выводится либо само значение i, полученное после завершения цикла, или утроенное значение i.

 В отличие от предыдущих задач, где *всегда* выводилось значение i, в этой программе требуемое нам значение может быть выведено не только на том же самом шаге цикла, что и при заданном значении k, но и на другом шаге цикла, если истинность условия окажется иной.

2. Сначала ищем, при каких k выполняется какой шаг цикла и на каком шаге цикл завершается для заданного $k = 55$. Для этого составляем таблицу. Начинаем с заданного в программе начального значения цикловой переменной i, для него вычисляем значение $F(i)$, а затем подбираем, какие натуральные числовые значения k обеспечивают ложное условие цикла и, соответственно, его завершение: $F(i) > k$ или, что то же самое, $k < F(i)$. Следующие значения i вычисляются умножением на 2, а ряд k каждый раз записывается

«с продолжением», т.е. значения k, найденные на предыдущем шаге, заново не повторяются.

i	1	2	4	8
$F(i) = i^2$	1	4	16	64
$k < F(i)$	—	1,2,3	4,5,…,14,15	16,17,…,**55**,…,62,63

На четвёртом шаге, для $i = 8$, мы в ряду значений k обнаруживаем заданное нам «опорное» значение $k = 55$. Значит, для этого значения k цикл завершается при $i = 8$.

3. Теперь проверяем заданное условие в операторе `if` для заданного $k = 55$ и найденного $i = 8$.

$2 \cdot F(i) - k > F(i-1) + k \Rightarrow 2 \cdot F(8) - 55 > F(8-1) + 55 \Rightarrow$

$\Rightarrow 2 \cdot 64 - 55 > 49 + 55 \Rightarrow 73 > 104.$

Это условие ложно, значит для $k = 55$ и $i = 8$ будет выведено число $i = 8$. Следовательно, нам нужно искать все такие k, при которых программа выводит число 8. Очевидно, оно может быть выведено в двух случаях:

1) при $i = 8$, если условие в `if` окажется ложным;

2) на другом шаге цикла, если условие вдруг окажется истинным.

4. Перепишем условие в `if`, чтобы выделить из него значение k:

$$2 \cdot F(i) - k > F(i-1) + k \Rightarrow 2 \cdot F(i) - F(i-1) > 2k \Rightarrow$$

$$\Rightarrow 2k < 2 \cdot F(i) - F(i-1) \Rightarrow k < \frac{2 \cdot F(i) - F(i-1)}{2}.$$

Теперь составим новую таблицу, в которой для различных значений i, для которых были ранее найдены наборы значений k, определяется значение $\frac{2 \cdot F(i) - F(i-1)}{2}$, записываются значения k, меньшие этого значения (тогда условие в `if` истинно) либо большие или равные этому значению (тогда условие ложно), а также записываются выводимые на экран значения i либо $i \cdot 2$. А затем в таблице мы выделяем искомое значение 8, где бы оно ни находилось, и выбираем соответствующие значения k.

i	1	2	4	8
$F(i) = i^2$	1	4	16	64
$k < F(i)$	—	1,2,3	4,5,…,14,15	16,17,…,**55**,…,62,63

i	$\frac{2 \cdot F(i) - F(i-1)}{2}$	$k < \frac{2 \cdot F(i) - F(i-1)}{2}$	Выводится $i \cdot 2$	$k \geqslant \frac{2 \cdot F(i) - F(i-1)}{2}$	Выводится i
2	3.5	1, 2, 3	4	—	—
4	11.5	**4, 5, …, 10, 11**	**8**	12, 13, 14, 15	4
8	39.5	16, 17, …, 38, 39	16	**40, 41, …, 62, 63**	**8**

Итак, искомое значение 8 выводится в двух случаях: для $k = [4, 11]$ и для $k = [40, 63]$. Остаётся подсчитать количество значений k:

— для $k = [4, 11]$ их $11 - 4 + 1 = 8$,

— для $k = [40, 63]$ их $63 - 40 + 1 = 24$,

— всего в сумме: $8 + 24 = 32$.

Ответ: 32.

В других подобных задачах возможны ситуации, когда заданное условие в `if` ***всегда*** истинно или же ***всегда*** ложно, и тогда искомое значение выводится только при одном каком-то наборе k.

Задача 15. Укажите количество различных значений входной переменной k, при которых программа выдаёт тот же ответ, что и при входном значении $k = 55$. Само значение 55 тоже включается в подсчёт.

```
var k, i: integer;
Function F(x: integer): integer;
 begin
  F := x*x
 end;
begin
 readln(k);
 i := 1;
 while (F(i) <= k) do i := 2*i;
```

```
  if 2*F(i)-k > F(i-1)+k then writeln(i*2)
  else writeln(i)
end.
```

 Всё «портит» добавленный после цикла условный оператор, который может выводить на экран либо само значение i, достигнутое после завершения цикла, либо значение $i \cdot 2$. Из-за этого решение становится существенно сложнее.

Решение

1. Анализируем алгоритм.

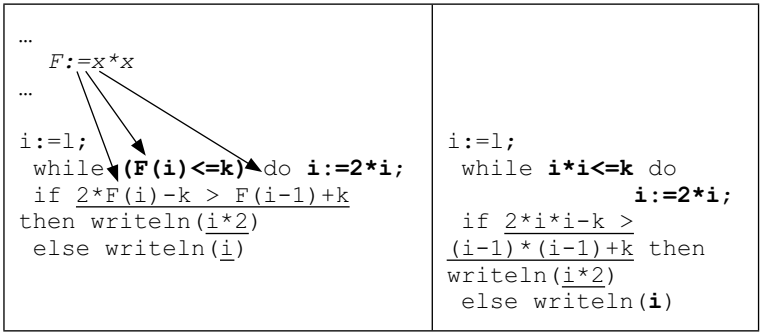
```
Function F(x: integer): integer;
 begin
  F := x*x
 end;
…
i := 1;
 while (F(i) <= k) do i := 2*i;
 if 2*F(i)-k > F(i-1)+k then writeln(i*2)
 else writeln(i)
```

1) *функция F выполняет возведение переданного ей значения в квадрат;*

2) цикл работает «вхолостую», меняя только значение цикловой переменной **(умножая её на 2)**, пока истинно условие **F(i) <= k;**

3) после завершения цикла выполняется оператор if: проверяется условие 2*F(i) – k > F(i-1) + k и, в зависимости от его истинности или ложности, выводится либо само значение *i*, полученное после завершения цикла, или удвоенное значение *i*.

2. Для удобства переписываем заданные нам условия цикла и ветвления, подставляя в них заданную функцию F.



Таким образом, мы можем отметить:

— цикл завершается, когда его условие становится ложным, т.е. **i*i > k**.

В заданном условии цикла знак неравенства меняется на противоположный, а строгое неравенство меняется на нестрогое и наоборот.

— цикл ещё в последний раз выполняется, когда его условие истинно, но при предыдущем значении *i*. Поскольку в цикле *i* умножается на два, предыдущее значение соответствует ***i*/2**. Тогда соответствующее неравенство: **(i/2)*(i/2) <= k**.

— в операторе `if` условие: `2*i*i – k > (i-1)*(i-1) + k`.

3. Найдём, при каком значении *i* заканчивается цикл, если задано $k = 55$.

Соответствующие неравенства см. выше. В них нужно подставить заданное значение k.

1) цикл завершается, когда: **i*i > k** ⇒ **i*i > 55**.

2) цикл ещё в последний раз выполняется, когда:

(i/2)*(i/2) <= k ⇒ **(i/2)*(i/2) <= 55** ⇒

⇒ **i*i/4 <= 55** ⇒ **i*i <= 220**.

Итак, имеем два неравенства: **i*i > 55** и **i*i <= 220**.

При этом помним, что *i* изначально равно 1 и каждый раз удваивается, т. е. возможные значения *i* равны: 1, 2, 4, 8, 16, ...

В промежутке от 55 до 220 находится только одно значение квадрата допустимого значения *i*: 64, т.е. 8^2. Значение $4^2 = 16$ слишком мало, а значение $16^2 = 256$ — слишком велико.

Тогда получаем, что для $k = 55$ цикл завершается при $i = 8$.

4. Проверяем, истинно ли условие в операторе `if` при полученных значениях $k = 55$ и $i = 8$.

Соответствующее неравенство см. выше. В него нужно подставить заданное значение *k* и полученное значение *i*. В зависимости от истинности полученного неравенства выводится или значение *i* · 2, если условие истинно, или значение *i*, если условие ложно:

```
if 2*F(i)-k > F(i-1)+k then writeln(i*2) else
      writeln(i)
```

```
2*i*i-k > (i-1)*(i-1)+k ⇒
⇒ 2*8*8-55 > (8-1)*(8-1)+55 ⇒ 73 > 104
```

Это неравенство (условие) ложно. Следовательно, при заданном значении $k = 55$ программа выводит значение $i = 8$.

5. Теперь нужно найти, в каких случаях программа может вывести то же самое значение 8. Очевидно, это может произойти в двух случаях:

а) если условие в операторе `if` ложно и выводится значение *i*, равное 8;

б) если условие в операторе `if` истинно и выводится значение *i**2, равное 8. Последнее означает, что $i = 4$, т.е. цикл завершился на один шаг раньше.

6 а). Ищем все возможные значения *k* в первом из указанных случаев, т.е. когда $i = 8$ и условие в операторе `if` ложно.

Помним, что цикл завершается, когда: `i*i > k`, и что цикл ещё в последний раз выполняется, когда:

(i/2)*(i/2) <= k. Подставляем в эти неравенства найденное значение i = 8.

1) цикл завершается при **i*i > k** $\Rightarrow$ **8*8 > k** $\Rightarrow$ $\Rightarrow$ **k < 64.**

2) цикл ещё в последний раз выполняется, когда: **(i/2)*(i/2)<=k** $\Rightarrow$ **(8/2)*(8/2) <= k** $\Rightarrow$ **4*4 <= k** $\Rightarrow$ $\Rightarrow$ **k >= 16.**

3) тогда допустимые значения k лежат в интервале: **[16, 64).**

Заметим, что граничное значение 64 в интервал не входит, так как соответствующее неравенство — строгое.

4) Однако, при этом условие в операторе `if` должно быть ложным, т. е. `2*i*i-k <= (i-1)*(i-1) + k`.

Берём исходное условие в операторе `if` и заменяем в нем знак неравенства на противоположный, а строгое неравенство заменяем на нестрогое и наоборот.

Подставляя в это неравенство значение $i = 8$, получаем: `2*i*i - k <= (i-1)*(i-1) + k` $\Rightarrow$
$\Rightarrow$ `2*8*8 - k <= (8-1)*(8-1) + k` $\Rightarrow$
$\Rightarrow$ `128 - k <= 49 + k` $\Rightarrow$ `79 <= 2*k` $\Rightarrow$
$\Rightarrow$ `2*k >= 79` $\Rightarrow$ `k >= 39,5`. Учитывая же, что значение k должно быть целым, получаем: **k >= 40**

5) В итоге имеем: $k \in$ **[16, 64)** и **k >= 40**, тогда $k \in$ **[40, 64).**

6 б). Ищем все возможные значения k во втором из указанных случаев, т. е. когда $i = 4$ и условие в операторе `if` истинно.

Помним, что цикл завершается, когда: i*i > k, и что цикл ещё в последний раз выполняется, когда: (i/2)*(i/2) <= k. Подставляем в эти неравенства найденное значение $i = 4$.

1) цикл завершается при **i*i > k** $\Rightarrow$ **4*4 > k** $\Rightarrow$ $\Rightarrow$ **k < 16.**

2) цикл ещё в последний раз выполняется, когда: **(i/2)*(i/2) <= k** $\Rightarrow$ **(4/2)*(4/2) <= k** $\Rightarrow$
$\Rightarrow$ **2*2 <= k** $\Rightarrow$ **k >= 4.**

3) тогда допустимые значения k лежат в интервале: **[4, 16).**

Заметим, что граничное значение 16 в интервал не входит, так как соответствующее неравенство — строгое.

4) При этом условие в операторе `if` должно быть истинным, т. е. `2*i*i - k > (i-1)*(i-1) + k`.

Подставляя в это неравенство значение $i = 4$, получаем: `2*i*i - k > (i-1)*(i-1) + k` $\Rightarrow$
$\Rightarrow$ `2*4*4 - k > (4-1)*(4-1) + k` $\Rightarrow$
$\Rightarrow$ `32 - k > 9 + k` $\Rightarrow$ `23 > 2*k` $\Rightarrow$ `2*k < 23` $\Rightarrow$
$\Rightarrow$ `k < 11,5`. Учитывая, что значение k должно быть целым, получаем: **`k <= 11`.**

5) В итоге имеем: $k \in$ **[4, 16)** и **`k <= 11`**, тогда $k \in$ **[4, 11].**

В конкретных задачах возможно, что в случае (а) или (б) решения не будет, так как условие, полученное из оператора `if`, не совместно с интервалом значений k, полученных из анализа работы цикла. Тогда последующий подсчёт количества значений k выполняется только для существующего интервала (для одного из случаев).

7. Подсчитываем количества допустимых значений k в каждом из двух случаев.

а) При $k \in$ **[40, 64):** $64 - 40 = 24$.

б) При $k \in$ **[4, 11]:** $11 - 4 + 1 = 8$.

В сумме это дает $24 + 8 = 32$ различных значения k.

Вычислить количество значений в заданном интервале можно по следующим правилам:

— если обе границы интервала входят в него, то количество значений равно разности большего и меньшего граничных значений плюс единица;

— если одна из границ интервала не входит в него, то количество значений равно разности большего и меньшего граничных значений;

— если обе границы интервала не входят в него, то количество значений равно разности большего и меньшего граничных значений минус единица.

Ответ: 32.

Задача 16. Найти **наибольшее** значение входной переменной k, при котором программа выдаёт ответ 9.

```
var k, i: longint;
function f(n: longint): longint;
 begin
   f := n * n * n
 end;

begin
   readln(k);
   i := 20;
   while f(i) > k do
      i := i - 1;
   writeln(i)
end.
```

Решение

Такие задачи нам уже знакомы. Но раньше требовалось искать значения переменной k, такие же, какие программа выдаёт при некотором заданном k. А теперь ответ программы приведён в условии явно. Как мы увидим далее, это изменение заметно упрощает решение.

1. Используемая функция f соответствует n^3.

2. Цикл прекращает выполняться, когда его условие будет нарушено, т.е. при $f(i) \leqslant k$. Цикл ещё выполнялся, когда условие было ещё истинным, но при предыдущем значении цикловой переменной, т.е. при $i + 1$. Получаем неравенство: $f(i + 1) > k$. Соединяем эти два неравенства в одно двойное по одинаковому k (второе неравенство для этого потребуется «зеркально развернуть», меняя его знак): $f(i) \leqslant k < f(i + 1)$.

3. Из текста программы видим, что выводится значение i, полученное сразу после завершения цикла, т.е. как раз то самое, которое рассматриваем в нашем неравенстве. И оно равно 9.

4. Подставляем в неравенство запись функции f (в данном случае — i^3) и заданное значение i:

$i^3 \leqslant k < (i+1)^3 \Rightarrow 9^3 \leqslant k < (9+1)^3 \Rightarrow 9^3 \leqslant k < 10^3 \Rightarrow$

$\Rightarrow 729 \leqslant k < 1000 \Rightarrow k \in [729, 1000)$.

Заметим, что верхняя граница в интервал **не входит** (соответствующий знак неравенства — строгий!).

5. Нам требуется наибольшее возможное значение k. Поскольку значение 1000 в найденный интервал не входит, берём предыдущее значение $k = 999$.

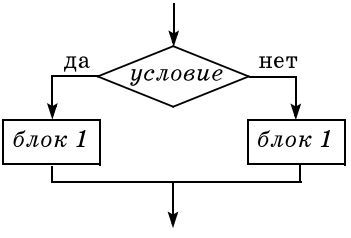
Ответ: 999.

Так как нам требовалось только наибольшее значение, определяемое верхней границей интервала, нижнюю границу (9^3) можно было бы не вычислять, сэкономив время.

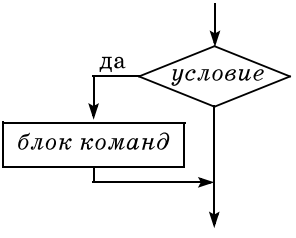
Задачи на исправление ошибок в программах

Конспект

Полная конструкция «Ветвление»



Неполная конструкция «Ветвление»



Конструкция ветвления в языке Паскаль

Полная	Неполная

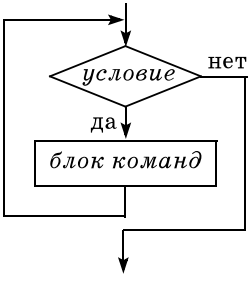
if (<условие>) then
  begin
  <блок 1>
  end
else
  begin
  <блок 2>
  end;
	if (<условие>) then
	  begin
	  <блок>
	  end;

Если блоки команд состоят только из одного оператора каждый, то использовать операторные скобки `begin … end` не обязательно:

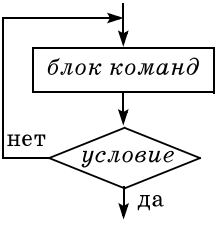
Полная	Неполная

if (<условие>) then
<команда 1>
else
<команда 2>;
	if (<условие>) then
	<команда>;

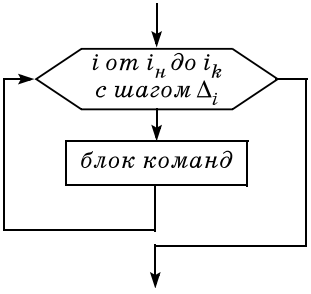
Цикл с предусловием (цикл ПОКА)



Цикл с постусловием (цикл ДО)



Цикл с параметром (цикл со счётчиком, цикл ДЛЯ)



Конструкции циклов в языке Паскаль

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;
	for <i> := <$i_к$> downto <$i_н$> do
	  begin
	  <блок команд>
	  end;

или	или
for <i> := <$i_н$> to <$i_к$> do <команда>;	for <i> := <$i_к$> downto <$i_н$> do <команда>;

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. На обработку поступает последовательность из четырёх неотрицательных целых чисел (некоторые числа могут быть одинаковыми). Нужно написать программу, которая выводит на экран количество нечётных чисел в исходной последовательности и максимальное нечётное число. Если нечётных чисел нет, требуется вывести на экран «NO». Известно, что вводимые числа не превышают 1000. Программист написал программу неправильно.

```
const n = 4;
var i, x: integer;
var maximum, count: integer;
begin
   count := 0;
   maximum := 999;
   for i := 1 to n do
    begin
      read(x);
      if x mod 2 <> 0 then
       begin
         count := count + 1;
         if x > maximum then
```

```
            maximum := i
          end
        end;
      if count > 0 then
        begin
          writeln(count);
          writeln(maximum)
        end
      else
          writeln('NO')
  end.
```

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе последовательности: 2 9 4 3.

2. Приведите пример такой последовательности, содержащей хотя бы одно нечётное число, что, несмотря на ошибки, программа печатает правильный ответ.

3. Найдите все ошибки в этой программе (их может быть одна или несколько). Известно, что каждая ошибка затрагивает только одну строку и может быть исправлена без изменения других строк. Для каждой ошибки:

1) выпишите строку, в которой сделана ошибка;

2) укажите, как исправить ошибку, т.е приведите правильный вариант строки.

Обратите внимание, что требуется найти ошибки в имеющейся программе, а не написать свою, возможно, использующую другой алгоритм решения. Исправление ошибки должно затрагивать только строку, в которой находится ошибка.

Решение

1) Начинаем с общего анализа алгоритма, помня, что в нём возможны отдельные ошибки. (Что делает программа, описано в условии, но умение анализа алгоритма требуется нам как для поиска ошибок в нём, так и потому, что в последующих вариантах ЕГЭ назначение программы в условии может быть не указано — в качестве дополнительного усложнения задания.)

- В начале программы инициализируются (получают начальные значения) две переменных — *count* (обнуляется) и *maximum* (приравнивается 999). Уже из названия этих переменных можно предположить, что *count* — это счётчик чего-либо, а переменная *maximum* предназначена для определения максимального значения.
- Далее следует цикл с параметром, выполняемый 4 раза (константное значение *n* задаётся равным 4):
  — считывается очередное значение *x*,
  — проверяется его нечётность (неравенство нулю остатка при делении на 2): если *x* нечётно, то счётчик увеличивается на 1, а конструкция имеет вид:
  `if x > maximum then maximum := ...`,
  — это хорошо знакомый приём определения максимума,
  — случай чётного *x* не обрабатывается (ветвь else отсутствует), проход цикла завершается;
- после цикла проверяется счётчик, если он больше нуля, то выводятся значения *count* и *maximum*, иначе выводится сообщение «NO».

Итак, назначение и общие принципы работы программы ясны. Принимая на входе последовательность чисел, она должна определять максимальное значение среди нечётных таких чисел и одновременно подсчитывать количество нечётных чисел.

2) Сразу же можно понять скрытые в тексте программы ошибки (тем самым отвечая на третий вопрос в задании).

Первая ошибка — в определении максимума:

```
if x > maximum then maximum := i
```

В типовом алгоритме поиска максимума (и аналогично для минимума) если текущее значение последовательности больше предполагаемого максимума, то предполагаемому максимуму присваивается это боль-

шее него текущее значение. То есть *x*. А здесь переменной *maximum* присваивается значение *i*.

Строка с ошибкой: `maximum := i`

Правильная строка: `maximum := x`

Согласно условию, в качестве ошибочной можно указать только одну строку. Поэтому оператор `if x > maximum then` (формально правильный) в запись ответа не входит.

Вторая ошибка — в начальном присваивании переменной *maximum* значения 999. В качестве исходного значения предполагаемого *максимума* нужно брать число, *меньшее самого меньшего возможного числа в последовательности или в массиве*. В условии указано, что числа вводимой последовательности — неотрицательные, значит, в качестве изначального предполагаемого максимума можно взять число 0 или любое отрицательное значение.

Строка с ошибкой: `maximum := 999;`

Правильная строка: `maximum := 0;`

3) Что программа выведет при вводе последовательности 2 9 4 3?

Программа, несмотря на ошибки, верно выполняет подсчёт количества нечётных чисел (в строках с переменной *count* ошибок нет). Поэтому значение *count* (первое выводимое на экран число) определится верно и будет равно 2 (так как в последовательности — два нечётных числа).

Что же касается поиска максимума, нетрудно догадаться: из-за того, что первоначально переменной *maximum* присвоено значение, заведомо большее любого из введённых чисел, условие $x > maximum$ никогда не будет истинным. Так что значение *maximum* так и останется равным 999 и будет выведено на экран вторым числом.

Теперь мы можем дать ответ на первый вопрос задания. При вводе последовательности чисел 2 9 4 3 программа выведет два числа: 2 и 999.

4) Определить, в каком случае программа, несмотря на ошибки, выдаст правильный ответ, лишь немного сложнее.

Так как строки с переменной *count* не содержат ошибок, количество нечётных чисел определяется верно. Следовательно, в вводимой последовательности может быть любое количество нечётных значений. Возможно даже, что все числа последовательности — чётные, например: 2 4 6 8. В этом случае *count* окажется равно нулю, программа выдаст ответ «NO», а ошибки в определении максимума будут просто «обойдены».

Если же в введённой последовательности есть хотя бы одно нечётное число, то нужно «подогнать» эту последовательность так, чтобы выводимое значение *maximum*, равное 999, было правильным. Очевидно, это возможно, если в последовательности действительно будет иметься хотя бы одно число (нечётное!), которое равно 999, и причём это число действительно будет максимальным среди прочих нечётных значений. А значит, правильными будут, например, следующие последовательности:

2 4 6 999 (единственное нечётное число равно 999),

2000 4444 88888 999 (хотя остальные числа превышают 999, они — чётные, поэтому в определении максимума среди нечётных чисел не участвуют),

1 3 5 999 (999 — максимальное значение среди всех нечётных),

999 999 999 999 (все числа равны 999, следовательно, и максимум тоже равен 999).

Любой такой ответ (равно как и последовательность только из чётных чисел) — правилен.

 В качестве ответа на второй вопрос задания можно записать любую последовательность, в которой максимальное нечётное значение равно 999, либо в которой нет ни одного нечётного значения.

Ответ:

1) При вводе последовательности 2 9 4 3 программа выведет два числа: 2 и 999.

2) Программа выдаёт правильный результат, например, при вводе последовательности 1 2 3 999.

3) Ошибки в программе и их исправление (по порядку следования строк в программе):

Строка с ошибкой: `maximum := 999;`
Правильная строка: `maximum := 0;`

Строка с ошибкой: `maximum := i`
Правильная строка: `maximum := x`

В подобных заданиях важно не только указать все имеющиеся ошибки, но и не указать в качестве «ошибочной» правильную строку.

Как правило, в таких задачах под ошибками понимаются существенные искажения алгоритма, приводящие к тому, что программа не выполняет заявленное для неё предназначение. Искомыми ошибками не являются:

- использование оператора write или read вместо writeln или readln соответственно и прочие необязательные элементы функционала программы, связанные с выводом ответа на экран (кроме, например, указания в операторе вывода не той переменной, которая нужна);
- отсутствие точки с запятой в конце оператора и т.д.

Задача 2. Требовалось написать программу, при выполнении которой с клавиатуры считывается положительное целое число N, не превосходящее 10^9, и определяется сумма цифр этого числа. Программист торопился и написал программу неправильно.

```
var N: longint;
sum, d: integer;
begin
readln(N);
sum := 1;
while N > 0 do
```

```
begin
        d := N mod 10;
        N := N div 10;
        sum := d;
end;
writeln(sum);
end.
```

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе числа 256.

2. Приведите пример такого трёхзначного числа, при вводе которого программа выдаёт правильный результат.

3. Найдите все ошибки в этой программе (их может быть одна или несколько). Для каждой ошибки:

1) выпишите строку, в которой сделана ошибка;

2) укажите, как исправить ошибку, т.е. приведите правильный вариант строки.

Достаточно указать ошибки и способ их исправления для одного языка программирования. Обратите внимание, что требуется найти ошибки в имеющейся программе, а не написать свою, возможно, использующую другой алгоритм решения. Исправление ошибки должно затрагивать только строку, в которой находится ошибка.

Решение

1) Анализируем алгоритм.

- Вводится число N.
- Инициализируется переменная *sum*, в которой будет накапливаться искомая сумма (ей присваивается значение 1).
- Цикл выполняется, пока значение N остается больше нуля:
  — оператор `d := N mod 10;` — в переменную d заносится последняя цифра числа N,

— оператор `N := N div 10;` — отбрасывание последней цифры числа,
— оператор `sum := d;` — записывает в переменную *sum* очередную цифру числа;

- После выполнения цикла значение *sum* выводится на экран.

2) Очевидна первая ошибка: значение суммы цифр должно *накапливаться* в переменной *sum*, т.е. соответствующий оператор должен иметь вид:

`sum := sum + d;`, а не `sum := d;`.

Строка с ошибкой: `sum := d;`

Правильная строка: `sum := sum + d;`

Вторая ошибка тоже понятна: при инициализации переменной, в которой накапливается сумма, эту переменную нужно обнулять (единица — это инициализационное значение для произведения).

Строка с ошибкой: `sum := 1;`

Правильная строка: `sum := 0;`

3) При вводе числа 256 программа разобьёт его на отдельные цифры: 6, 5 и 2. Поскольку на каждом проходе цикла в переменную *sum* записывается очередная цифра, после выполнения цикла значение *sum* будет равно последней обработанной цифре. Цифры отделяются и обрабатываются по порядку справа налево, поэтому *sum* после выполнения цикла будет равно 2, это число и будет выведено на экран.

4) Выданный программой результат будет правильным, если последняя обработанная цифра числа будет в точности равна сумме цифр этого числа. Следовательно, все остальные цифры (кроме самой первой в числе) должны быть нулевыми. Количество же цифр в числе может быть любым. Например, это могут быть числа: 2, 30, 4000 и т.д. Однако поскольку значение *sum* изначально инициализируется неверно (этой переменной присваивается значение 1 вместо 0), вводимое число не может быть нулём (в этом случае программа выведет неправильный ответ 1 вместо правильного 0).

Ответ:

1) При вводе числа 256 программа выведет число 2.

2) Вводимое число, при котором программа выдаёт правильный ответ, может быть, например, равно 100.

3) Ошибки в программе и их исправление (по порядку следования строк в программе):

Строка с ошибкой: `sum := 1;`

Правильная строка: `sum := 0;`

Строка с ошибкой: `sum := d;`

Правильная строка: `sum := sum + d;`

Задачи на анализ и обработку данных

Конспект

Перебор всех пар соседних элементов

Перебор всех пар соседних элементов массива выполняется единственным циклом:

```
for i := 1 to N-1 do
    <тело цикла: A[i] в паре с A[i+1]>
```

Перебор всех возможных пар элементов массива

При просмотре всех возможных пар элементов массива нужно исключить из рассмотрения «пары» из одного и того же элемента «самого с собой», а также исключить повторное рассмотрение одних и тех же пар элементов: например, пара из элементов с номерами 5 и 8 и пара из элементов с номерами 8 и 5 — это одна и та же пара.

С учётом вышеуказанных требований перебор **всех** пар элементов массива без их повторения реализуется структурой из двух вложенных циклов:

```
for i := 1 to N - 1 do
  for j := (i+1) to N do begin
 <тело цикла: A[i] в паре с A[j]>
end;
```

При этом:

— элемент A[1] поочерёдно рассматривается в паре с элементами A[2], A[3], ... A[N];

— элемент A[2] поочерёдно рассматривается в паре с элементами A[3], A[4], ... A[N];

...

— элемент A[N−1] рассматривается в паре только с элементом A[N].

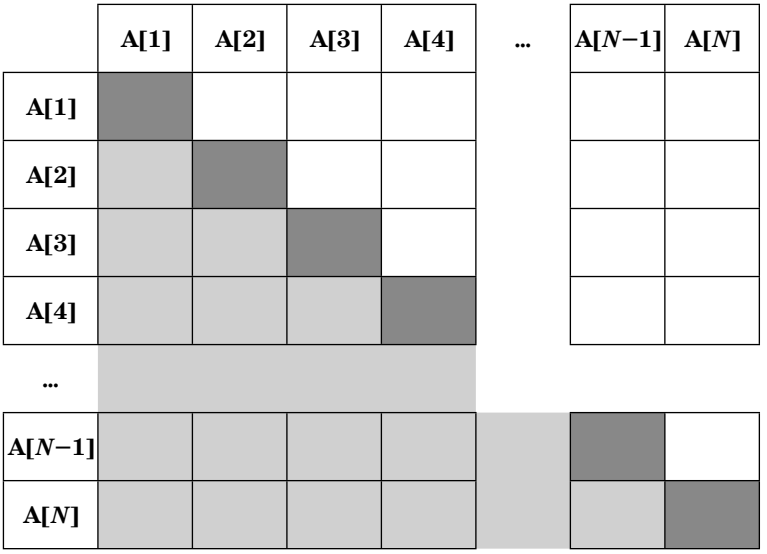
Количество всех возможных пар элементов массива либо набора чисел

Количество всех возможных пар элементов, если количество самих этих элементов равно N, может быть вычислено по формуле: $\frac{N^2 - N}{2}$.

Объяснение

Представим N чисел как элементы линейного массива длиной N.

Создадим квадратную таблицу размера $N \times N$, в которой и строки, и столбцы соответствуют элементам исходного линейного массива.



- Очевидно, что каждая ячейка этой таблицы соответствует рассмотрению пары каких-либо из N элементов массива. При этом общее количество таких ячеек равно N^2.
- Ячейки, лежащие на главной диагонали таблицы, соответствуют рассмотрению каждого из N элементов «самого с собой». Таких ячеек N штук, и они должны быть исключены из рассмотрения. Остаётся $(N^2 - N)$ ячеек, соответствующих парам **различных** элементов.
- Однако при этом нас интересует только половина таблицы — например, ячейки, расположенные над главной диагональю, поскольку все ячейки под главной диагональю — это повторы тех же самых пар элементов. Поэтому вычисленное ранее количество пар необходимо разделить на 2.

Получаем, что количество не повторяющихся пар из N элементов равно $\frac{N^2 - N}{2}$.

Понятие «эффективность программы»

«Эффективность программы по памяти» означает, что необходимо использовать минимальное количество переменных. Объявление массива из 1000 элементов, если количество входных чисел может быть очень небольшим, явно неэффективно. Однако в данном случае речь идёт о необходимости найти и реализовать такой алгоритм обработки поступающих на вход чисел, чтобы вообще не хранить сами эти числа, а только лишь накапливать определённые данные в нескольких выделенных для этого переменных — счётчиках. В этом случае объём памяти, требуемый для работы программы, не будет зависеть от количества поступающих на вход значений.

«Эффективность программы по времени» означает, что время выполнения программы не зависит от количества входных данных либо зависит от него линейно (обработка каждого значения занимает некоторое вре-

мя t, а для N входных данных время работы программы может быть оценено как произведение $t \times N$). В неэффективной же программе время работы может с увеличением количества входных данных возрастать нелинейно — например, по квадратичному закону (t^2).

Программа, предполагающая запись всех поступающих на вход чисел в массив, а затем — перебор всех пар таких элементов с помощью конструкции из двух вложенных циклов, является неэффективной:

— *по памяти* — из-за использования массива длиной 1000 (к тому же, скорее всего, не заполненного целиком);

— *по времени* — так как с ростом значения N количество выполнений тела вложенного цикла возрастает нелинейно (зависимость не квадратичная, так как реализуется не полный перебор пар каждого из N элементов с каждым, но степенная). Для получения эффективной по времени программы необходимо в алгоритме обработки отказаться от вложенных циклов и реализовать только единичный цикл ввода входных данных и анализа каждого из них.

Разбор типовых задач

Задача 1. На вход программы поступает последовательность из N целых положительных чисел, все числа в последовательности различны.

Рассматриваются все пары различных элементов последовательности (элементы пары не обязаны стоять в последовательности рядом, порядок элементов в паре не важен). Необходимо определить количество пар, для которых произведение элементов делится на 22.

Описание входных и выходных данных

В первой строке входных данных задаётся количество чисел N ($1 \leqslant N \leqslant 1000$). В каждой из последующих N строк записано одно целое положительное число, не превышающее 10 000.

В качестве результата программа должна напечатать одно число: количество пар, в которых произведение элементов кратно 22.

Пример входных данных:

4
2
6
11
33

Пример выходных данных для приведённого выше примера входных данных:

4

Пояснение. Из четырёх заданных чисел можно составить 6 попарных произведений: 2·6, 2·11, 2·33, 6·11, 6·33, 11·33 (результаты: 12, 22, 66, 66, 198, 363). Из них на 22 делятся 4 произведения (2·11 = 22; 2·33 = 66; 6·11 = 66; 6·33 = 198).

Требуется написать эффективную по времени и по памяти программу для решения описанной задачи.

Программа считается эффективной по времени, если при увеличении количества исходных чисел N в k раз время работы программы увеличивается не более чем в k раз.

Программа считается эффективной по памяти, если память, необходимая для хранения всех переменных программы, не превышает 1 Кбайт и не увеличивается с ростом N.

Максимальная оценка за правильную (не содержащую синтаксических ошибок и дающую правильный ответ при любых допустимых входных данных) программу, эффективную по времени и по памяти, — 4 балла.

Максимальная оценка за правильную программу, эффективную только по времени — 3 балла.

Максимальная оценка за правильную программу, не удовлетворяющую требованиям эффективности, — 2 балла.

Вы можете сдать **одну** программу или **две** программы решения задачи (например, одна из программ может быть менее эффективна). Если вы сдадите две программы, то каждая из них будет оцениваться независимо от другой, итоговой станет **бóльшая** из двух оценок.

Перед текстом программы обязательно кратко опишите алгоритм решения.

Укажите использованный язык программирования и его версию.

Решение

1. Количество поступающих на вход чисел задано (не превышает 1000). Соответственно, при решении задачи в программе может быть использован цикл `for`, а также (для неэффективного варианта) — массив из 1000 элементов, часть из которых может оказаться не используемыми.

2. Требуется перебор **всех** возможных пар входных чисел (в отличие от заданий, в которых речь идёт только лишь о парах соседних элементов).

Неэффективная программа (*2 балла*)

Неэффективный алгоритм заключается в записи всех поступающих на вход чисел в линейный массив, объявленный «с запасом» на 1000 элементов (согласно заданному условию на количество входных данных) с последующим перебором всех неповторяющихся пар элементов при помощи конструкции из двух вложенных циклов. В теле внутреннего цикла выполняется вычисление произведения элементов очередной пары и определение делимости этого произведения на заданное значение (22). При выполнении этого условия значение переменной-счётчика увеличивается на 1.

По завершении работы вложенных циклов переменная-счётчик будет содержать искомое количество пар значений, удовлетворяющих заданному условию.

```
var a: array [1..1000] of integer;  // массив
                                    // для
                                    // хранения
                                    // чисел
    N: integer;                 // количество
                                // входных
                                // значений
    K: integer;                 // количество
                                // пар
    i: integer;
begin
  readln(N);
  for i := 1 to N do
    readln(a[i]);
  k := 0;
  for i := 1 to n - 1 do
    for j := i + 1 to n do
      if a[i]*a[j] mod 22 = 0 then k := k + 1;
  writeln(k);
end.
```

Эффективная программа (*4 балла*)

Можно ли отказаться от использования массива и вложенных циклов и определить количество пар, кратных заданному значению 22, не рассматривая сами пары и не вычисляя произведения входящих в них элементов?

Рассмотрим произведение двух целых чисел.

Оно делится на требуемое число при выполнении одного из следующих условий:

— оба перемножаемых числа делятся на требуемое значение (22);

— хотя бы одно из перемножаемых чисел делится на требуемое значение (22);

— одно из чисел делится на один из сомножителей требуемого значения, а другое — на другой сомножитель этого значения: в данном случае — если одно число

делится на 2, а другое — на 11. (Если заданное значение представляет собой простое число, то данное условие не рассматривается.)

При поступлении на вход очередного числа будем определять, делится ли оно:

— на требуемое значение (22),

— на каждый из сомножителей требуемого значения (для 22 — на 2 и на 11), но при этом не кратных самому значению 22, — чтобы исключить возможные повторы.

Соответственно, будем в выделенных переменных отдельно подсчитывать количества входных чисел:

— кратных требуемому значению (22) — переменная $n22$,

— не кратных самому требуемому значению 22, но кратных первому сомножителю (2) — переменная $n2$,

— не кратных самому требуемому значению 22, но кратных второму сомножителю (11) — переменная $n11$.

Очевидно, что при таких подсчётах каждое входное число будет отражено только в одном из этих счётчиков, т. е. лишних повторов не будет.

Сами входные числа после проверки и изменения состояния соответствующего счётчика тут же «забываются» — не хранятся ни в каком массиве, а переменная, в которую было считано очередное число, на следующем проходе цикла чтения используется под следующее входное число. Это обеспечивает эффективность программы по памяти.

Теперь нужно по полученным значениям счётчиков определить искомое количество пар чисел.

1) Для чисел, из которых оба делятся на заданное значение 22, количество пар можно вычислить по формуле $\frac{n22 \times (n22 - 1)}{2}$, так как:

— если таких чисел имеется $n22$ штук, то всего пар этих чисел «каждый с каждым» будет $n22 \times n22$,

— однако из них $n22$ пар исключается, так как это — «пары из числа самого с собой»: $n22 \times n22 - n22 = n22 \times (n22 - 1)$,

— а поскольку *оба* числа кратны 22, то соответствующая пара будет посчитана дважды («по первому числу» и «по второму»), — значит, вычисленное количество надо разделить на 2.

2) Для чисел, в которых только одно число делится на требуемое значение 22, количество пар вычисляется по формуле $n22 \times (N - n22)$, так как любое из чисел, кратных 22, даёт нужную нам пару, но пары с числами, также кратными 22, мы уже подсчитали ранее. То есть всего пар из чисел, кратных 22, со всеми другими числами было бы $n22 \times N$, но $n22 \times n22$ пар (в том числе при сочетании числа «с самим собой») из рассмотрения нужно исключить.

3) Количество пар чисел, из которых одно делится на один из сомножителей требуемого значения (например, 2), а другое делится на второй сомножитель (11), легко вычислить просто как произведение количества чисел первого типа на количество второго типа, т.е. по формуле $n2 \times n11$.

(Если бы требуемое значение состояло из более чем двух сомножителей, то вычисление было бы более сложным.)

Общее же количество пар, удовлетворяющих условию «произведение чисел кратно 22», вычисляется как сумма подсчитанных выше количеств:

$n22 \times (n22 - 1)/2 + n22 \times (N - n22) + n2 \times n11$.

Таким образом, эффективный алгоритм может быть следующим:

— объявляем три переменных-счётчика для подсчёта количеств чисел, делящихся на 22, на 2 и на 11, а также одну «буферную» переменную для временного хранения очередного входного числа (объявление переменной N — количества входных чисел, а также цикловой переменной подразумевается);

— в линейном цикле по очереди считываем входные числа, проверяем на делимость и соответственно меняем состояние счётчиков;

— вычисляем и выводим искомое количество пар.

```
var N: integer; // количество чисел
     a: integer; // считываемое число
     n22, n11, n2: integer; // счётчики
     i: integer;
begin
   readln(N);
   n22 := 0; n11 := 0; n2 := 0;
   for i := 1 to N do begin
     readln(a);
     if a mod 22 = 0 then n22 := n22 + 1
     else begin
       if a mod 11 = 0 then n11 := n11 + 1
      else if a mod 2 = 0 then n2 := n2 + 1;
     end;
   end;
   writeln(n22*(n22-1) div
        2 + n22*(N - n22) + n2*n11);
end.
```

Возможна модификация данной задачи, в которой требуется определить количество пар, **не кратных** заданному значению. В этом случае нужно сначала, согласно вышеприведённому алгоритму, вычислить количество пар элементов, **кратных** указанному значению, а затем вычесть это количество из общего количества неповторяющихся пар.

Задача 2. Дан набор из N целых положительных чисел. Из этих чисел формируются все возможные пары (парой считаются два элемента, которые находятся на разных местах в наборе, порядок чисел в паре не учитывается), в каждой паре вычисляются сумма и произведение элементов. Необходимо определить количество

пар, у которых сумма нечётна, а произведение делится на 7.

Напишите эффективную по времени и по памяти программу для решения этой задачи.

Перед текстом программы кратко опишите алгоритм решения. Укажите использованный язык программирования и его версию.

Описание входных и выходных данных

В первой строке входных данных задаётся количество чисел N ($1 \leqslant N \leqslant 1000$).

В каждой из последующих N строк записано одно натуральное число, не превышающее 100.

Решение

В этой задаче, так же как и в предыдущей, речь идёт об анализе всех неповторяющихся пар вводимых чисел — изменено только условие, которым должны соответствовать эти пары. Поэтому решение в целом будет аналогичным.

Неэффективная программа (*2 балла*)

Неэффективный алгоритм заключается в записи всех поступающих на вход чисел в линейный массив, объявленный «с запасом» на 1000 элементов с последующим перебором всех неповторяющихся пар при помощи конструкции из двух вложенных циклов. В теле внутреннего цикла выполняется вычисление суммы и произведения элементов очередной пары и определение делимости произведения на 7, а также чётности или нечётности суммы. При выполнении условия значение переменной-счётчика увеличивается на 1.

По завершении работы вложенных циклов переменная-счётчик будет содержать искомое количество пар значений, удовлетворяющих заданному условию.

```
var N: integer; // количество чисел
    a: array[1..1000] of integer; // хранение
                                  // чисел
```

```
    s: integer; // счетчик
    i, j: integer;
begin
  readln(N);
  for i := 1 to N do readln(a[i]);
  s := 0;
  for i := 1 to N - 1 do
    for j := i + 1 to N do begin
      if ((a[i]+a[j]) mod 2 = 1) and
                        ((a[i]*a[j]) mod 7 = 0)
        then s := s + 1
  end;
  writeln(s);
end.
```

Эффективная программа (*4 балла*)

1) В каких случаях сумма двух целых чисел нечётна? Очевидно, если одно из них нечётное, а другое — чётное.

2) В каких случаях произведение двух целых чисел кратно 7? Аналогично предыдущей задаче — если оба этих числа делятся на 7 или если хотя бы одно из них делится на 7.

Заготовим переменные — счётчики для подсчёта количества входных чисел, которые:

— чётны и кратны 7 (переменная $m27$);

— чётны и не кратны 7 (переменная $m2$);

— нечётны и кратны 7 (переменная $m7$);

— нечётны и не кратны 7 (переменная m).

Как по ним подсчитать количество требуемых пар?

Нечётную сумму и кратность 7 дают все пары чисел, из которых:

1) первое нечётно и кратно 7, а второе чётно и тоже кратно 7 — таких пар будет $m7 \times m27$ (каждое из чисел первого типа берется в пару с каждым числом второго типа);

2) первое нечётно и не кратно 7, а второе чётно и кратно 7 — таких пар будет $m \times m27$;

3) первое чётно и не кратно 7, а второе нечётно и кратно 7 — таких пар $m2 \times m7$.

(Поскольку во всех перечисленных сочетаниях в пары берутся числа различных типов (в частности — только чётные с нечётными), автоматически исключается ситуация, когда число рассматривается «в паре с самим собой», а также повторные рассмотрения одних и тех же пар.)

Тогда общее количество подходящих по условию пар определяется как сумма количеств пар, вычисленных по каждому из трёх вышеперечисленных случаев: $m7 \times m27 + m \times m27 + m2 \times m7$.

Таким образом, эффективный алгоритм имеет следующий вид:

— объявляем четыре переменных-счётчика для подсчёта количеств чисел соответствующих типов (чётное — нечётное, кратное 7 или некратное), а также одну «буферную» переменную для временного хранения очередного входного числа, переменную N для количества входных чисел, а также цикловую переменную;

— в линейном цикле по очереди считываем входные числа, проверяем на чётность и делимость на 7 и соответственно меняем состояние счётчиков;

— вычисляем и выводим искомое количество пар.

```
var N: integer;          // количество чисел
    a: integer;          // считанное число
    m: integer;          // кол-во нечётных,
                         // не кратных 7
    m2: integer;         // кол-во чётных,
                         // не кратных 7
    m7: integer;         // кол-во нечётных,
                         // кратных 7
    m27: integer;        // кол-во чётных,
                         // кратных 7
    i: integer;
begin
  m := 0; m2 := 0; m7 := 0; m27 := 0;
```

```
  readln(N);
  for i :=1 to N do begin
    readln(a);
    if a mod 2 = 0 then begin
      if a mod 7 = 0 then m27 := m27 + 1
         else m2 := m2 + 1
    end
    else begin
      if a mod 7 = 0 then m7 := m7 + 1
      else m := m + 1
    end
  end;
  writeln(m7*m27 + m*m27 + m2*m7);
end.
```

Если в задаче потребуется определить количество пар, в которых сумма чётна и кратна некоторому числу, то для получения чётной суммы потребуется брать в пары либо оба чётных числа, либо оба нечётных. Кроме того, нужно не забыть, что кратность числа чётному числу автоматически означает чётность самого исходного числа.

 **Важно!** Для получения полной балльной оценки решения задач этого типа необходимо в качестве решения не только записать саму программу на выбранном языке программирования, но и сопроводить своё решение достаточно подробным и чётким словесным описанием алгоритма.

Задача 3. Дан набор из N целых положительных чисел. Из этих чисел формируются все возможные пары (парой считаются два элемента, которые находятся на разных местах в наборе, порядок чисел в паре не учитывается). Необходимо определить количество пар, в которых сумма чисел делится на 8.

Напишите эффективную по времени и по памяти программу для решения этой задачи.

Перед текстом программы кратко опишите алго-

ритм решения. Укажите использованный язык программирования и его версию.

Описание входных и выходных данных

В первой строке входных данных задаётся количество чисел N ($1 \leqslant N \leqslant 1000$).

В каждой из последующих N строк записано одно натуральное число, не превышающее 1000.

Решение

Неэффективная программа (*2 балла*)

Неэффективный алгоритм аналогичен рассмотренным в предыдущих задачах:

- все поступающие на вход числа записываются в линейный массив,
- выполняется полный перебор всех неповторяющихся пар (два вложенных цикла),
- для каждой пары вычисляется сумма элементов, определяется её кратность заданному значению (8) и при удовлетворении этого условия счётчик увеличивается на 1.

```
var  N: integer;                      // количество
                                      // чисел
     a: array[1..1000] of integer; // хранение
                                      // чисел
     s: integer;                      // счетчик
     i, j: integer;
begin
   readln(N);
   for i := 1 to N do readln(a[i]);
   s :=0;
   for i := 1 to N - 1 do
     for j := i + 1 to N do begin
       if (a[i] + a[j]) mod 8 = 0 then s := s + 1
   end;
   writeln(s);
end.
```

Эффективная программа (*4 балла*)

В каких случаях сумма двух целых чисел кратна 8?

Если сумма остатков от деления двух чисел делится на какое-то число, то сумма самих этих чисел также делится на это число.

Доказательство

Пусть даны два числа a и b, каждое из которых по отдельности не делится нацело на число x.

Представим каждое из этих чисел как деление с остатком:

$a = nx + c, \quad b = mx + d,$

где n и m — коэффициенты кратности, а c и d — остатки.

Тогда сумма чисел a и b равна:

$a + b = (nx + c) + (mx + d) = (nx + mx) + (c + d) = x(m + n) + (c + d)$.

По условию, сумма остатков $(c + d)$ делится без остатка на число x. Второе слагаемое — $x(m + n)$ — очевидно, также делится на x без остатка. Следовательно, оба слагаемых в полученной сумме делятся без остатка на число x, тогда и вся эта сумма делится на число x.

Пользуясь вышеуказанным правилом, определим для каждого числа остаток от его деления на число 8 и будем запоминать только количества чисел, имеющих соответствующие остатки: 0, 1, 2, 3, 4, 5, 6 или 7. Для этого организуем массив `ost[0..7]`.

1) Любое число, которое делится на 8 без остатка, можно рассматривать в паре с любым другим числом, которое также делится на 8 без остатка. Количество таких чисел (с остатком, равным 0) хранится в ячейке `ost[0]`. Количество получаемых таких пар равно $\frac{ost[0] \times (ost[0]-1)}{2}$, так как нужно исключить рассмотрение каждого такого числа «в паре самого с собой», а также учесть, что при вычислении произведения ost[0]×(ost[0]−1) одни и те же пары будут подсчитаны дважды.

2) Любое число с ненулевым остатком при делении на 8 может образовать нужную нам пару с числом, остаток которого при делении на 8 дополняет первый остаток до 8. В нашем случае: число с остатком 1 и число с остатком 7, число с остатком 2 и число с остатком 6, число с остатком 3 и с остатком 5, а также числа с остатками 4.

Количества соответствующих пар вычисляются как произведения количеств соответствующих чисел: ost[0]×ost[7], ost[2]×ost[6], ost[3]×ost[5], ost[4]×ost[4], — однако в последнем случае (с остатками 4) опять же одни и те же пары будут посчитаны дважды, а кроме того, нужно исключить рассмотрение каждого такого числа «в паре с самим собой», поэтому правильной будет формула $\frac{ost[4]\times(ost[4]-1)}{2}$.

 Вообще указанное действие — уменьшение одного из сомножителей на 1 и деление получаемого произведения на 2 — требуется производить при вычислении количества пар, если сомножители одинаковы.

3) Общее количество получаемых пар, сумма чисел в которых делится нацело на заданное число 8, будет равна сумме получаемых произведений:

$$\frac{ost[0]\times(ost[0]-1)}{2} + ost[1]\times ost[7] + ost[2]\times ost[6] + ost[3]\times ost[5] + \frac{ost[4]\times(ost[4]-1)}{2}.$$

Таким образом, эффективный алгоритм имеет следующий вид:

- объявляем массив-счётчик количеств чисел, которые делятся на заданное число с остатком, равным 0, 1, 2, 3 и т. д. (размер этого массива равен количеству возможных различных остатков, т.е. равен самому числу, на которое предлагается осуществлять деление,

а индексы элементов этого массива удобнее всего нумеровать с нуля);

- также требуется объявить одну «буферную» переменную для временного хранения очередного входного числа, переменную N для количества входных чисел и цикловую переменную;
- в линейном цикле по очереди считываем входные числа, вычисляем остаток от деления этого числа на заданное число, используем вычисленное значение остатка как индекс и увеличиваем соответствующий элемент массива-счётчика на 1;
- вычисляем и выводим искомое количество пар.

```
var N: integer;     // количество чисел
    a: integer;     // считанное число
    ost: array[0..7] of integer;  // кол-ва
                          // чисел с остатками
    i: integer;
begin
  for i:=0 to 7 do ost[i]:=0;  // обнуление
                               // счетчиков
  readln(N);
  for i:=1 to N do begin
    readln(a);
    ost[a mod 8]:=ost[a mod 8]+1;
  end;
writeln(ost[0]*(ost[0]-1) div 2+
   ost[1]*ost[7]+ost[2]*ost[6]+ost[3]*ost[5]+
                    ost[4]*(ost[4]-1) div 2);
end.
```

 В данном случае количество возможных остатков от деления невелико, и можно просто записать в операторе вывода развёрнутую сумму произведений. Если же количество возможных остатков слишком большое, то удобно будет объявить отдельную переменную для вычисления количества пар (например, k) и подсчитывать количество пар с различ-

ными значениями остатков при помощи цикла, а затем прибавить к полученному количеству также количества неповторяющихся пар с равными остатками.

Например, для данной задачи соответствующий фрагмент текста программы мог бы быть таким:

```
for i:=1 to 3 do
  k:=k+ost[i]*ost[8-i];
  k:=k+ost[0]*(ost[0]—1) div 2+ost[4]*(ost[4]—1) div 2;
```

Кроме того, для повышения эффективности программы по времени вместо строки

```
ost[a mod 8]:=ost[a mod 8]+1;
```

можно использовать отдельную операцию инкремента:

```
inc(ost[a mod 8]);
```

которая во многих языках программирования при трансляции в машинный код реализуется более быстрой процессорной операцией инкремента, а не операцией суммы с константой 1.

Раздел 11. **Теория игр**

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:

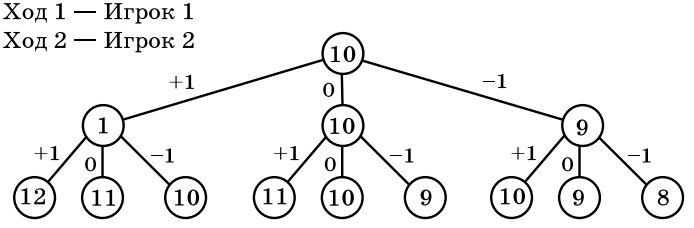
		Игрок 1	
		Стратегия 1	**Стратегия 2**
Игрок 2	**Стратегия 1**	(1, 0)	(–1, –1)
	Стратегия 2	(1, 1)	(0, 1)

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 таблицы

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 договорённость, 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, Петя и Ваня, играют в следующую игру. Перед игроками лежит куча камней. Игроки ходят по очереди, первый ход делает Петя. За один ход игрок может добавить в кучу один или три камня или увеличить количество камней в куче в 2 раза. Например, имея кучу из 15 камней, за один ход можно получить кучу из 16, 18 или 30 камней. У каждого игрока, чтобы делать ходы, есть неограниченное количество камней.

Игра завершается в тот момент, когда количество камней в куче становится не менее 35.

Победителем считается игрок, сделавший последний ход, т.е. первым получивший кучу, в которой будет 35 или больше камней.

В начальный момент в куче было S камней;

$1 \leqslant S \leqslant 34$.

Будем говорить, что игрок имеет выигрышную стратегию, если он может выиграть при любых ходах противника. Описать стратегию игрока — значит описать, какой ход он должен сделать в любой ситуации, которая ему может встретиться при различной игре противника.

Выполните следующие задания. Во всех случаях обосновывайте свой ответ.

Задание 1

а) Укажите все такие значения числа S, при которых Петя может выиграть в один ход. Обоснуйте, что

найдены все нужные значения S, и укажите выигрывающие ходы.

б) Укажите такое значение S, при котором Петя не может выиграть за один ход, но при любом ходе Пети Ваня может выиграть своим первым ходом. Опишите выигрышную стратегию Вани.

Задание 2

Укажите два таких значения S, при которых у Пети есть выигрышная стратегия, причём одновременно выполняются два условия:

Петя не может выиграть за один ход;

Петя может выиграть своим вторым ходом независимо от того, как будет ходить Ваня.

Для каждого указанного значения S опишите выигрышную стратегию Пети.

Задание 3

Укажите значение S, при котором одновременно выполняются два условия:

– у Вани есть выигрышная стратегия, позволяющая ему выиграть первым или вторым ходом при любой игре Пети;

– у Вани нет стратегии, которая позволит ему гарантированно выиграть первым ходом.

Для указанного значения S опишите выигрышную стратегию Вани.

Постройте дерево всех партий, возможных при этой выигрышной стратегии Вани (в виде рисунка или таблицы). На рисунке на рёбрах дерева указывайте, кто делает ход; в узлах — количество камней в позиции.

Решение

Подобную задачу можно решить, как и предыдущую, путём построения «дерева игры». Однако существует другой способ решения, который можно назвать «алгебраическим», — более быстрый и менее громоздкий.

1*а*. Запишем условия выполнения игрового хода в виде системы неравенств. При этом каждое неравенство соответствует одному из возможных вариантов хода плюс ещё одно неравенство определяет ограничение на

возможное количество камней в куче. При этом S в левой части неравенств обозначает количество камней в куче перед первым ходом, а правая часть неравенств указывает условие выигрыша (если в куче станет 35 камней или больше):

- игрок может добавить в кучу один камень: $S + 1 \geqslant 35$;
- игрок может добавить в кучу три камня: $S + 3 \geqslant 35$;
- игрок может увеличить количество камней в два раза: $S * 2 \geqslant 35$;
- в куче изначально может быть не более 34 камней: $S \leqslant 34$.

Решаем полученную систему неравенств, причём учитываем, что поскольку Петя может выбирать любой из трёх возможных вариантов хода, первые три неравенства для хода Пети объединены логической связкой ИЛИ, а последнее, четвёртое неравенство добавляется как обязательное условие (логическая связка И). Нужно также не забывать, что количества камней могут быть только целыми значениями, поэтому получаемые дробные значения нужно округлять в соответствии со знаком неравенства:

$$\begin{cases} \left[\begin{array}{l} S + 1 \geqslant 35; \\ S + 3 \geqslant 35; \\ S * 2 \geqslant 35; \end{array}\right. \\ S \leqslant 34; \end{cases} \Rightarrow \begin{cases} \left[\begin{array}{l} S \geqslant 34; \\ S \geqslant 32; \\ S \geqslant 17{,}5; \end{array}\right. \\ S \leqslant 34; \end{cases} \Rightarrow \begin{cases} \left[\begin{array}{l} S \geqslant 34; \\ S \geqslant 32; \\ S \geqslant 18; \end{array}\right. \\ S \leqslant 34. \end{cases}$$

Далее объединяем первые три неравенства. Так как они связаны логической связкой ИЛИ, достаточно, чтобы было истинно хотя бы одно из этих условий:

$$\begin{cases} S \geqslant 18; \\ S \leqslant 34. \end{cases}$$

Решение этой системы неравенств записываем в виде интервала возможных значений S:

$S \in$ [18 ... 34].

Это и есть ответ на вопрос задания под номером 1*а*: Петя может выиграть в один ход, если изначально в куче имелось от 18 до 34 камней.

Обоснование ответа: если в куче имеется минимум 18 камней, то Петя своим первым ходом может удвоить количество камней, получить в куче не менее 36 камней и выиграть первым ходом.

1*б*. Продолжаем решение задачи с помощью системы, но уже не неравенств, а логических условий принадлежности допустимому диапазону значений. Для хода Вани эти условия объединены логической связкой И, так как Ваня должен иметь выигрышный ход при каждом из выбранных Петей вариантов первого хода (И при первом, И при втором, И при третьем). Так же, как и раньше, записывается одно условие для каждого возможного варианта хода. Решение таких условий производится аналогично решению неравенств, только в правой части нужно вычитать соответствующие значения или делить на 2 обе границы интервала, а при получении дробных значений границы — «стягивать» интервал (округлять дробные значения границ до целых внутрь интервала):

$$\begin{cases} S + 1 \in [18 \ldots 34]; \\ S + 3 \in [18 \ldots 34]; \\ S * 2 \in [18 \ldots 34]; \end{cases} \Rightarrow \begin{cases} S \in [17 \ldots 33]; \\ S \in [15 \ldots 31]; \\ S \in [9 \ldots 17]. \end{cases}$$

Решением такой системы условий является интервал, в который входят только значения S, имеющиеся во всех трёх интервалах. В данном случае три указанных интервала возможных значений S пересекаются только в одном значении:

$S \in [17]$.

Это ответ на вопрос задания под номером 1*б*: Петя не может выиграть за один ход, но при любом ходе Пети Ваня может выиграть своим первым ходом, если изначально в куче имелось 17 камней.

Обоснование ответа: если в куче имелось 17 камней, то после первого хода Пети в куче может стать 18, 20 или 34 камня. В *любом* из этих случаев Ваня своим первым ходом выигрывает, удвоив количество камней в куче.

2. Продолжаем аналогичным способом запись системы условий. Для хода Пети они объединены условием ИЛИ, так как после первого ответного хода Вани Петя может выбрать любой возможный вариант хода:

$$\left[\begin{array}{l} S+1 \in [17]; \\ S+3 \in [17]; \\ S*2 \in [17]; \end{array}\right. \Rightarrow \left[\begin{array}{l} S \in [16]; \\ S \in [14]; \\ S \in [17/2]; \end{array}\right. \Rightarrow \left[\begin{array}{l} S \in [16]; \\ S \in [14]; \\ S \in \varnothing. \end{array}\right.$$

Так как в последнем условии единственное предполагаемое значение S оказывается не целым, решением этого условия является пустое множество.

Решением этой системы условий является объединение интервалов: $S \in [14, 16]$.

Это ответ на второй вопрос задания: у Пети есть выигрышная стратегия, причём одновременно выполняются два условия: Петя не может выиграть за один ход, но Петя может выиграть своим вторым ходом независимо от того, как будет ходить Ваня, если изначально в куче было 14 или 16 камней.

Обоснование ответа: в этих случаях Петя не может выиграть первым ходом, так как даже удвоение количества камней в куче даст только 28 или 32 камня. Но Петя может получить кучу из 17 камней, прибавив к 14 камням 3 камня либо прибавив к 16 камням 1 камень. Если теперь в куче 17 камней, то Ваня своим ответным первым ходом даже путём удвоения количества камней не может выиграть (при любом ходе Вани в куче получится не более 34 камней). Но как бы Ваня ни ходил (даже если он увеличит количество камней в куче всего на 1 камень и получит в ней 18 камней), Ваня создаст тем самым для Пети выигрышную ситуацию — Петя выиграет вторым ходом, удвоив количество камней в куче.

3. Составляем систему условий для второго хода Вани.

Прежде всего рассматриваем отдельно случай, когда Ваня гарантированно выигрывает вторым ходом (первые три условия объединены логической связкой И, так как выигрыш Вани должен обеспечиваться при каждом предыдущем ходе Пети).

$$\begin{cases} S+1 \in [14, 16]; \\ S+3 \in [14, 16]; \\ S*2 \in [14, 16]; \end{cases}$$ — случай гарантированного выигрыша Вани вторым ходом;

$$\begin{cases} S+1 \in [14, 16]; \\ S+3 \in [14, 16]; \\ S*2 \in [14, 16]; \end{cases} \Rightarrow \begin{cases} S \in [13, 15]; \\ S \in [11, 13]; \\ S \in [7, 8]; \end{cases} \Rightarrow S \in \varnothing.$$

Следовательно, у Вани нет возможности гарантированно выиграть вторым ходом.

Теперь рассматриваем возможные условия негарантированного выигрыша Вани первым или вторым ходом. Отдельно рассматриваются выигрышные условия на первом ИЛИ на втором ходе для первого, второго и третьего возможных вариантов хода (поэтому соответствующие условия в каждом случае объединяются логической связкой ИЛИ). После этого полученные объединения условий выигрыша соединяются в систему логической связкой И, поскольку Ваня должен иметь возможность выигрыша в каждом из показанных трёх случаев. Дополнительно учитываем, что S не может быть равно значению (значениям), полученному в ответ на вопрос 1*б*, иначе Ваня гарантированно выиграл бы уже своим первым ходом (у нас же в третьем вопросе задания этот случай исключается). В результате вся система условий выигрыша Вани будет иметь вид:

$$\begin{cases} \begin{cases} S+1 \in [18 \ldots 34]; \\ S+1 \in [14, 16]; \end{cases} \\ \begin{cases} S+3 \in [18 \ldots 34]; \\ S+3 \in [14, 16]; \end{cases} \text{ — выигрыш Вани первым или вторым ходом при третьем возможном варианте хода;} \\ \begin{cases} S+2 \in [18 \ldots 34]; \\ S+2 \in [14, 16]; \end{cases} \text{ — выигрыш Вани первым или вторым ходом при втором возможном варианте хода;} \\ S \notin [17]. \text{ — нет гарантированного выигрыша Вани первым ходом.} \end{cases}$$

Решение такой сложной системы условий нужно выполнять по шагам:

$$\begin{cases} S+1 \in [18 \ldots 34]; \\ S+1 \in [14, 16]; \end{cases} \Rightarrow \begin{cases} S \in [17 \ldots 33]; \\ S \in [13, 15]; \end{cases} \Rightarrow S \in [13, 15; 17 \ldots 33];$$

$$\begin{cases} S+3 \in [18 \ldots 34]; \\ S+3 \in [14, 16]; \end{cases} \Rightarrow \begin{cases} S \in [15 \ldots 31]; \\ S \in [11, 13]; \end{cases} \Rightarrow S \in [11, 13; 15 \ldots 31];$$

$$\begin{cases} S * 2\ [18 \ldots 34]; \\ S * 2 \in [14, 16]; \end{cases} \Rightarrow \begin{cases} S \in [9 \ldots 17]; \\ S \in [7, 8]; \end{cases} \Rightarrow S \in [7, 8; 9 \ldots 17].$$

Тогда:

$$\begin{cases} S \in [13, 15, 17 \ldots 33]; \\ S \in [11, 13, 15 \ldots 31]; \\ S \in [7, 8, 9 \ldots 17]; \\ S \notin [17]; \end{cases} \Rightarrow S \in [13, 15].$$

Нужно внимательно отследить пересечение всех указанных интервалов: в результирующий интервал войдут только те значения, которые входят во все три первых интервала, за исключением значения, указанного в четвёртом интервале (17).

Это ответ на вопрос 3 задания: у Вани есть выигрышная стратегия, позволяющая ему выиграть первым или вторым ходом при любой игре Пети, но у Вани нет стратегии, которая позволит ему гарантированно выиграть первым ходом, если изначально в куче было 13 или 15 камней.

Обоснование ответа.

Например, для $S = 13$ после первого хода Пети в куче будет 14, 16 или 26 камней.

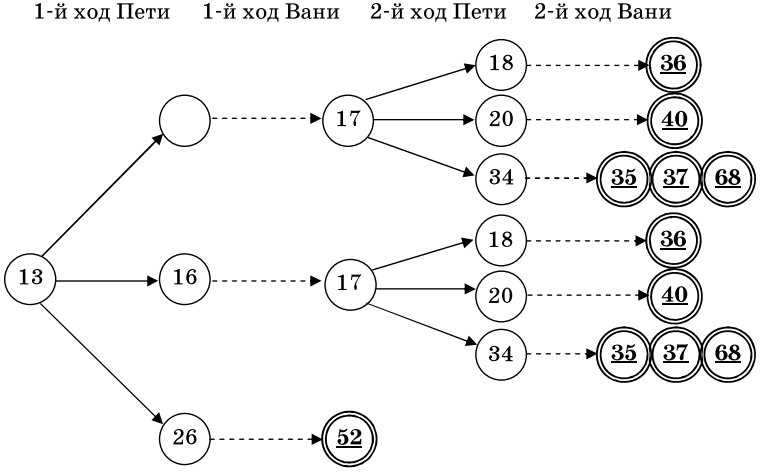
Если после первого хода Пети камней окажется 26, то Ваня своим первым ходом удваивает количество камней в куче и выигрывает.

Если после первого хода Пети камней окажется 14 или 16, то Ваня своим ответным первым ходом прибавляет один камень (для случая 16 камней) или прибавляет 3 камня (для случая 14 камней), чтобы получить в куче 17 камней. Тогда Петя вторым ходом даже при удвоении камней в куче не сможет получить выигрыш. Но при любом втором ходе Пети (даже при прибавлении всего одного камня) камней в куче станет достаточно, чтобы Ваня своим вторым ходом выиграл, удвоив количество камней в куче.

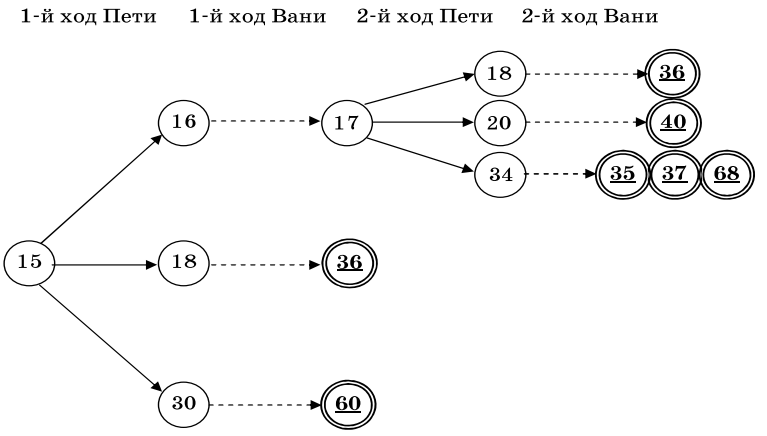
Аналогично обосновывается ответ $S = 15$. После первого хода Пети в куче будет 16, 18 или 30 камней. Если камней станет 18 или 30, то Ваня уже первым ходом достигнет выигрыша, удвоив количество камней. Если же камней в куче станет 16, то Ваня первым ходом прибавляет 1 камень, получая в куче 17 камней. Далее развитие игровой ситуации такое же, как описанное в предыдущем случае для $S = 13$.

«Дерево ходов» игры для показанных двух ответов (в ЕГЭ достаточно вычертить «дерево ходов» для одного любого ответа):

$S = 13$



$S = 15$



Задача 2. Два игрока, Петя и Ваня, играют в игру. Имеются две кучи камней. Игроки ходят по очереди, первый ход делает Петя. За один ход можно добавить в одну любую кучу **один** камень или **удвоить** количество камней в любой куче. Для этого у каждого игрока есть неограниченно много камней.

Игра завершается, когда суммарное количество камней в обеих кучах становится не менее 94. При этом победителем считается тот, кто сделал последний ход, т.е. первым получил выигрышную позицию.

Игрок имеет выигрышную стратегию, если он может выиграть при любых ходах противника. Описание стратегии игрока — это описание ходов в любой ситуации, которая может встретиться при различной игре противника.

Задание 1

Для каждой из начальных позиций (9, 42) и (11, 41) укажите, кто из игроков имеет выигрышную стратегию, опишите эту стратегию в каждом случае, объясните, почему эта стратегия ведёт к выигрышу, и укажите, какое наибольшее количество ходов может потребоваться победителю для выигрыша.

Задание 2

Для каждой из начальных позиций (9, 41), (10, 41), (11, 40) укажите, кто из игроков имеет выигрышную стратегию, опишите эту стратегию в каждом случае, объясните, почему эта стратегия ведёт к выигрышу, и укажите, какое наибольшее количество ходов может потребоваться победителю для выигрыша.

Задание 3

Для начальной позиции (10, 40) укажите, кто из игроков имеет выигрышную стратегию, опишите эту стратегию в каждом случае, объясните, почему эта стратегия ведёт к выигрышу, и укажите, какое наибольшее количество ходов может потребоваться победителю для выигрыша. Постройте дерево ходов, возможных при указанной выигрышной стратегии (в виде рисунка или таблицы).

В отличие от предыдущей задачи, где требовалось указать исходное состояние игры, здесь достаточно построить для каждой из представленных начальных позиций «дерево ходов» и проследить в нём выигрышные стратегии на требуемом количестве ходов.

При этом нужно учитывать следующее:

— игроку, который делает ход первым, для выигрыша достаточно получить **хотя бы одну** выигрышную ситуацию;

— игроку, делающему ход вторым и т.д., для выигрыша требуется получить выигрышную ситуацию **при любом** предыдущем ходе первого игрока.

Решение

Задание 1

1. Строим дерево ходов для начальной позиции (9, 42). В квадратных скобках указано суммарное количество камней в обеих кучах.

Начальная позиция	1-й ход Пети	1-й ход Вани
(9, 42) [51]	(10, 42) [52]	(11, 42) [53]
		(20, 42) [62]
		(10, 43) [53]
		(10, 84) [94]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани
	(18, 42) [60]	(19, 42) [61]
		(36, 42) [78]
		(18, 43) [61]
		(18, 84) [102]
	(9, 43) [52]	(10, 43) [53]
		(18, 43) [61]
		(9, 44) [53]
		(9, 86) [95]
	(9, 84) [93]	**(10, 84) [94]**
		(18, 84) [102]
		(9, 85) [94]
		(9, 168) [177]

Таким образом, у Пети (первого игрока) не имеется стратегии, позволяющей достигнуть выигрыша первым ходом. Однако выигрышная стратегия имеется у Вани (второго игрока), который может гарантированно получить своим ответным **первым** ходом выигрышную ситуацию при **любом** первом ходе Пети.

2. Строим дерево ходов для начальной позиции (11, 41). В квадратных скобках указано суммарное количество камней в обеих кучах.

Начальная позиция	1-й ход Пети	1-й ход Вани
(11, 41) [52]	(12, 41) [53]	(13, 41) [54]
		(24, 41) [65]
		(12, 42) [54]
		(12, 82) [94]
	(22, 41) [63]	(23, 41) [64]
		(44, 41) [85]
		(22, 42) [64]
		(22, 82) [104]
	(11, 42) [53]	(12, 42) [54]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани
		(22, 42) [64]
		(11, 43) [54]
		(11, 84) [95]
	(11, 82) [93]	**(12, 82) [94]**
		(22, 82) [104]
		(11, 83) [94]
		(11, 164) [175]

В этом случае также у Пети (первого игрока) не имеется стратегии, позволяющей достигнуть выигрыша первым ходом. Однако выигрышная стратегия имеется у Вани (второго игрока), который может гарантированно получить своим ответным **первым** ходом выигрышную ситуацию при **любом** первом ходе Пети.

Задание 2

Строим дерево ходов для начальной позиции (9, 41). В квадратных скобках указано суммарное количество камней в обеих кучах.

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети
(9, 41) [50]	(10, 41) [51]	(11, 41) [52]	(12, 41) [53]
			(22, 41) [63]
			(11, 42) [53]
			(11, 82) [93]
		(20, 41) [61]	(21, 41) [62]
			(40, 41) [81]
			(20, 42) [62]
			(20, 82) [102]
		(10, 42) [52]	(11, 42) [53]
			(20, 42) [62]
			(10, 43) [53]
			(10, 84) [94]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети
		(10, 82) [92]	(11, 82) [93]
			(20, 82) [102]
			(10, 83) [93]
			(10, 164) [174]
	(18, 41) [59]	(19, 41) [60]	
		(36, 41) [77]	
		(18, 42) [60]	
		(18, 82) [100]	
	(9, 42) [51]	(10, 42) [52]	(11, 42) [53]
			(20, 42) [62]
			(10, 43) [53]
			(10, 84) [94]
		(18, 42) [60]	(19, 42) [61]
			(36, 42) [78]
			(18, 43) [61]
			(18, 84) [102]
		(9, 43) [52]	(10, 43) [53]
			(18, 43) [61]
			(9, 44) [53]
			(9, 86) [95]
		(9, 84) [93]	**(10, 84) [94]**
			(18, 84) [102]
			(9, 85) [94]
			(9, 168) [177]
	(9, 82) [91]	(10, 82) [92]	
		(18, 82) [100]	
		(9, 83) [92]	
		(9, 164) [173]	

При указанной начальной ситуации Петя (первый игрок) может выбрать такой первый ход (прибавить 1 камень ко второй куче, получив ситуацию (9, 42) —

в сумме 51 камень), что Ваня (второй игрок) не имеет на своём ответном первом ходе ни одной выигрышной ситуации, но своим **вторым** ходом Петя может получить выигрыш при ***любом*** ответном первом ходе Вани.

 Самостоятельно выполните анализ выигрышных стратегий при двух других начальных позициях: (10, 41) и (11, 40).

Задание 3

1. Строим дерево ходов для начальной позиции (10, 40). В квадратных скобках указано суммарное количество камней в обеих кучах.

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
(10, 40) [50]	(11, 40) [51]	(12, 40) [52]	(13, 40) [53]	(14, 40) [54]
				(26, 40) [66]
				(13, 41) [54]
				(13, 80) [93]
			(24, 40) [64]	(25, 40) [65]
				(48, 40) [88]
				(24, 41) [65]
				(24, 80) [104]
			(12, 41) [53]	(13, 41) [54]
				(24, 41) [65]
				(12, 42) [54]
				(12, 82) [94]
			(12, 80) [92]	(13, 80) [93]
				(24, 80) [104]
				(12, 81) [93]
				(12, 160) [172]
		(22, 40) [62]	(23, 40) [63]	(24, 40) [64]
				(46, 40) [86]
				(23, 41) [64]
				(23, 80) [103]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
			(44, 40) [84]	(45, 40) [85]
				(88, 40) [128]
				(44, 41) [85]
				(44, 80) [124]
			(22, 41) [63]	(23, 41) [64]
				(44, 41) [85]
				(22, 42) [64]
				(22, 82) [104]
			(22, 80) [102]	
		(11, 41) [52]	(12, 41) [53]	(13, 41) [54]
				(24, 41) [65]
				(12, 42) [54]
				(12, 82) [94]
			(22, 41) [63]	(23, 41) [64]
				(44, 41) [85]
				(22, 42) [64]
				(22, 82) [104]
			(11, 42) [53]	(12, 42) [54]
				(22, 42) [64]
				(11, 43) [54]
				(11, 84) [95]
			(11, 82) [93]	**(12, 82) [94]**
				(22, 82) [104]
				(11, 83) [94]
				(11, 164) [175]
		(11, 80) [91]	(12, 80) [92]	(13, 80) [93]
				(24, 80) [104]
				(12, 81) [93]
				(12, 160) [172]
			(22, 80) [102]	
			(11, 81) [92]	(12, 81) [93]
				(22, 81) [103]
				(11, 82) [93]

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
				(11, 162) [173]
			(11, 160) [171]	
	(20, 40) [60]	(21, 40) [61]	(22, 40) [62]	(23, 40) [63]
				(44, 40) [84]
				(22, 41) [63]
				(22, 80) [102]
			(42, 40) [82]	(43, 40) [83]
				(84, 40) [124]
				(42, 41) [83]
				(42, 80) [122]
			(21, 41) [62]	(22, 41) [63]
				(42, 41) [83]
				(21, 42) [63]
				(21, 82) [103]
			(21, 80) [101]	
		(40, 40) [80]	(41, 40) [81]	(42, 40) [82]
				(82, 40) [122]
				(41, 41) [82]
				(41, 80) [121]
			(80, 40) [120]	
			(40, 41) [81]	(41, 41) [82]
				(80, 41) [121]
				(40, 42) [82]
				(40, 82) [122]
			(40, 80) [120]	
		(20, 41) [61]	(21, 41) [62]	(22, 41) [63]
				(42, 41) [83]
				(21, 42) [63]
				(21, 82) [103]
			(40, 41) [81]	(41, 41) [82]
				(80, 41) [121]
				(40, 42) [82]
				(40, 82) [122]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
			(20, 42) [62]	(21, 42) [63]
				(40, 42) [82]
				(20, 43) [63]
				(20, 84) [104]
			(20, 82) [102]	
		(20, 80) [100]		
	(10, 41) [51]	(11, 41) [52]	(12, 41) [53]	(13, 41) [54]
				(24, 41) [65]
				(12, 42) [54]
				(12, 82) [94]
			(22, 41) [63]	(23, 41) [64]
				(44, 41) [85]
				(22, 42) [64]
				(22, 82) [103]
			(11, 42) [53]	(12, 42) [54]
				(22, 42) [64]
				(11, 43) [54]
				(11, 84) [95]
			(11, 82) [93]	**(12, 82) [94]**
				(22, 82) [104]
				(11, 83) [94]
				(11, 164) [175]
		(20, 41) [61]	(21, 41) [62]	(22, 41) [63]
				(42, 41) [83]
				(21, 42) [63]
				(21, 82) [103]
			(40, 41) [81]	(41, 41) [82]
				(80, 41) [121]
				(40, 42) [82]
				(40, 82) [122]
			(20, 42) [62]	(21, 42) [63]
				(40, 42) [82]
				(20, 43) [63]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
				(20, 84) [104]
			(20, 82) [102]	
		(10, 42) [52]	(11, 42) [53]	(12, 42) [54]
				(22, 42) [64]
				(11, 43) [54]
				(11, 84) [95]
			(20, 42) [62]	(21, 42) [63]
				(40, 42) [82]
				(20, 43) [63]
				(20, 84) [104]
			(10, 43) [53]	(11, 43) [54]
				(20, 43) [63]
				(10, 44) [54]
				(10, 86) [96]
			(10, 84) [94]	
		(10, 82) [92]	(11, 82) [93]	**(12, 82) [94]**
				(22, 82) [104]
				(11, 83) [94]
				(11, 164) [175]
			(20, 82) [102]	
			(10, 83) [93]	**(11, 83) [94]**
				(20, 83) [103]
				(10, 84) [94]
				(10, 166) [176]
			(10, 164) [174]	
	(10, 80) [90]	(11, 80) [91]	(12, 80) [92]	(13, 80) [93]
				(24, 80) [104]
				(12, 81) [93]
				(12, 160) [172]
			(22, 80) [102]	
			(11, 81) [92]	(12, 81) [93]
				(22, 81) [103]
				(11, 82) [93]

Продолжение таблицы

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
				(11, 162) [173]
			(11, 160) [171]	
		(20, 80) [100]		
		(10, 81) [91]	(11, 81) [92]	(12, 81) [93]
				(22, 81) [103]
				(11, 82) [93]
				(11, 162) [173]
			(20, 81) [101]	
			(10, 82) [92]	(11, 82) [93]
				(20, 82) [102]
				(10, 83) [93]
				(10, 164) [174]
			(10, 162) [172]	
		(10, 160) [170]		

При указанной начальной ситуации:

— Петя (первый игрок) не может достигнуть выигрыша своим первым ходом;

— Ваня (второй игрок) может получить выигрыш на своём первом ответном ходе, только если первый ход Пети создаёт позицию (20, 40) [60] или (10, 80) [90] (закрасим серым фоном соответствующие ячейки таблицы ходов); остальные варианты первого хода Пети оставляют Ваню без выигрыша на его первом ответном ходе;

— своим вторым ходом Петя может получить выигрыш только при следующих вариантах ответного первого хода Вани (до стрелки указана позиция перед ходом Вани): (11, 40) [51] → (22, 40) [62], (11, 40) [51] → (11, 80) [91], (10, 41) [51] → (20, 41) [61], (10, 41) [51] → (10, 42) [52], (10, 41) [51] → (10, 82) [92] (закрасим соответствующие ячейки серым фоном); в остальных случаях ответного первого хода Вани Петя не может выиграть своим вторым ходом;

— наконец, анализируя оставшуюся незакрашенной часть таблицы, можно сделать вывод, что при указан-

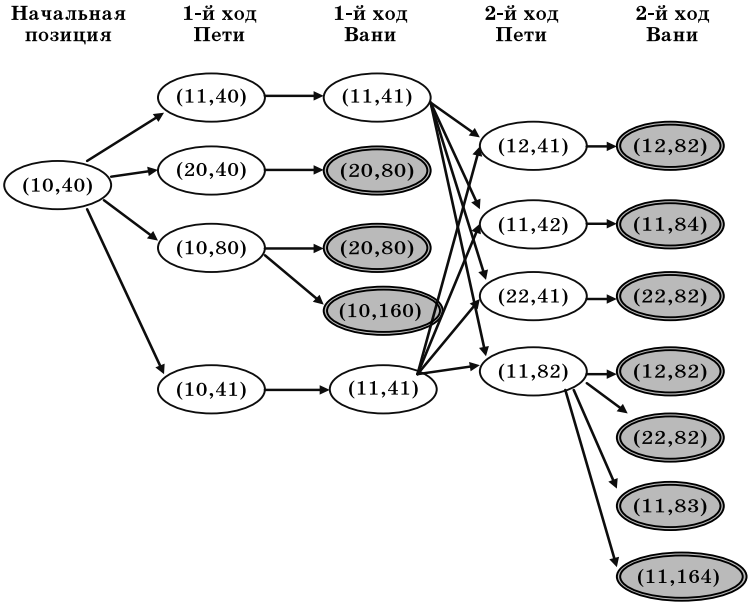
ной начальной позиции (10, 40) [50] и при грамотной игре выигрывает Ваня:

— если первый ход Пети (20, 40) [60] или (10, 80) [90], то Ваня выигрывает ответным первым ходом (см. таблицу выше),

— если первый ход Пети (11, 40) [51] или (10, 41) [51], то Ваня ответным первым ходом может создать такие игровые ситуации ((11, 40) [51] → (11, 41) [52], либо (10, 41) [51] → (11, 41) [52]), что при **любом** втором ходе Пети у Вани имеется вариант ответного второго хода, приводящий к выигрышу.

Таким образом, при указанных начальных условиях выигрывает Ваня своим первым либо вторым ответным ходом.

Дерево ходов, соответствующих выигрышной стратегии Вани, в графическом виде может выглядеть так:



Дерево ходов, соответствующих выигрышной стратегии Вани, также может быть представлено как фрагмент общей таблицы ходов:

Начальная позиция	1-й ход Пети	1-й ход Вани	2-й ход Пети	2-й ход Вани
(10, 40) [50]	(11, 40) [51]	(11, 41) [52]	(12, 41) [53]	**(12, 82) [94]**
			(22, 41) [63]	**(22, 82) [104]**
			(11, 42) [53]	**(11, 84) [95]**
			(11, 82) [93]	**(12, 82) [94]**
				(22, 82) [104]
				(11, 83) [94]
				(11, 164) [175]
	(20, 40) [60]	**(20, 80) [100]**		
	(10, 41) [51]	(11, 41) [52]	(12, 41) [53]	**(12, 82) [94]**
			(22, 41) [63]	**(22, 82) [103]**
			(11, 42) [53]	**(11, 84) [95]**
			(11, 82) [93]	**(12, 82) [94]**
				(22, 82) [104]
				(11, 83) [94]
				(11, 164) [175]
	(10, 80) [90]	**(20, 80) [100]**		
	(10, 80) [90]	**(10, 160) [170]**		

Задача 3. Два игрока, Петя и Ваня играют в игру с цепочками символов. Игра начинается со слова, которое состоит из n букв А и m букв Б. Такое слово обозначается (n, m). Игроки ходят по очереди, первый ход делает Петя. За один ход игрок может

1) добавить в слово одну букву — А или Б;

2) удвоить количество букв А;

3) удвоить количество букв Б.

Игра завершается, когда длина слова составит не менее Z символов. Победителем считается игрок, сделавший последний ход — первым получивший слово длиной Z или больше букв.

Задание 1. Для каждой из двух заданных начальных позиций укажите, кто из игроков имеет выигрышную стратегию.

Задание 2. Для каждой из трёх заданных начальных позиций укажите, кто из игроков имеет выигрышную стратегию.

Задание 3. Для заданной начальной позиции укажите, кто из игроков имеет выигрышную стратегию.

В каждом случае требуется описать выигрышную стратегию: объяснить, почему она ведёт к выигрышу, и указать, какое наибольшее количество ходов может потребоваться победителю при этой стратегии. В задании 3 также требуется построить дерево всех партий, возможных при указанной выигрышной стратегии, и представить это дерево в виде рисунка или таблицы.

Указания к решению:

В этой задаче можно считать добавление к слову букв А аналогом добавления камней в первую кучу, а добавление букв Б — аналогом добавления камней во вторую кучу. В остальном принципы решения здесь те же самые, что в прежних задачах с двумя кучами.

Задача 4. Имеется некий набор слов (осмысленных либо произвольных наборов букв), записанных в фигурных скобках. Два игрока — Петя и Ваня — по очереди составляют слово из букв по следующим правилам:

— первый игрок может записать одну первую букву слова, выбрав любую из возможных первых букв слов, имеющихся в наборе;

— второй игрок должен дописать к уже имеющейся первой букве одну из возможных вторых букв, которые в заданном наборе слов имеются после ранее выбранной первой буквы;

— все последующие ходы делаются аналогично;

— выигрывает тот игрок, который очередным своим ходом получит одно из слов, имеющихся в заданном наборе.

Например, если задан набор {КОТ, КИНО, МАК}, то первый игрок может выбрать в качестве первой буквы

составляемого слова или букву К (она первая в первых двух словах набора), или букву М (первая в третьем слове набора).

Предположим, что первый игрок выбрал букву К. Тогда второй игрок может выбрать любую вторую букву из слов набора, которые начинаются с К, — то есть или букву О (из слова КОТ), или букву И (из слова КИНО).

Пусть второй игрок выбрал букву О, и тем самым получил в итоге «КО». В этом случае первый игрок своим вторым ходом дописывает третьей буквой единственно возможную букву Т, получает полностью слово КОТ и выигрывает.

Задание 1*а*. Даны последовательности букв {ABCDABCDZ} и {DCBADCBA}. У кого из игроков есть выигрышная стратегия? Какая?

Задание 1*б*. Даны последовательности букв {КОТКОТ...КОТКОТ} (33 буквы) и {ОЗОНОЗОН...ОЗОНОЗОН} (88 букв). Укажите последовательность, при которой выигрывает Петя.

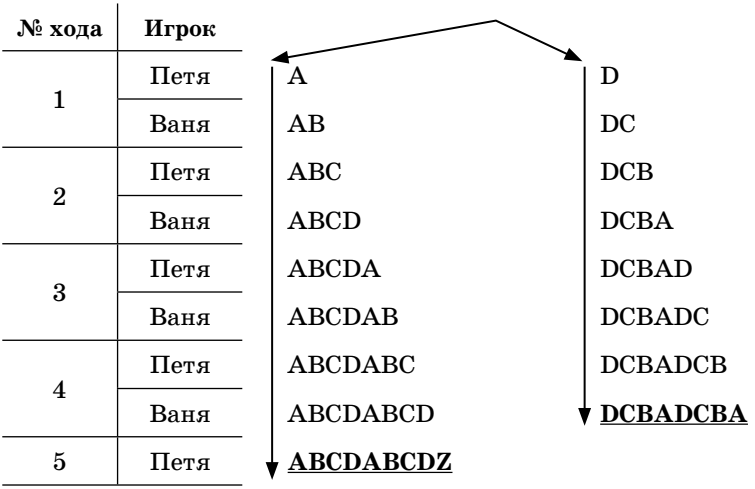
Задание 2. Даны те же последовательности, что и в задании 1*б*. Какие две буквы нужно поменять в последовательности {КОТКОТ...КОТКОТ}, чтобы выигрышная стратегия была у Вани?

Задание 3. Дан набор слов {КОЗА, КОРОВА, ПОРОГ, ПОРТАЛ, ПОРЦИОН}. Кто из игроков имеет выигрышную стратегию для этого набора?

Решение

Задание 1*а*. Очевидно, Петя (первый игрок) имеет возможность выбрать первую букву любого из заданных двух слов — A или D. Далее же, поскольку в этих словах никакие две буквы в «одноимённых» позициях не совпадают, ход игры жёстко предопределён: каждый игрок очередным ходом вынужден записывать очередную букву выбранного слова, пока оно не будет записано целиком.

Составим табличное дерево игры:



№ хода	Игрок		
1	Петя	A	D
	Ваня	AB	DC
2	Петя	ABC	DCB
	Ваня	ABCD	DCBA
3	Петя	ABCDA	DCBAD
	Ваня	ABCDAB	DCBADC
4	Петя	ABCDABC	DCBADCB
	Ваня	ABCDABCD	**DCBADCBA**
5	Петя	**ABCDABCDZ**	

Легко понять, что при таком ходе игры, когда игроки попеременно делают целиком предопределённые ходы, исход игры зависит только от длины заданных слов:

— если длина слова нечётна, то последним делает ход (заканчивая слово и тем самым выигрывая) тот же самый игрок, который делал первый ход;

— если длина слова чётна, то последним делает ход (тем самым выигрывая) игрок, который делал второй ход.

Итак, на слове из нечётного числа букв выигрывает Петя, а на слове из чётного числа букв выигрывает Ваня. А поскольку Петя на своём первом ходе имеет возможность выбора — какое из двух слов начать, именно Петя имеет выигрышную стратегию, заключающуюся в выборе на первом ходе первой буквы слова из нечётного количества букв.

Задание 1*б*. В решении предыдущего задания мы уже выяснили, что для слова из нечётного количества

букв выигрыш однозначно имеет первый игрок, а для слова из чётного количества букв — второй игрок.

Если Петя делает ход первым, то выигрывает он для последовательности, в которой нечётное число букв. Это последовательность {КОТКОТ...КОТКОТ} из 33 букв.

Последовательность же {ОЗОНОЗОН...ОЗОНОЗОН} имеет чётное число букв (88) и потому не даёт выигрыша для Пети.

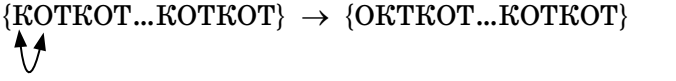
Задание 2. Если рассматривать весь набор слов, приведённый в задании 1*б*, а именно: {КОТКОТ...КОТКОТ} (33 буквы) и {ОЗОНОЗОН...ОЗОНОЗОН} (88 букв), — то, по аналогии с заданием 1*а*, выигрышная стратегия однозначно имеется у Пети.

Можно ли поменять местами две буквы в слове {КОТКОТ...КОТКОТ}, чтобы полученная новая последовательность обеспечивала выигрышную стратегию уже для Вани?

Очевидно, это возможно только тогда, когда Ваня при наличии уже написанной Петей первой буквы получает возможность выбрать для продолжения слово из чётного числа букв. Или, иными словами, нам нужно, чтобы оба слова начинались с одной и той же буквы.

Тогда Петя будет лишён возможности выбора, а вот у Вани, наоборот, появится искомый выбор.

Поскольку по условию менять буквы можно только в слове {КОТКОТ...КОТКОТ}, нужно в нём поменять местами, например, первую и вторую букву:

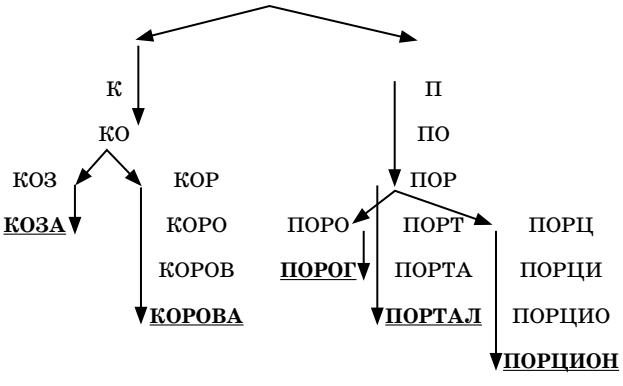


В результате мы получаем набор слов: {ОКТКОТ... КОТКОТ} (33 буквы) и {ОЗОНОЗОН...ОЗОНОЗОН} (88 букв). Тогда Петя во время своего первого хода будет вынужден записать букву О (которая у обоих слов одна и та же). А вот Ваня на втором ходе может выбрать бук-

ву К (из первого слова) или же букву З (из второго слова). Выбрав именно букву З и тем самым выбрав выигрышное для него слово из чётного числа букв, Ваня обеспечивает себе однозначно выигрышную стратегию.

Задание 3. Сразу строим дерево игры в табличном виде, учитывая, что Петя во время первого хода может выбрать или букву К, или букву П.

№ хода	Игрок
1	Петя
	Ваня
2	Петя
	Ваня
3	Петя
	Ваня
4	Петя



Итак, если Петя первым ходом выберет букву К, то в обоих возможных исходах (независимо от выбора, сделанного Петей на втором ходе) выигрывает Ваня. А если Петя первым ходом выбирает букву П, то Ваня может обеспечить себе выигрыш, выбрав во время своего второго хода букву Т.

Делаем вывод, что для данного набора слов выигрышная стратегия есть у Вани. Заключается она (при, насколько возможно, грамотной игре Пети) в выборе четвёртой буквы Т во время второго хода, чтобы свести игру к слову ПОРТАЛ из чётного числа букв.

Возможен другой вариант задачи, в котором играющим предлагается не дописывать, а, наоборот, стирать в имеющемся слове или одну какую-то букву, или сразу все одинаковые буквы.

При этом даётся довольно большой набор различных слов и спрашивается, для каких из них выигрывает один игрок, а для каких — второй.

Вполне очевидно, что первое действие, которое нужно сделать при решении такой задачи, — это убрать в обрабатываемом слове все повторы (например, «ОГОГО» превращается в «ОГ», а «АРАРАТ» — в «АРТ»). А после этого решение будет таким же, как и в уже рассмотренной задаче: всё зависит от того, чётным или нечётным является количество букв в полученном слове.

Справочное издание

12+

Богомолова Ольга Борисовна

И Н Ф О Р М А Т И К А
Новый полный справочник для подготовки к ЕГЭ

Редакция «Образовательные проекты»
Ответственный редактор *Н.А. Шармай.* Художественный редактор *Т.Н. Войткевич.* Технический редактор *Е.П. Кудиярова*
Компьютерная верстка *И.А. Ковалевой*

ЕДИНЫЙ ГОСУДАРСТВЕННЫЙ ЭКЗАМЕН

В справочнике, адресованном выпускникам и абитуриентам, а также учителям и методистам, в полном объёме представлен материал школьного курса «Информатика», который проверяется на едином государственном экзамене.

Структура книги соответствует кодификатору элементов содержания по предмету, на основе которого составлены экзаменационные задания — контрольные измерительные материалы ЕГЭ.

В справочнике подробно разобраны все необходимые для успешной подготовки к ЕГЭ темы: «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».

Каждый раздел справочника включает краткий конспект теоретического материала и разбор решений типовых заданий ЕГЭ за последние годы.

Автор пособия — Ольга Борисовна Богомолова, практикующий школьный учитель, методист, доктор педагогических наук, автор большого количества книг и статей по вопросам преподавания информатики в школе. Награждена почётным знаком «Заслуженный учитель города Москвы».

ОБРАЗОВАТЕЛЬНЫЕ ПРОЕКТЫ