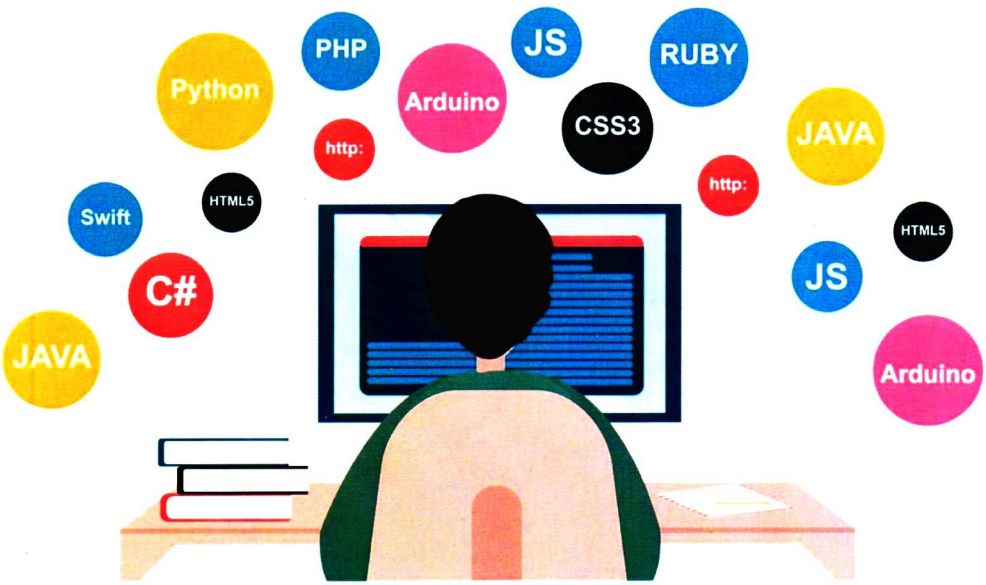
Златопольский Д. М.

Подготовка к ЕГЭ по информатике в компьютерной форме



ДМК ПРЕСС

Златопольский Д. М.

Подготовка к ЕГЭ по информатике в компьютерной форме

Москва, 2021

УДК 373.167.1:004+004(075.3)
ББК 32.97я72
367

Златопольский Д. М.

367 Подготовка к ЕГЭ по информатике в компьютерной форме. – М.: ДМК Пресс, 2021. – 304 с.: ил.

ISBN 978-5-97060-896-8

Книга предназначена для самостоятельной подготовки учащихся к единому государственному экзамену по информатике и ИКТ, который, начиная с 2021 года, будет проходить в компьютерной форме. Согласно демонстрационному варианту ЕГЭ 2021 года, в содержании экзамена будет существенно увеличено количество заданий, связанных с алгоритмизацией и программированием. Таким задачам в книге уделяется особое внимание. Обсуждаются и методики выполнения других заданий, представленных в демонстрационном варианте. В приложениях приведены вспомогательные материалы, связанные с заданиями ЕГЭ. Кроме учащихся старших классов, готовящихся к сдаче экзамена, книгу могут использовать преподаватели информатики, а также студенты и школьники, желающие углубить общие знания по информатике и ИКТ.

УДК 373.167.1:004+004(075.3)
ББК 32.97я72

ISBN 978-5-97060-896-8

Содержание

Предисловие

Книга, которую вы читаете, является первым и на конец 2020 года единственным пособием по подготовке к единому государственному экзамену по информатике и ИКТ, который, начиная с 2021 года, будет проходить в компьютерной форме [1].

Как показывает анализ проекта демонстрационного варианта ЕГЭ 2021 года, в содержании экзамена существенно увеличится количество заданий, связанных с алгоритмизацией и программированием. Так, в нем таких заданий 12 при общем числе заданий – 27. При этом весомость заданий 25, 26 и 27 составляет не 1, а 2 балла (общий максимальный первичный балл за весь экзамен – 30).

Это говорит о том, что от умения решать задачи по алгоритмизации и программированию в значительной степени зависит успешность сдачи ЕГЭ в целом. В то же время, как показывает опыт, такие задачи часто вызывают у школьников заметные трудности. В большой степени это связано с недостаточным числом часов, отводимых на изучение алгоритмизации и программирования в школе.

Данная книга должна восполнить этот недостаток – помочь учащимся подготовиться к экзамену самостоятельно. В ней системно, подробно и доступно описана методика выполнения заданий по программированию и алгоритмизации, которые могут встретиться на экзамене.

Содержание книги основано на материалах, представленных в [1] и [2][1], и информации, полученной автором от коллег из регионов России, в которых проводился эксперимент по проведению ЕГЭ по информатике в компьютерной форме.

Сначала в главе 1 книги обсуждены все частные, вспомогательные задачи, умение решать которые позволит успешно выполнить задания экзамена. Общая методика выполнения заданий, связанных с алгоритмизацией и программированием, а также ряда других заданий, в том числе нового типа, описана в главе 2. В заключительной главе обсуждены методики выполнения соответствующих заданий экзамена, представленных в [1].

---

[1] Пользуясь случаем, автор выражает благодарность Константину Юрьевичу Полякову и его коллегам за опубликованную информацию.

В приложениях приведены другие материалы, связанные с заданиями ЕГЭ.

При разработке программ (частных и из заданий ЕГЭ) использован школьный алгоритмический язык (система программирования КуМир [3]). Русский синтаксис этого языка и большое число комментариев делают приведенные программы максимально понятными и легко переносимыми на любой другой язык. Это означает, что книга может быть использована читателями, владеющими любым языком программирования. В большинстве случаев после разбора методики решения и программы на школьном алгоритмическом языке приводятся также соответствующие программы на популярных среди школьников языках Python и Паскаль.

Кроме учащихся, готовящихся к сдаче экзамена самостоятельно, книгу могут использовать учителя и преподаватели информатики, а также студенты и учащиеся, желающие углубить свои знания по информатике вне связи с ЕГЭ.

Глава 1

Вспомогательные задачи

В данной главе рассмотрена методика решения задач, предусмотренных Кодификатором элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2021 году единого государственного экзамена по информатике и ИКТ [4], а также задач, которые используются в демонстрационных вариантах экзамена 2021 года [1] и нескольких последних лет [5–10].

1.1. Обработка натурального числа

1.1.1. Выделение цифр

Задача формулируется так: «Дано шестизначное[1] натуральное число n. Вывести его цифры в столбик, начиная с последней. Оператор цикла не использовать».

Решение

Первую цифру шестизначного числа можно найти, разделив заданное число на 100 000:

```
цифра1 := div(n, 100000),
```

где `div` – функция школьного алгоритмического языка, возвращающая целую часть частного от деления первого аргумента на второй (в других языках для расчета также используется не функция, а специальная операция).

Последняя цифра равна остатку от деления на 10:

```
цифра6 := mod(n, 10),
```

где `mod` – функция школьного алгоритмического языка, возвращающая остаток от деления своего первого аргумента на второй (в других языках программирования для этого также используется специальная операция).

Остальные цифры одним действием определить нельзя.

Приведем программу, в которой используются однотипные формулы:

```
алг
нач цел n, цифра1, цифра2, цифра3, цифра4, цифра5, цифра6
  ввод n
  цифра6 := mod(n, 10); n := div(n, 10)
  цифра5 := mod(n, 10); n := div(n, 10)
```

[1] Или любой другой заданной «значности».

```
  цифра4 := mod(n, 10); n := div(n, 10)
  цифра3 := mod(n, 10); n := div(n, 10)
  цифра2 := mod(n, 10); n := div(n, 10)
  цифра1 := mod(n, 10)
  вывод нс, цифра6
  вывод нс, цифра5
  ...
  вывод нс, цифра1
кон
```

1.1.2. Выделение цифр. Вариант 2

Задача формулируется так: «Дано натуральное число n. Вывести его цифры в столбик, начиная с последней».

Решение

Когда количество цифр в заданном числе известно, можно выделить любую его цифру. Но в данном случае это количество неизвестно. Поэтому подумаем над вопросом – какую цифру целого числа (см. схему ниже) определить можно?

первая	*вторая*	...	*предпоследняя*	*последняя*

цифры числа

Ответ – последнюю (последняя цифра любого натурального числа равна остатку от деления этого числа на 10 – убедитесь в этом!):

```
посл = mod(n, 10) | посл - последняя цифра числа n
```

А остальные цифры? Как, например, определить предпоследнюю цифру? Ее можно найти только так – получить число без последней цифры исходного числа (разделив исходное нацело на 10) и для него определить последнюю цифру.

Аналогично можно получить и предпредпоследнюю, и остальные цифры.

Следовательно, для решения задачи в программе надо многократно выполнить следующие действия:

1) определить последнюю цифру числа;

2) вывести ее на экран;

3) получить число без последней цифры.

Но сколько раз надо повторить указанные действия? Неизвестно (неизвестно число разрядов в заданном числе n), т. е. оператор цикла с параметром использовать нельзя. Надо применить оператор цикла с предусловием.

Соответствующий фрагмент программы:

```
нц пока n > 0:
    посл := mod(n, 10)
    вывод нс, посл
    n := div(n, 10)
кц
```

Язык Паскаль

```
while n > 0 do
  begin:
    posl := n mod 10;
    writeln(posl);
    n := n div 10
  end;
```

Язык Python

```
while n > 0:
  posl = n % 10
  print(posl)
  n = n//10
```

1.1.3. Определение суммы цифр

Для решения задачи в программе надо многократно выполнить следующие действия:

1) определить последнюю цифру числа;
2) учесть ее в найденной ранее сумме;
3) получить число без последней цифры.

С использованием переменной сум, накапливающей в себе сумму уже учтенных цифр, программа решения задачи оформляется следующим образом:

```
вывод нс, "Введите натуральное число "
ввод n
сум := 0      | Начальное значение суммы цифр
нц пока n > 0
  посл := mod(n, 10)
  сум := сум + посл
  n := div(n, 10)
кц
вывод нс, "Сумма цифр этого числа равна ", сум
```

Язык Паскаль

```
...
sum := 0;
```

Язык Python

```
...
sum = 0
```

```
while n > 0 do
  begin
    posl := n mod 10;
    sum := sum + posl;
    n := n div 10
  end;
...
```

```
while n > 0:
  posl = n % 10
  sum = sum + posl
  n = n//10
...
```

1.1.4. Определение произведения цифр

Задача решается аналогично предыдущей. Отличия:

1) рассчитывается не сумма, а произведение цифр;
2) начальное значение искомой величины произв должно быть принято равным 1.

Программа:

```
вывод нс, "Введите натуральное число "
ввод n
произв := 1      | Начальное значение
нц пока n > 0
  посл := mod(n, 10)
  произв := произв * посл
  n := div(n, 10)
кц
вывод нс, "Произведение цифр этого числа равно ", произв
```

Язык Паскаль

```
...
proizv := 1;
while n > 0 do
  begin
    posl := n mod 10;
    proizv := proizv * posl;
    n := n div 10
  end;
...
```

Язык Python

```
...
proizv = 1
while n > 0:
  posl = n % 10
  proizv = proizv * posl
  n = n//10
...
```

1.1.5. Определение количества цифр числа

Здесь следует использовать переменную-счетчик количества уже «обработанных» цифр:

```
вывод нс, "Введите натуральное число "
ввод n
кол := 0      | Начальное значение количества цифр
```

```
нц пока n > 0
  посл := mod(n, 10)
  кол := кол + 1
  n := div(n, 10)
кц
вывод нс, "Количество цифр этого числа равно ", кол
```

Обратим внимание на то, что последнюю цифру посл можно не определять.

Язык Паскаль

```
...
kol := 0;
while n > 0 do
  begin
    kol := kol + 1;
    n := n div 10
  end;
...
```

Язык Python

```
...
kol = 0
while n > 0:
  kol = kol + 1
  n = n//10
...
```

1.1.6. Выделение цифр шестизначного натурального числа и запись их в массив (в программе на языке Python – в список)

Решение

Здесь после выделения каждой очередной цифры следует записать ее в массив:

```
алг
нач цел n, к, посл, цел таб цифры[1:6]
  ввод n
  нц для к от 1 до 6
    | Определяем последнюю цифру
    посл := mod(n, 10)
    | и записываем ее в массив
    цифры[к] := посл
    | Отбрасываем последнюю цифру числа n
    n := div(n, 10)
  кц
  ...
кон
```

Можно определение цифры и запись ее в массив провести одним оператором:

```
    цифры[k] := mod(n, 10)
```

Язык Паскаль

```
var n: longint; posl, k: byte;
    tsifri: array[1..6] of byte,
BEGIN
readln(n);
kol := 0;
for k := 1 to 6 do
  begin
    posl := n mod 10;
    tsifri[k] := posl;
    n := n div 10
  end;
...
```

Язык Python

```
# Инициация и начальное
# заполнение списка с цифрами
tsifri = [0, 0, 0, 0, 0, 0]
# Можно использовать генератор
списков
# tsifri = [0 for k in range(6)]
n = int(input())
for k in range(6):
  posl = n % 10
  tsifri[k] = posl
  n = n//10
...
```

В программе на языке Python можно также использовать метод append():

```
# Инициация списка
tsifri = [] # Пустой список
n = int(input())
for k in range(6):
  # Определяем последнюю цифру
  posl = n % 10
  # Добавляем ее в конец списка
  tsifri.append(posl)
  # Отбрасываем последнюю цифру числа n
  n = n//10
```

Нетрудно убедиться, что цифры числа записываются в массив в обратном порядке – последняя цифра записана в первом элементе массива[1], предпоследняя – во втором и т. д.

В программе на языке Python можно также после заполнения списка в обратном порядке перевернуть элементы с помощью метода reverse():

```
...
tsifri.reverse()
```

[1] В программах на языке Python, в котором нумерация элементов списка начинается с нуля, последняя цифра будет записана в элементе с индексом 0.

1.1.7. Определение максимальной цифры числа

Алгоритм решения этой задачи аналогичен алгоритму действий человека, который определяет максимальную цифру в некотором числе:

324086312

– сначала он запоминает первую цифру, а затем рассматривает вторую. Если она больше того числа, которое помнил, то он запоминает вторую цифру и переходит к следующей, третьей, цифре, в противном случае просто переходит к следующей цифре и делает то же самое. В нашей программе единственное отличие в том, что просматривать (выделять и сравнивать) цифры мы будем, начиная с последней.

В приведенной далее программе искомое максимальное значение хранит переменная макс.

```
вывод нс, "Введите натуральное число "
ввод n
посл := mod(n, 10) | Выделяем последнюю цифру
макс := посл       | Принимаем ее в качестве максимальной
n := div(n, 10)    | Отбрасываем последнюю цифру
| Рассматриваем остальные цифры
нц пока n > 0
  посл := mod(n, 10)     | Выделяем
  если посл > макс       | Сравниваем
    то                   | и при необходимости
      макс := посл       | меняем значение макс
  все
  n := div(n, 10)
кц
вывод нс, "Максимальная цифра заданного числа равна ", макс
```

В данном случае, так как диапазон возможных значений цифр известен, отдельно последнюю цифру можно не выделять, а в качестве начального значения переменной макс принять 0 (см. п. 1.4.1):

```
вывод нс, "Введите натуральное число "
ввод n
макс := 0
| Рассматриваем все цифры
нц пока n > 0
  посл := mod(n, 10)
```

```
  если посл > макс
    то
      макс := посл
  все
  n := div(n, 10)
кц
вывод нс, "Максимальная цифра заданного числа равна ", макс
```

Язык Паскаль

```
...
max := 0;
while n > 0 do
  begin
    posl := n mod 10;
    if posl > max then
      max := posl;
    n := n div 10
  end;
...
```

Язык Python

```
...
max = 0
while n > 0:
  posl = n % 10
  if posl > max:
    max = posl
  n = n//10
...
```

1.1.8. Определение минимальной цифры числа

Задача решается аналогично предыдущей (начальное значение искомой переменной мин можно принять равным 9).

1.1.9. Определение количества делителей числа

Прежде чем представлять методику решения задачи, заметим, что установить, является ли некоторое число b делителем числа a, можно, определив остаток от деления a на b. Если этот остаток равен нулю, то ответ положительный, в противном случае – отрицательный:

```
если mod(a, b) = 0
  то
    число b является делителем числа a
  иначе
    число b не является делителем числа a
все
```

Самый простой способ определить количество делителей числа n – это проверить по очереди делимость n на каждое из чисел 1, 2, 3, ..., n и в случае кратности n проверяемому числу увеличить значение некоторой переменной-счетчика.

В приведенной ниже программе, кроме переменной n, использованы следующие переменные величины:

кол_дел – искомое количество делителей (переменная-счетчик);
возм_дел – проверяемые числа.

```
алг Определение_количества_делителей
нач цел n, кол_дел, возм_дел
  ввод n
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0
      то  | возм_дел - делитель числа n
        кол_дел := кол_дел + 1
    все
  кц
  вывод нс, кол_дел
кон
```

Язык Паскаль

```
…
readln(n);
kol_del := 0;
for vozm_del := 1 to n do
  if n mod vozm_del = 0 then
    kol_del := kol_del + 1;
writeln(n)
```

Язык Python

```
n = int(input())
kol_del = 0
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0:
    kol_del = kol_del + 1
print(kol_del)
```

Возможен ряд усовершенствований программы, обеспечивающих ускорение ее работы. Одно их них учитывает тот факт, что в интервале от n/2 + 1 до n - 1 делителей числа n нет, т. е. количество проверок можно сократить вдвое:

```
кол_дел := 0
нц для возм_дел от 1 до div(n, 2)
  если mod(n, возм_дел) = 0
    то
      кол_дел := кол_дел + 1
  все
кц
| Учитываем также число n
кол_дел := кол_дел + 1
```

В описанном варианте можно при начальном присваивании переменной кол_дел сразу учесть делитель, равный n:

```
кол_дел := 1
...
```

а после оператора цикла не менять значение переменной кол_дел.

Язык Паскаль

```
...
readln(n);
kol_del := 1;
for vozm_del := 1 to n div 2 do
  if n mod vozm_del = 0 then
    kol_del := kol_del + 1;
writeln(kol_del);
...
```

Язык Python

```
n = int(input())
kol_del = 1
for vozm_del in range(1, n//2 + 1):
  if n % vozm_del == 0:
    kol_del = kol_del + 1
print(kol_del)
```

Обратим внимание на то, что в общем количестве делителей учитываются также значения 1 и n.

1.1.10. Поиск делителей натурального числа и сохранение их в массиве

Пусть имя массива для записи делителей – делители. Так как количество делителей заданного числа неизвестно, то в программе на школьном алгоритмическом языке, языке Паскаль и ряде других размер этого массива придется определить с запасом (примем, что указанное количество не превышает 10 000).

Программа:

```
алг Поиск_и_сохранение_делителей
нач цел n, возм_дел, кол_дел, цел таб делители[1 : 10000]
  ввод n
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0
      то
        | Найден очередной делитель
        кол_дел := кол_дел + 1
        | Записываем его в массив
        делители[кол_дел] := возм_дел
    все
  кц
  | Все делители (их количество - кол_дел) записаны в массив
...
```

Язык Паскаль

```
var n, kol_del: longint;
   deliteli: array[1..6] of
longint;
BEGIN
readln(n);
kol_del := 0;
for vozm_del := 1 to n do
  if n mod vozm_del = 0 then
    begin
      kol_del := kol_del + 1;
      deliteli[] := vozm_del
    end;
...
```

Язык Python

```
n = int(input())
deliteli = []
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0:
    deliteli.append(vozm_del)
...
```

Здесь также можно сократить количество проверяемых чисел.

1.1.11. Проверка числа на «простоту»

Как известно, *простыми* называют числа, которые имеют только два делителя: самого себя и 1. Такими являются числа 2[1], 3, 5, 7, 11, 13, 17, 19, 23, 31, 37, 41... .

Из определения простого числа вытекает методика решения указанной задачи – для получения ответа следует определить количество делителей и сравнить его с числом 2. Задача подсчета количества делителей была рассмотрена в п. 1.1.9. С учетом приведенного там простейшего варианта программы программа решения нашей задачи оформляется следующим образом:

```
алг
нач цел n, возм_дел, кол_дел
  ввод n
  | Подсчет делителей числа n
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0
      то
        кол_дел := кол_дел + 1
    все
  кц
  | Проверка
```

[1] По соглашению в теории чисел число 1 простым не является.

```
  если кол_дел = 2
    то
      вывод "Это число простое"
    иначе
      вывод "Это число простым не является"
  все
кон
```

Язык Паскаль

```
var n, kol_del: longint;
BEGIN
  readln(n);
  …
  kol_del := 0;
  for vozm_del := 1 to n do
    if n mod vozm_del = 0 then
      kol_del := kol_del + 1;
  if kol_del = 2 then
    writeln('Это число простое')
  else
    …
```

Язык Python

```
n = int(input())
kol_del = 0
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0:
    kol_del = kol_del + 1
if kol_del == 2:
  print("Это число простое»)
else:
  …
```

Заметим, что если в задаче происходит проверка на «простоту» большого количества чисел и/или проверяемые числа достаточно большие, то с целью ускорения работы программы целесообразно применять усовершенствованные методы подсчета количества делителей, один из которых описан в п 1.1.9.

Можно также прекратить подсчет делителей, когда их количество станет равно трем:

```
алг
нач цел n, возм_дел, кол_дел
  ввод n
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0
      то
        кол_дел := кол_дел + 1
        если кол_дел = 3
          то | Прекращаем подсчет делителей
            выход
        все
    все
```

```
  кц
  если кол_дел = 2
    то
      ...
кон
```

Язык Паскаль

```
...
kol_del := 0;
for vozm_del := 1 to n do
  begin
    if n mod vozm_del = 0 then
      kol_del := kol_del + 1;
    if kol_del = 3 then break;
if kol_del = 2 then
  ...
```

Язык Python

```
...
kol_del = 0
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0:
    kol_del = kol_del + 1
  if kol_del == 3:
    break
if kol_del == 2:
  ...
```

Задачи для самостоятельной работы

1. Дано натуральное число. Определить:

   а) сумму его четных делителей;

   б) количество его нечетных делителей;

   в) количество его делителей, больших числа d.

2. Определить количество простых чисел (см. выше) в заданном диапазоне натуральных чисел.

3. Натуральное число называется совершенным, если оно равно сумме своих делителей, включая 1 и, естественно, исключая это самое число. Например, число 6 – совершенное (6 = 1 + 2 + 3). Дано натуральное число. Выяснить, является ли оно совершенным.

4. Дано натуральное число. Определить медиану всех его делителей. *Медианой ряда чисел* называется число, стоящее посередине упорядоченного по возрастанию ряда (в случае, если количество чисел нечетное). Если же количество чисел в ряду четно, то медианой ряда является полусумма двух стоящих посередине чисел упорядоченного по возрастанию ряда. Например, для чисел 5, 2, 18, 8, 3 медиана равна 5, для чисел 5, 17, 3, 9, 14, 2 – 7.

1.2. Операции с элементами массива, отобранными по некоторому условию[1]

Для каждой рассмотренной задачи в данном разделе приведен фрагмент программы ее решения на школьном алгоритмическом языке и на языках Паскаль и Python. Использованы следующие основные величины:

- a – имя массива;
- n – общее количество элементов массива.

Смысл остальных величин можно легко определить по их именам. Принято, что элементы массива имеют целые значения.

1.2.1. Изменение элементов массива с заданными свойствами (удовлетворяющих некоторому условию)

Общая формулировка задач рассматриваемого типа: «Все элементы массива с заданными свойствами заменить на ...», где вместо многоточия указывается конкретное значение, на которое следует заменить указанные элементы, или правило, по которому надо провести замену.

Примеры задач

1. Дан массив целых чисел. Все четные элементы заменить числом 666.

2. Дан массив чисел, среди которых есть отрицательные. Все отрицательные элементы уменьшить на 10.

3. Дан массив целых чисел. Все элементы с нечетным индексом, оканчивающиеся цифрой 5, увеличить на 1.

Решение

```
нц для i от 1 до n
  | Если элемент обладает заданными свойствами
  если <условие>
    то
      | Меняем его значение
      a[i] := …
  все
кц
```

[1] Такая формулировка приведена в Кодификаторе элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2021 году единого государственного экзамена по информатике и ИКТ [4].

где *<условие>* – заданное условие. Это условие может определяться значением элемента массива a[i] или/и его индексом i (см., например, выше пример 3-й задачи).

Язык Паскаль

```
for i := 1 to n do
  if <условие>
    then a[i] := …;
```

Язык Python

```
for i in range(n):
    if <условие>:
        a[i] := …
```

1.2.2. Нахождение суммы элементов массива с заданными свойствами (удовлетворяющих некоторому условию)

Примеры задач рассматриваемого типа

1. В массиве записана стоимость 30 предметов. Определить общую стоимость тех предметов, которые стоят менее 200 руб.

2. В массиве записаны 20 целых чисел. Определить сумму тех из них, которые оканчиваются нулем.

3. В массиве записаны данные о количестве осадков, выпавших за каждый день января. Определить общее количество осадков, выпавших второго, четвертого, шестого и т. д. числа этого месяца. В программу должны вводиться данные за каждый день месяца.

Решение

Обсудим сначала более простую задачу: «Найти сумму всех элементов массива».

С использованием переменной сум, накапливающей в себе сумму значений уже рассмотренных элементов массива, фрагмент программы решения задачи оформляется следующим образом:

```
сум := 0
нц для i от 1 до n
  сум := сум + a[i]
кц
```

В программах на языках Паскаль[1] и Python задача может быть решена с использованием, соответственно, метода и функции sum:

Язык программирования	Оператор, решающий задачу
Паскаль	`summa := m.sum`
Python	`summa = sum(m)`

[1] Здесь (и далее в книге) имеется в виду система программирования PascalABC.NET [11].

Вернемся теперь к основной задаче. Отличие задач данного типа от только что рассмотренной в том, что добавлять значение элемента массива к уже рассчитанной ранее сумме следует только тогда, когда элемент обладает заданными свойствами:

```
сум := 0
нц для i от 1 до n
  | Если элемент обладает заданными свойствами
  если <условие>
    то
     | Учитываем его значение в сумме
     сум := сум + a[i]
  все
кц
```

В данном случае *<условие>* также может определяться значением элемента массива `a[i]` или/и его индексом `i`.

Однако так можно оформить программу только в случае, когда известно, что в массиве имеется хотя бы один элемент с заданными свойствами. Если допускается, что таких элементов может не быть, то фрагмент, связанный с выводом ответа, должен иметь вид:

```
если сум = 0
  то
    вывод нс, "Чисел, удовлетворяющих условию, в массиве нет"
  иначе
    вывод нс, "Сумма чисел, удовлетворяющих условию, равна ", сум
все
```

Примечание. Случай, когда элементы с заданными свойствами в массиве имеются, но их сумма равна нулю, не учитывается.

Язык Паскаль

```
...
sum := 0;
for i := 1 to n do
  if <условие>
    then sum := sum + a[i];
if sum = 0
  then write('Чисел, удовлетворяющих
условию, в массиве нет'
  else
    ...
```

Язык Python

```
sum = 0
for i in range(n):
  if <условие>:
    sum = sum + a[i]
if sum == 0:
  ...
```

1.2.3. Нахождение количества элементов массива с заданными свойствами

Примеры задач рассматриваемого типа

1. В массиве хранятся 40 целых чисел. Найти количество четных чисел.

1. В массиве записан рост 22 юношей. Определить, сколько из них имеют рост менее 165 см.

2. В массиве записаны оценки по информатике каждого из 25 учеников класса. Определить количество пятерок.

Особенность задач данного типа в том, что в случае, когда элемент обладает заданными свойствами (удовлетворяет некоторому условию), искомое количество увеличивается на 1:

```
кол := 0
нц для i от 1 до n
  | Если элемент обладает заданными свойствами
  если <условие>
    то
      | Учитываем его в искомом количестве
      кол := кол + 1
  все
кц
вывод нс, "Количество чисел, удовлетворяющих условию, равно ", кол
```

В данном случае условие в команде **если** (в условном операторе) определяется значением элемента массива `a[i]` или одновременно значениями `a[i]` и `i`. Количество элементов, зависящих только от значения индекса `i`, может быть найдено без использования оператора цикла (убедитесь в этом!).

Язык Паскаль

```
…
kol := 0;
for i := 1 to n do
  if <условие>
    then
      kol := kol + 1; { или inc(kol)
}
…
```

Язык Python

```
kol = 0
for i in range(n):
  if <условие>:
    kol = kol + 1
…
```

В языке Python задача нахождения количества элементов списка, *равных* некоторому значению X, может быть решена с использованием функции count():

```
kol = m.count(X)
```

Описанный же выше способ расчетов – универсальный и может быть применен для подсчета количества элементов, удовлетворяющих любому условию.

Метод count предусмотрен также в языке Паскаль. С особенностями его использования ознакомьтесь самостоятельно.

1.2.4. Нахождение среднего арифметического значения элементов массива с заданными свойствами

Для нахождения искомого значения необходимо определить сумму элементов массива с заданными свойствами и их количество. Такие две задачи мы уже решили ранее. Здесь их можно объединить в одном операторе цикла:

```
сум := 0
кол := 0
нц для i от 1 до n
  | Если элемент обладает заданными свойствами
  если <условие>
    то
      | Учитываем его значение в сумме
      сум := сум + a[i]
      | и учитываем этот элемент в количестве
      кол:= кол + 1
  все
кц
| Подсчет результата
сред_арифм := сум/кол
```

Обратите внимание на то, что многократно определять значение сред_арифм в «теле» условного оператора (команды **если**) необходимости нет. Это можно сделать один раз после окончания оператора цикла. Однако может оказаться, что чисел, удовлетворяющих заданному условию, в массиве не окажется. В этом случае при расчете будет иметь место деление на ноль, что недопустимо. Правильное оформление:

```
...
```

```
| Подсчет и вывод результата
если кол > 0
  то
   средн_ариф := сум/кол
   вывод нс, "Среднее арифметическое: ", сред_арифм
  иначе
   вывод нс, "Чисел, удовлетворяющих условию, в массиве нет"
все
```

Язык Паскаль

```
…
sum := 0;
kol := 0;
for i := 1 to n do
  if <условие> then
    begin
      sum := sum + a[i];
      kol := kol + 1
    end;
if kol > 0 then
  begin
    sred_arifm := sum/kol;
    write('Среднее арифметическое: ',
          sred_arifm:7:2)
  end
else
  write('Чисел, удовлетворяющих условию,
в массиве нет')
…
```

Язык Python

```
sum = 0
kol = 0
for i in range(n):
  if <условие>:
    sum = sum + a[i]
    kol = kol + 1
if kol > 0:
  …
else
  …
```

1.2.5. Нахождение максимального количества подряд идущих элементов массива, обладающих заданными свойствами

Пример задачи: «Определить максимальное количество подряд идущих четных элементов в заданном целочисленном массиве».

В дальнейшем для краткости будем называть участок массива с указанными элементами подмассивом. Значит, для решения обсуждаемой задачи надо определить максимальное из чисел – длин каждого подмассива (количество элементов в них). Это можно сделать по аналогии с определением максимальной цифры натурального числа (см. п. 1.1.7), только вместо цифр следует использовать длину каждого подмассива.

Используем в программе следующие основные величины:

- кол – число элементов в текущем подмассиве;
- макс_кол – искомое значение (максимальное количество подряд идущих элементов с заданными свойствами).

Можно рассуждать так. Если очередной элемент обладает заданными свойствами, то подмассив таких элементов продолжается или начался – увеличиваем его длину кол на 1, иначе – текущий подмассив кончился, и в этом случае:

1) сравниваем его длину с максимальной длиной уже рассмотренных ранее подмассивов макс_кол. Если длина текущего подмассива больше, то принимаем ее в качестве нового значения величины макс_кол;
2) имея в виду обработку следующего подмассива, обнуляем значение величины кол.

Фрагмент программы на основе сделанных рассуждений:

```
кол := 0
макс_кол := 0
нц для i от 1 до n
  если <условие>
   то
    | Подмассив продолжается или начался
    | Увеличиваем его длину на 1
    кол :=  кол + 1
   иначе | Встретился элемент, не обладающий заданными свойствами
    | Текущий подмассив кончился
    | Сравниваем его длину со значением макс_кол
    если кол >  макс_кол
     то
      макс_кол := кол
    все
    кол :=  0 | Новое значение
  все
кц
```

Однако при таких рассуждениях в случае, когда последний, n-й, элемент массива также обладает заданными свойствами (например, четный), последний подмассив не будет учтен. Значит, нужно дополнительно учесть и такую возможность, сравнив длину этого подмассива с максимальной длиной в любом случае:

```
| Проверяем длину последнего (возможного) подмассива
если кол > макс_кол[1]
  то
    макс_кол := кол
все
| Выводим ответ
вывод нс, макс_кол
```

Обратим внимание на то, что после окончания текущего подмассива величине кол присваивается нулевое значение независимо от результата сравнения величин кол и макс_кол.

Язык Паскаль

```
...
kol := 0;
max_kol := 0;
for i := 1 to n do
  if <условие>
   then { Подмассив продолжается или начался.
          Увеличиваем его длину на 1 }
     kol := kol + 1
   else { Встретился элемент, не обладающий
          заданными свойствами - текущий подмассив кончился.
          Сравниваем его длину со значением max_kol }
       begin
         if kol >  max_kol then max_kol := kol;
         kol := 0 { Новое значение }
       end;
{ Проверяем длину последнего (возможного) подмассива }
if kol > max_kol
  then max_kol := kol;
{ Выводим ответ }
write(max_kol);
```

Язык Python

```
kol = 0
max_kol = 0
for i in range(n):
  if <условие>:
        # Подмассив продолжается или начался.
          Увеличиваем его длину на 1
```

[1] Если массив не заканчивается элементом с заданными свойствами, то произойдет сравнение значения макс_кол с нулем.

```
      kol = kol + 1
    else: # Встретился элемент, не обладающий заданными свойствами -
          # текущий подмассив кончился.
          # Сравниваем его длину со значением max_kol
        if kol >  max_kol:
           max_kol = kol
        kol = 0 # Новое значение
# Проверяем длину последнего (возможного) подмассива
if kol > max_kol:
  max_kol = kol
# Выводим ответ
print(max_kol)
```

Можно сократить размер программы, не рассматривая возможный последний подмассив.

Логика действий описана в комментариях к программе:

```
кол := 0
макс_кол := 0
нц для i от 1 до n
  если <условие>
    то
      | Подмассив продолжается или начался
      | Увеличиваем его длину на 1
      кол :=  кол + 1
      | Сравниваем его длину со значением макс_кол
      если кол >  макс_кол
        то
          макс_кол := кол
      все
    иначе | Встретился элемент, не обладающий заданными свойствами
    | Текущий подмассив кончился
    кол :=  0 | Новое значение
  все
кц
вывод нс, макс_кол
```

Возможны также задачи рассмотренного типа, в которых начальное значение величины кол и ее значение после окончания некоторого подмассива принимается равным 1. Это имеет место, когда первый элемент подмассива также учитывается в его длине.

Пример: «Определить максимальное количество подряд идущих совпадающих элементов целочисленного массива».

Ясно, что здесь первый из совпадающих элементов также должен быть учтен. Поэтому начальные значения переменных величин, хранящих длину текущего подмассива и максимальную длину подмассивов, равны 1 (если потом выяснится, что во всех парах находящихся рядом элементов массива нет одинаковых, то искомая величина будет равна 1):

```
кол_совп := 1
макс_совп := 1
нц для i от 2 до n
  если m[i] = m[i - 1]
    то
      кол_совп := кол_совп + 1
    иначе
      если кол_совп >  макс_совп
        то
          макс_совп := кол_совп
      все
      кол_совп := 1 | Новое значение
  все
кц
если кол_совп >  макс_совп
  то
    макс_совп := кол_совп
все
вывод макс_совп
```

где кол_совп – количество подряд идущих совпадающих элементов в текущем подмассиве; макс_совп – максимальное количество подряд идущих совпадающих элементов в подмассивах.

Язык Паскаль

```
...
kol_sovp := 1;
max_sovp := 1;
for i := 2 to n do
  if m[i] = m[i - 1] then
    kol_sovp := kol_sovp + 1
  else
    begin
      if kol_sovp > max_sovp
        then max_sovp := kol_sovp;
      kol_sovp := 1
    end;
```

Язык Python

```
kol_sovp = 1
max_sovp = 1
for i in range(1, n):
  if m[i] == m[i - 1]:
    kol_sovp = kol_sovp + 1
  else:
   if kol_sovp > max_sovp:
      max_sovp = kol_sovp
    kol_sovp = 1
if kol_sovp > max_sovp:
  max_sovp = kol_sovp
print(max_sovp)
```

```
if kol_sovp > max_sovp
  then max_sovp := kol_sovp;
write(max_sovp);
...
```

1.2.6. Нахождение максимальной суммы подряд идущих элементов массива, обладающих заданными свойствами

Пример задачи: «Определить максимальную сумму подряд идущих четных элементов в заданном целочисленном массиве».

Задача решается во многом аналогично предыдущей (при обработке подмассива рассчитывается сумма значений элементов, а по его окончании сумма сравнивается с максимальной суммой, найденной ранее). Конечно, и здесь нужно при необходимости дополнительно проверить возможный последний подмассив:

```
сум := 0
макс_сум := 0
нц для i от 1 до n
  если <условие>
    то
      | Подмассив продолжается или начался
      | Учитываем текущий элемент в сумме
      сум := сум + m[i]
    иначе | Встретился элемент, не обладающий заданными свойствами
      | Текущий подмассив кончился
      | Сравниваем сумму его элементов со значением макс_сум
      если сум >  макс_сум
        то
          макс_сум := сум
      все
      сум := 0 | Новое значение
  все
кц
| Проверяем сумму элементов последнего (возможного) подмассива
если сум >  макс_сум
  то
    макс_сум := сум
все
вывод нс, макс_сум
```

Язык Паскаль

```
sum := 0;
max_sum := 0;
for i := 1 to n do
  if <условие>
   then
     sum := sum + m[i]
   else
      begin
        if sum > max_sum then
          max_sum := sum;
        summa := 0
      end;
if sum > max_sum then max_sum := sum;
write(max_sum);
```

Язык Python

```
sum = 0
max_sum = 0
for i in range(n):
  if <условие>:
    sum = sum + m[i]
  else:
    if sum > max_sum:
      max_sum = sum
    summa = 0
if sum > max_sum:
  max_sum = sum
print(max_sum)
```

Возможны также задачи обсуждаемого типа, в которых начальное значение величины сум и ее значение после окончания некоторого подмассива принимается равным первому элементу массива (и, соответственно, текущему). Это имеет место, когда первый элемент подмассива также учитывается в его сумме.

Пример: «Определить сумму элементов наибольшей возрастающей последовательности подряд идущих элементов массива».

Обратим внимание на то, что здесь необходимо определить не максимальную длину[1] (количество элементов) подмассива и не максимальную сумму его элементов, а сумму элементов в подмассиве максимальной длины. Значит, нужно в ходе поиска такого подмассива рассчитывать и запоминать также сумму его элементов (в том числе и первого элемента подмассива).

В программе решения приведенного примера используем следующие основные величины:

кол_возр – количество элементов возрастающей последовательности в текущем подмассиве;

сумма_возр – сумма значений элементов в таком подмассиве;

макс_кол – максимальное количество элементов в возрастающих последовательностях элементов;

макс_сумма – максимальная сумма значений элементов в соответствующей последовательности.

[1] Хотя эту величину также придется рассчитывать, но как вспомогательную.

Соответствующий фрагмент программы:

```
кол_возр := 1          | Учитываем первый
сумма_возр := m[1]     | элемент
макс_кол := 1
нц для i от 2 до n
  если m[i] > m[i - 1]
    то
      кол_возр := кол_возр + 1
      сумма_возр := сумма_возр + m[i]
    иначе
      если кол > макс_кол
        то
          макс_кол := кол_возр
          макс_сумма := сумма_возр
      все
      | Новые значения
      сумма_возр := m[i]
      кол_возр := 1
  все
кц
если кол_возр > макс_кол
  то
    макс_сумма := сумма_возр
    | Здесь уточняем только значение макс_сумма
все
вывод нс, макс_сумма
```

Обратим внимание на то, что начальные значения величины кол_возр принимаются равными 1.

Язык Паскаль

```
kol_vozr := 1;
summa_vozr := m[1];
max_kol := 1;
for i := 2 to n do
  if m[i] m[i] > m[i - 1] then
    begin
      kol_vozr := kol_vozr + 1;
      summa_vozr := summa_vozr + m[i]
    end
  else
    begin
```

Язык Python

```
kol_vozr = 1
summa_vozr = m[0]
max_kol = 1
for i in range(1, n):
  if m[i] m[i] > m[i - 1]:
    kol_vozr = kol_vozr + 1
    summa_vozr = summa_
vozr + m[i]
  else:
    if kol_vozr > max_kol:
      max_kol = kol_vozr
      max_summa = summa_vozr
```

Язык Паскаль

```
      if kol_vozr > max_kol then
        begin
          max_kol := kol_vozr;
          max_summa := summa_vozr
        end;
      summa_vozr := m[i];
      kol_vozr := 1
    end;
if kol_vozr > max_kol
  then max_summa := summa_vozr;
write(max_summa);
```

Язык Python

```
    summa_vozr = m[i]
    kol_vozr = 1
if kol_vozr > max_kol:
  max_summa = summa_vozr
print(max_summa)
```

В заключение заметим, что задачи нахождения максимального/минимального значения среди элементов с заданными свойствами и номера такого значения рассматриваются в разделе 1.4.

1.3. Линейный поиск элемента

1.3.1. Проверка факта наличия в массиве элемента с заданным значением

Пример задачи: «Определить, есть ли в целочисленном массиве число 13».

Решение

Если перебрать все элементы массива, каждый сравнивать с числом 13 и в случае равенства выводить «Да, есть», в противном случае выводить «Нет, такого числа нет»:

```
нц для i от 1 до n
  если a[i] = 13
    то
      вывод нс, "Да, число 13 есть"
    иначе
      вывод нс, "Нет, такого числа нет"
  все
кц
```

– то на экран будет выведено несколько ответов (причем противоречащих друг другу). Ясно, что ответ (правильный) должен выводиться только один раз, и приведенное решение неприемлемо. Еще одна типичная ошибка, встречающаяся при решении

обсуждаемой задачи, – использование переменной (логического типа или принимающей значения 0 и 1), которая фиксирует факт равенства 13 для каждого проверяемого элемента, и вывод ответа в зависимости от значения этой переменной. В приведенном ниже фрагменте программы имя этой переменной – имеется.

```
нц для i от 1 до n
  если a[i] = 13
    то
      имеется := да     | или имеется := 1
    иначе
      имеется := нет    | или имеется := 0
  все
кц
если имеется  | или имеется = да или имеется = 1
  то
    вывод нс, "Да, число 13 есть"
  иначе
    вывод нс, "Нет, такого числа нет"
все
```

Здесь ответ, выводимый на экран один раз, может быть ошибочным – он зависит от того, равен ли 13 последний элемент массива!

Правильный вариант:

```
имеется := нет          | или имеется := 0
нц для i от 1 до n
  если a[i] = 13
    то
      имеется := да     | или имеется := 1
  все
кц
если имеется            | или имеется = да или имеется = 1
  то
    вывод нс, "Да, число 13 есть "
  иначе
    вывод нс, "Нет, такого числа"
все
```

Можно также решить задачу так:

1) подсчитать количество элементов массива, равных 13 (такая задача рассмотрена в п. 1.2.3);

2) в зависимости от найденного количества вывести соответствующий ответ.

В обоих описанных вариантах решения массив просматривается полностью, в то время как искомое значение может оказаться в начале массива, и после нахождения числа 13 продолжать проверку остальных элементов массива нерационально. Желательно прекратить обработку массива после нахождения искомого элемента. В программах на языках программирования, в которых предусмотрена процедура выхода из цикла (break или др.), для этого следует использовать эту процедуру. Можно также использовать оператор цикла с условием. Например, для нашей задачи соответствующий фрагмент должен быть оформлен в виде:

```
i := 1
нц пока i < n и не (a[i] <> 13)   | Обратите внимание на условие
  i := i + 1
кц
```

В результате величина i не будет превышать значение n. Если значение элемента, на котором обработка массива остановилась (на последнем элементе или «досрочно»), равно 13, то это определяет один из вариантов ответа, в противном случае – второй вариант:

```
| Вывод результата
если a[i] = 13
  то
    вывод нс, "Да, число 13 есть"
  иначе
    вывод нс, "Нет, такого числа нет"
все
```

Обратим внимание на то, что при этом величина логического типа (или принимающая значение 0 или 1) не используется.

1.3.2. Проверка факта наличия в массиве элемента с заданными свойствами

Пример задачи: «Определить, есть ли в целочисленном массиве четное число».

Решение

Различные методы решения данной задачи аналогичны описанным в п.1.3.1.

```
имеется := нет          | или имеется := 0
нц для i от 1 до n
  если <условие>
    то
      имеется := да     | или имеется := 1
  все
кц
если имеется            | или имеется = да или имеется = 1
  то
    вывод нс, "Да, имеется"
  иначе
    вывод нс, "Нет, такого элемента нет"
все


i := 1
нц пока i < n и не (mod(a[i], 2) = 0)   | Обратите внимание на условие
  i := i + 1
кц
| Вывод результата
если mod(a[i], 2) = 0
  то
    вывод нс, "Да, имеется"
  иначе
    вывод нс, "Нет, такого элемента нет"
все
```

В общем случае программа имеет вид:

```
i := 1
нц пока i < n и не <условие>
  i := i + 1
кц
| Вывод результата
если <условие> | Условие, соответствующее заданным свойствам
  то
    вывод нс, "Да, имеется"
  иначе
    вывод нс, "Нет, такого элемента нет"
все
```

Можно также решить задачу так:

1) подсчитать количество элементов массива с заданными свойствами (такая задача рассмотрена в разделе 1.2);

2) в зависимости от найденного количества вывести соответствующий ответ.

1.3.3. Поиск индекса элемента массива, равного некоторому числу

В приведенных далее фрагментах программы величина значение – это число, индекс которого (искомый_индекс) ищется.

```
нц для i от 1 до n
  если a[i] = значение
    то
      искомый_индекс := i
  все
кц
```

Прежде чем обсуждать представленный вариант, предлагаем читателю ответить на вопрос: индекс какого элемента будет найден, если в массиве окажутся несколько элементов, значение которых равно искомому?

Нетрудно заметить, что при таком оформлении, в случае если числа значение в массиве нет, результат будет неправильным (в программе на языке Паскаль), а при выполнении программ на школьном алгоритмическом языке и языке Python появится сообщение об ошибке, связанной с тем, что значение переменной искомый_индекс не определено. Чтобы учесть в программе возможность такого случая, необходимо изменить фрагмент следующим образом:

```
искомый_индекс := 0[1]  | Условно
нц для i от 1 до n
  если a[i] = значение
    то
      искомый_индекс := i
  все
кц
| Вывод результата
если искомый_индекс > 0
  то
    вывод нс, "Индекс этого элемента: ", искомый_индекс
```

[1] В программе на языках программирования, в которых нумерация элементов массива начинается с нуля, условное начальное значение переменной искомый_индекс должно быть равно –1 (или –2 или т. п.).

```
  иначе
    вывод нс, "Такого числа в массиве нет»
все
```

Язык Паскаль

```
iskomiy_index := 0;
for i := 1 to n do
  if a[i] = znachenie
  then iskomiy_index := i;
if iskomiy_index > 0 then
  write('Индекс этого элемента: ',
     iskomiy_index)
else write('Такого числа в массиве
нет');
```

Язык Python

```
iskomiy_index = -1
for i in range(n):
  if a[i] == znachenie:
    iskomiy_index = i
if iskomiy_index > -1:
  print("Индекс этого элемента:",
        iskomiy_index)
else
  print("Такого числа
        в массиве нет")
```

1.3.4. Поиск индекса элемента массива с заданными свойствами

Задача решается аналогично предыдущей (вместо конкретного значения используется условие, соответствующее заданным свойствам).

1.3.5. Поиск индекса *первого* элемента массива, равного некоторому числу

Для решения такого варианта задачи можно в приведенной чуть выше программе использовать величину логического типа первый, принимающую значение «истина» (**да**), если элемент, равный заданному числу, встретился в массиве впервые, и «ложь» (**нет**) – в противном случае:

```
| Встреченный элемент будет первым
первый := да
искомый_индекс := 0
нц для i от 1 до n
  если a[i] = значение
    то
     | Проверяем, впервые ли встретилось в массиве заданное число
     если первый = да
       | Если впервые
       то
```

```
        | Найден искомый элемент массива
        | Запоминаем его индекс
        искомый_индекс := i
        | Другие элементы, равные заданному числу,
        | уже будут не первыми
        первый := нет
    все
  все
кц
| Вывод результата
если искомый_индекс > 0
  то
    вывод нс, "Индекс этого элемента: ", искомый_индекс
  иначе
    вывод нс, "Такого числа в массиве нет"
все
```

Язык Паскаль

```
perviy := true;
iskomiy_index := 0;
for i := 1 to n do
  if a[i] = znachenie then
    if perviy then
      begin
        iskomiy_index := i;
        perviy := false
      end;
if iskomiy_index > 0 then
  write('Индекс этого элемента: ',
    iskomiy_index)
else
  write('Такого числа в массиве
нет');
```

Язык Python

```
perviy = True
iskomiy_index = -1
for i in range(n):
  if a[i] == znachenie:
    if perviy:
      iskomiy_index = i
      perviy = False
if iskomiy_index > -1:
  print("Индекс этого
элемента:",
        iskomiy_index)
else
  print("Такого числа в массиве
нет")
```

Можно также провести проверку, начиная с последнего элемента массива:

```
искомый_индекс := 0
нц для i от n до 1 шаг -1
  если a[i] = значение
    то
      искомый_индекс := i
```

```
  все
кц
| Вывод результата
если искомый_индекс > 0
  то
    вывод нс, "Индекс этого элемента: ", искомый_индекс
  иначе
    вывод нс, "Такого числа в массиве нет"
все
```

Язык Паскаль

```
iskomiy_index := 0;
for i := n downto 1 do
  if a[i] = znachenie
    then iskomiy_index := i;
if iskomiy_index > 0 then
  write('Индекс этого элемента: ',
      iskomiy_index)
else
  write('Такого числа в массиве
нет');
```

Язык Python

```
iskomiy_index = -1
for i in range(n - 1, 1, -1):
  if a[i] == znachenie:
    if perviy:
      iskomiy_index = i
if iskomiy_index > -1:
  print("Индекс этого элемента:",
        iskomiy_index)
else
  print("Такого числа в массиве
нет")
```

Видно, что такой метод значительно упрощает программу.

Заметим, что в языке программирования Python имеется стандартная функция `index`, которая возвращает индекс первого элемента массива, равного заданному значению `N`:

```
ind = m.index(N)   # Функция вызывается как метод
```

Однако надо учитывать, что, если нужного элемента в массиве нет, при выполнении программы эта строчка вызовет ошибку. Кроме того, функция `index` возвращает индекс элемента, *равного* заданному значению, а описанный до этого метод – универсальный и применим к нахождению индекса первого элемента в массиве с любыми заданными свойствами (четного, положительного и т. п.) – см. следующую задачу.

В языке программирования Паскаль имеется аналогичная функция `FindIndex`. С ее особенностями ознакомьтесь самостоятельно.

1.3.6. Поиск индекса первого элемента массива с заданными свойствами

Задача решается аналогично предыдущей (вместо конкретного значения используется условие, соответствующее заданным свойствам).

При решении задач 1.3.2–1.3.6, как и при решении задачи 1.3.1, можно прекратить обработку массива после нахождения (возможного) искомого элемента.

1.4. Задачи на нахождение максимальных (минимальных) элементов массива, их индексов, количеств и т. п.

1.4.1. Определение максимального элемента массива

Алгоритм решения этой задачи аналогичен алгоритму действий человека, который определяет максимальное значение в некоторой одномерной таблице с числами:

12	6	20	13	4	45	7	12	9	34

Сначала он смотрит в первую ячейку таблицы и запоминает записанное там число. Затем смотрит во вторую ячейку и, в случае если имеющееся там число больше запомненного, в качестве максимального запоминает новое число. Для остальных ячеек действия аналогичны.

Соответствующий фрагмент программы:

```
| Начальное присваивание значения искомой величине
макс := a[1]
| Рассматриваем остальные элементы
нц для i от 2 до n
  | Сравниваем i-й элемент со значением макс
  если a[i] > макс
    то
      | Принимаем встреченный элемент в качестве макс
      макс := a[i]
  все
кц
| Выводим ответ
вывод нс, макс
```

Язык Паскаль

```
max := a[1];
for i := 2 to n do
  if a[i] > max
    then max := a[i];
write(max);
```

Язык Python

```
max = a[0]
for i in range(1, n):
  if a[i] > max:
    max = a[i]
print(max)
```

Обсудим вопрос – можно ли не рассматривать отдельно первый элемент массива, а обрабатывать в цикле *все* элементы?

```
| Рассматриваем все элементы
нц для i от 1 до n
  | Сравниваем i-й элемент со значением макс
  если a[i] > макс
    то
     | Принимаем встреченный элемент в качестве макс
     макс := a[i]
  все
кц
| Выводим ответ
вывод нс, макс
```

На первый взгляд – да, можно. Однако это не так, точнее, так можно оформлять программу не во всех случаях. Например, если стоит задача определения максимальной температуры в первой декаде января и значения элементов массива, в котором записана температура каждого дня, такие:

–6, –5, –4, –6, –3, –3, –2, –5, –5, –4,

то в приведенном варианте программы при ее выполнении на школьном алгоритмическом языке и языке Python появится сообщение об ошибке, связанное с тем, что значение переменной макс не определено, а в программе на языке Паскаль результат будет неправильным (будет выведен ответ 0).

Появится сообщение об ошибке, связанное с тем, что значение макс при выводе ответа не определено.

Если же до оператора цикла присвоить величине макс нулевое значение:

```
макс := 0
```

– то результат окажется неправильным (он будет равен 0).

Можно сделать вывод о том, что краткий вариант программы может быть применен, если известно, что в обрабатываемом массиве не все значения отрицательны.

В общем случае краткий вариант может быть использован, если известно минимальное число мин обрабатываемого массива. Например, часто известен диапазон возможных значений элементов в массиве. В таких случаях фрагмент программы решения задачи может быть оформлен так:

```
макс := мин
| Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > макс
    то
      макс := a[i]
  все
кц
| Выводим ответ
вывод нс, макс
```

где мин – нижняя граница диапазона возможных значений.

Язык Паскаль

```
max := min;
for i := 1 to n do
  if a[i] > max
    then max := a[i];
write(max);
```

Язык Python

```
max = min
for i in range(n):
  if a[i] > max:
    max = a[i]
print(max)
```

1.4.2. Определение минимального элемента массива

Задача решается аналогично предыдущей (конечно, с необходимыми изменениями).

Заметим, что в программах на языках Паскаль и Python две последние задачи могут быть решены с использованием, соответственно, методов и функций max и min:

Язык программирования	Оператор, решающий задачу
Паскаль	`max:= m.max;` `min:= m.min;`
Python	`max = max(m)` `min = min(m)`

Однако рассмотренные нами методы являются полезными и будут использованы при решении задач, для которых стандартные методы применить нельзя (см. п. 1.4.5–1.4.8).

1.4.3. Определение индекса максимального элемента массива

Здесь также алгоритм решения задачи аналогичен алгоритму действий человека, определяющего номер ячейки с максимальным значением в некоторой одномерной таблице с числами, – сначала он запоминает первое число и номер 1, а затем рассматривает второе число. Если оно больше того числа, которое помнил, то он запоминает новое число и номер 2 и переходит к следующему, в противном случае просто переходит к следующему, третьему, числу и делает то же самое и т. д.

Фрагмент программы, в котором решается задача:

```
| Начальное присваивание значений искомым величинам
макс := a[1]
номер_макс := 1
| Рассматриваем остальные элементы
нц для i от 2 до n
  если a[i] > макс
    то
      | Принимаем встреченный элемент в качестве макс
      максимальное := a[i]
      | а его индекс - в качестве номер_макс
      номер_макс := i
  все
кц
| Выводим ответ
вывод нс, номер_макс
```

Язык Паскаль

```
max := a[1];
nomer_max := 1;
for i := 2 to n do
  if a[i] > max then
    begin
      max := a[i];
      nomer_max := i
    end;
write(nomer_max);
```

Язык Python

```
max = a[0]
nomer_max = 0
for i in range(1, n):
  if a[i] > max:
    max = a[i]
    nomer_max = i
print(nomer_max)
```

Возникает вопрос: индекс какого элемента будет найден, если в массиве есть несколько элементов с максимальным значением? Что надо изменить в приведенном фрагменте, чтобы находился индекс последнего элемента с таким значением?

Если диапазон возможных значений элементов массива известен, то можно оформить фрагмент так:

```
макс := мин
| Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > макс
    то
      | Принимаем встреченный элемент в качестве макс
      макс := a[i]
      | а его индекс - в качестве номер_макс
      номер_макс := i
  все
кц
| Выводим ответ
вывод нс, номер_макс
```

где мин – нижняя граница диапазона возможных значений.

Видно, что в приведенных программах в результате определяются два значения – номер_макс и макс.

Если последнее значение находить не требуется, то без величины макс можно вообще обойтись. В самом деле, если нам известно значение индекса максимального среди рассмотренных элементов, то мы знаем и значение соответствующего элемента (оно равно a[номер_макс]):

```
номер_макс := 1
нц для i от 2 до n
  если a[i] > a[номер_макс]
    то
      номер_макс := i
  все
кц
вывод нс, номер_макс
```

Язык Паскаль

```
nomer_max := 1;
for i := 2 to n do
  if a[i] > a[nomer_max]
    then nomer_max := i;
write(nomer_max);
```

Язык Python

```
nomer_max = 0
for i in range(1, n):
  if a[i] > a[nomer_max]:
    nomer_max = i
print(nomer_max)
```

1.4.4. Нахождение индекса минимального элемента

Задача решается способами, аналогичными описанным применительно к предыдущей задаче.

1.4.5. Нахождение минимального (максимального) элемента массива и количества элементов, равных ему

Задача может быть решена двумя способами:

1) за два прохода по массиву;

2) за один проход по массиву.

В первом случае решение очевидно – на первом проходе следует найти минимальный (максимальный) элемент массива (см. задачи 1.4.1 и 1.4.2), а на втором – подсчитать количество элементов, равных минимальному (максимальному) значению (см. задачу 1.2.3).

Во втором случае идея решения такая: проходя по массиву, кроме значения максимума, контролировать также количество элементов, равных максимальному (кол_макс). Если очередной элемент оказывается больше текущего максимума – он принимается в качестве максимального значения, а величина кол_макс – равной 1. Если же очередной элемент не больше максимального, то сравниваем его с максимумом. Если они равны, то встретился еще один максимум, и значение кол_макс увеличиваем на 1. Соответствующий фрагмент:

```
| Начальное присваивание значений искомым величинам
макс := a[1]
кол_макс := 1
| Рассматриваем остальные элементы
нц для i от 2 до n
  если a[i] > макс
    то       | Встретился новый максимум
      | Принимаем его в качестве значения макс
      макс := a[i]
```

```
      | Пока он - единственный
      кол_макс := 1
    иначе
      | Проверяем, не равно ли очередное значение "старому» максимуму
      если a[i] = макс
        то  | Встретился еще один максимум
            | Учитываем это
            кол_макс := кол_макс + 1
      все
  все
кц
вывод нс, кол_макс
```

Язык Паскаль

```
max := a[1];
kol_max := 1;
for i := 2 to n do
  if a[i] > max then
    begin
      max := a[i];
      kol_max := 1
    end
  else
    if a[i] = max then
       kol_max := kol_max + 1;
writeln(kol_max);
```

Язык Python

```
max = a[0]
kol_max = 1
for i in range(1, n):
  if a[i] >  max:
    max = a[i]
    kol_max = 1
  else:
    if a[i] == max:
      kol_max = kol_max + 1
print(kol_max)
```

Если диапазон возможных значений элементов массива известен, то и здесь можно отдельно не рассматривать первый элемент:

```
| Начальное присваивание значений искомым величинам
макс := мин - 1
кол_макс := 0
| Рассматриваем все элементы
нц для i от 1 до n
  если a[i] > макс
  ...
```

где мин – нижняя граница диапазона возможных значений.

Язык Паскаль

```
max := min - 1;
kol_max := 0;
for i := 1 to n do
  if a[i] > max then …
```

Язык Python

```
max = min - 1
kol_max = 0
for i in range(n):
  if a[i] > max:
    …
```

1.4.6. Нахождение количества минимальных элементов

Задача решается аналогично предыдущей (конечно, с необходимыми изменениями).

1.4.7. Определение минимального значения среди тех элементов массива, которые удовлетворяют некоторому условию

Пример: нахождение минимального четного элемента массива.

Решение

Сначала для простоты примем, что известен тот факт, что числа, удовлетворяющие заданному условию, в исследуемом массиве имеются.

Задача решается во многом аналогично задаче 1.4.2 с учетом того обстоятельства, что в качестве начального значения искомого минимума нельзя принимать значение первого элемента массива, так как он может не входить во множество элементов, удовлетворяющих заданному условию:

```
мин := X
нц для i от 1 до n
  если <условие>
    то | Встретилось число, удовлетворяющее заданному условию
     | Сравниваем его с величиной мин
     если a[i] < мин
       то
        мин := a[i]
     все
  все
кц
| Выводим ответ
вывод нс, мин
```

где X – число, о котором заведомо известно, что оно не меньше максимального из чисел, для которых определяется минималь-

ное значение. Например, если в массиве записаны только отрицательные значения, то можно принять X равным нулю.

Язык Паскаль

```
min := X;
for i := 1 to n do
  { Если очередной элемент обладает заданными свойствами }
  if <условие> then
     { Сравниваем его с величиной min }
     if a[i] < min
       then min := a[i];
{ Выводим ответ }
write(min);
```

Язык Python

```
min = X
for i in range(n):
  if <условие>:
    if a[i] < min:
      min = a[i]
print(min)
```

Если же такое значение заранее неизвестно, задача несколько усложняется. Можно поступить так:

1) найти первое значение в массиве, удовлетворяющее заданному условию;
2) принять его в качестве минимального;
3) рассмотреть оставшиеся элементы и те из них, которые удовлетворяют заданному условию, сравнить с минимальным среди уже рассмотренных ранее.

Соответствующий фрагмент программы:

```
| Этап 1
i := 1
нц пока <заданное условие не соблюдается>
  i := i + 1
кц
| Первый элемент массива, удовлетворяющий заданному условию, найден.
| Его индекс равен i
| Этап 2
```

```
мин := a[i]
| Этап 3
нц для j от i + 1 до n
  если <условие>
    то
     если a[j] < мин
       то
         мин := a[j]
     все
  все
кц
| Выводим ответ
вывод нс, мин
```

Язык Паскаль

```
{ Этап 1 }
i := 1;
while <заданное условие
не соблюдается> do
  i := i + 1;
{ Этап 2 }
min := a[i];
{ Этап 3 }
for j := i + 1 to n do
  if <условие> then
     if a[j] < min then min:= a[j];
write(min);
```

Язык Python

```
# Этап 1
i = 0
while <заданное условие
не соблюдается>:
  i = i + 1
# Этап 2
min = a[i]
# Этап 3
for j in range(i + 1, n):
  if <условие>:
    if a[j] < min:
      min = a[j]
print(min)
```

Если же допускается, что чисел, удовлетворяющих заданному условию, в исследуемом массиве может не быть, в программе необходимо учесть возможность выхода за пределы массива. Основные этапы решения задачи в таком случае:

1. Найти (попробовать найти) первое значение в массиве, удовлетворяющее заданному условию:

```
i := 1
нц пока i < n и <заданное условие не соблюдается>
  i := i + 1
кц
```

2. Проверить, есть ли такое значение в массиве:

```
если <заданное условие соблюдается>
  то
    | Первое (или единственное) значение в массиве есть
    | Его индекс равен i
    | Принимаем его в качестве минимального
    мин := a[i]
    | Рассматриваем оставшиеся элементы и те из них,
    | которые удовлетворяют заданному условию,
    | сравниваем с минимальным среди уже рассмотренных
    нц для j от i + 1 до n
      если <условие>
        то
          если a[j] < мин
            то
              мин := a[j]
          все
      все
    кц
    | Выводим ответ
    вывод нс, мин
  иначе | Элементов с заданными свойствами в массиве нет
    вывод нс, "Таких чисел в массиве нет»
все
```

Язык Паскаль

```
i := 1;
while (i < n) and (<заданное
условие не соблюдается>) do
  i := i + 1;
if <заданное условие соблюдается>
then
  begin
    min := a[i];
    for j := i + 1 to n do
      if <условие> then
        if a[j] < min then min
                      := a[j];
    write(min)
  end
else
  write('Таких чисел в массиве
нет');
```

Язык Python

```
i = 0
while i < n - 1 and <заданное
условие не соблюдается>:
  i = i + 1
if <заданное условие
соблюдается>:
  min = a[i]
  for j in range(i + 1, n):
    if <условие>:
      if a[j] < min:
        min = a[j]
  print(min)
else:
  print("Таких чисел в массиве
нет")
```

Возможен также более компактный вариант решения:

```
индекс_мин := 0 | Условно
| Рассматриваем все элементы
нц для i от 1 до n
  если <условие>
    то
     | Встретился элемент массива,
     | удовлетворяющий заданному условию.
     | Если это произошло впервые или он меньше "старого» минимума
     если индекс_мин = 0 или a[i] < a[индекс_мин]
       то
        | Запоминаем его индекс в качестве значения индекс_мин
        индекс_мин := i
     все
  все
кц
| Выводим ответ
если индекс_мин > 0
  то
    вывод нс, a[индекс_мин]
  иначе
    вывод нс, "Таких чисел в массиве нет»
все
```

В его особенностях разберитесь самостоятельно.

Язык Паскаль

```
index_min := 0;
for i := 1 to n do
  if <условие> then
    if (index_min = 0) or
      (a[i] < a[index_min] then
        index_min := i;
if index_min > 0
  then
    write(a[index_min])
  else
    write('Таких чисел в массиве
нет');
```

Язык Python

```
index_min = -1
for i in range(n):
  if <условие>:
    if index_min == -1 or
          a[i] < a[index_min]:
      index_min = i
if index_min > -1:
  print(a[index_min])
else:
  print("Таких чисел в массиве
нет")
```

1.4.8. Определение индекса минимального элемента среди элементов массива, которые удовлетворяют некоторому условию

Задача решается аналогично задаче 1.4.4 с учетом имеющихся здесь особенностей (см. также предыдущую задачу).

Задачи типа 1.4.7 и 1.4.8 применительно к максимальному элементу решаются аналогично.

1.4.9. Нахождения второго по величине максимального элемента

Данная задача допускает два толкования. Если рассматривать, например, массив 5 10 22 6 22 20 6 12, то каким должен быть ответ?

Под «вторым по величине максимальным элементом», или, короче, «вторым максимумом», можно понимать:

1) значение элемента массива, который стоял бы на предпоследнем месте, если бы массив был отсортирован по неубыванию. При таком толковании – 22;
2) значение элемента массива, *больше* которого только максимальный. В этом случае ответ – 20.

Если в массиве только один максимальный элемент (все остальные меньше), то оба толкования совпадают, и искомые значения будут одними и теми же, в противном случае – нет.

Обсудим оба варианта задачи. В каждом из них будем использовать две переменные:

1) максимум1 – максимальный элемент массива (первый максимум);
2) максимум2 – второй максимум (искомое значение).

Обе задачи решим за однократный просмотр массива.

1.4.9.1. Поиск элемента массива, который стоял бы на предпоследнем месте, если бы массив был отсортирован по неубыванию

Сначала рассмотрим вариант, в котором диапазон значений элементов массива известен.

В качестве начальных значений величин макс1 и макс2 (см. выше) принимаем число, которое заведомо меньше нижней границы диапазона значений элементов массива (например, при диапазоне от –1000 до 1000 – число –1001):

```
макс1 := -1001
макс2 := -1001
```

Строго говоря, значение макс2 можно не задавать, что будет показано далее.

Затем рассматриваем все элементы, сравнивая их сначала со значением макс1, а затем (при необходимости) – и со значением макс2:

```
нц для i от 1 до n
  если a[i] > макс1
    | Встретился элемент, больший, чем макс1
    то
      | Бывший первый максимум станет вторым
      макс2 := макс1
      | Первым максимумом станет встреченный элемент
      макс1 := a[i]
      | Внимание - именно в таком порядке!
    иначе
      | Очередной элемент не больше макс1
      | Сравниваем его со значением макс2
      если a[i] > макс2
        | Встретился элемент, больший макс2
        то
          | Принимаем его в качестве нового значения макс2
          макс2 := a[i]
          | Значение макс1 не меняется
      все
  все
кц
```

Нетрудно убедиться, что уже после рассмотрения первого элемента произойдет «переприсваивание»:

```
макс2 := макс1
```

т. е. начальное значение величины макс2, равное –1001, можно не задавать.

Язык Паскаль

```
max1 := -1001;
for i := 1 to n do
  if a[i] > max1
    then begin
```

Язык Python

```
max1 = -1001
for i in range(n):
  if a[i] > max1:
    max2 = max1
```

Язык Паскаль

```
            max2 := max1;
            max1 := a[i];
          end
      else
        if a[i] > max2
          then
            max2 := a[i];
```

Язык Python

```
    max1 = a[i]
  else:
    if a[i] > max2:
      max2 = a[i]
```

Если же диапазон значений элементов массива неизвестен, то предварительно нужно определить начальные значения величин макс1 и макс2, сравнив первый и второй элементы массива:

```
если a[1] > a[2]
  то
    макс1 := a[1]
    макс2 := a[2]
  иначе
    макс1 := a[2]
    макс2 := a[1]
все
```

или короче:

```
макс1 := a[1]
макс2 := a[2]
если a[2] > a[1]
  то
    макс1 := a[2]
    макс2 := a[1]
все
```

Затем рассматриваем остальные элементы, как и ранее, сравнивая их сначала со значением макс1, а затем (при необходимости) – и со значением макс2:

```
нц для i от 3 до n
  если a[i] > макс1
    ...
```

Язык Паскаль

```
max1 := a[1];
max2 := a[2];
if a[2] > a[1] then
  begin
    max1 := a[2];
    max2 := a[1]
  end;
for i := 3 to n do
  ...
```

Язык Python

```
max1 = a[0]
max2 = a[1]
if a[1] > a[0]:
  max1 = a[1]
  max2 = a[0]
for i in range(2, n):
  ...
```

1.4.9.2. Нахождение элемента массива, больше которого только максимальный

Примем, что диапазон значений элементов массива известен.

В качестве начального значения величины макс1[1] принимаем число, которое заведомо меньше нижней границы диапазона значений элементов массива (например, при диапазоне от –1000 до 1000 – число –1001):

```
макс1 := -1001
```

Рассматриваем все элементы массива и проверяем каждое из них сначала на первый, а затем на второй максимум:

```
нц для i от 1 до n
  если a[i] > макс1
    | Встретился элемент, больший, чем макс1
    то
      | Бывший первый максимум станет вторым
      макс2 := макс1
      | Первым максимумом станет встреченный элемент
      макс1 := a[i]
    иначе
      | Очередной элемент не больше макс1
      | В этом случае сравниваем его со значением макс2
      если a[i] < макс1 и a[i] > макс2
        | Только в этом случае!
        то
          | Встретился элемент, меньший макс1 и больший макс2
          | Принимаем его в качестве нового значения макс2
```

[1] Для величины макс2 начальное значение, как и для первого варианта, можно не задавать.

```
            макс2 := а[i]
            | Значение макс1 не меняется
        все
    все
кц
```

Фрагмент программы, относящийся к выводу ответа, оформляется так:

```
если макс2 = -1001
  | Второй максимум не встретился
  то
    вывод нс, "Нет такого значения в массиве "
  иначе
    вывод нс, "Второй максимум равен ", макс2
все
```

Язык Паскаль

```
maximum1 := -1001;
for i := 1 to n do
  if a[i] > max1
    then begin
           maxi2 := max1;
           max1 := a[i]
         end
    else
      if (a[i] < max1) and
(a[i] > max2)
        then
          max2 := a[i];
if max2 = -1001 then
  writeln('Нет такого значения
в массиве')
else
  writeln('Второй максимум равен
', max2)
```

Язык Python

```
maximum1 = -1001
for i in range(n):
  if a[i] > max1:
    maxi2 = max1
    max1 = a[i]
  else:
    if a[i] < max1 and a[i] >
max2:
      max2 = a[i]
if max2 = -1001:
  print("Нет такого значения
        в массиве")
else:
  print("Второй максимум равен",
        max2)
```

Ясно, что второго максимума в принятом толковании может не быть только тогда, когда все элементы массива равны.

1.4.10. Нахождение второго минимума

Все варианты этой задачи решаются аналогично предыдущей (конечно, с необходимыми изменениями).

Задания для самостоятельной работы

1. В массиве хранится информация о стоимости 1 кг 20 видов конфет. Определить, сколько стоят самые дешевые конфеты.

2. Известны расстояния от Москвы до нескольких городов. Найти расстояние от Москвы до самого удаленного от нее города из представленных в списке городов.

3. Известны результаты каждого из участников соревнований по лыжным гонкам (время, затраченное на прохождение дистанции гонки). Спортсмены стартовали по одному. Результаты даны в том порядке, в каком спортсмены стартовали. Определить, каким по порядку стартовал лыжник, показавший лучший результат? Если таких спортсменов несколько, то должен быть найден первый из них.

4. В массиве хранится информация о количестве осадков, выпавших за каждый день июля. Определить дату самого дождливого дня. Если таких дней было несколько, то должна быть найдена дата:

а) первого из них;

б) последнего из них.

5. Известна информация о максимальной скорости каждой из 12 марок легковых автомобилей. Определить скорость автомобиля, больше которой только максимальное значение в массиве.

6. В массиве хранится информация о результатах 22 спортсменов, участвовавших в соревнованиях по бегу на 100 м. Определить результаты спортсменов, занявших первое и второе места. Задачу решить, не используя два прохода по массиву.

7. В одном массиве записаны названия 20 команд – участниц чемпионата по футболу, в другом – соответствующее им количество набранных очков. Определить команды, занявшие первое и второе место. Задачу решить, не используя два прохода по массиву.

8. Известна информация о среднедневной температуре за каждый день июля. Определить даты двух самых холодных дней.

9. Известна информация о баллах, набранных каждым из 25 учеников на ЕГЭ по информатике. Определить, сколько учеников набрали максимальную сумму баллов.

10. В массиве записана информация о весе каждого из 30 человек (в кг). Вывести на экран:

1) в первой строке – порядковые номера людей, которые имеют максимальный вес среди представленных;

2) во второй строке – их количество.

В заключение заметим следующее. Хотя методики решения задач, рассмотренных в разделах 1.2 (кроме задачи 1.2.1), 1.3 и 1.4, описаны применительно к массивам, они (методики) применимы также для обработки последовательности чисел:

- задаваемых в ходе выполнения программы (вместо i-го элемента массива нужно рассматривать очередное число последовательности, которое обрабатывается сразу после его ввода в программу без сохранения в массиве);
- читаемых из файла (см. раздел 2.11);
- получаемых в программе в результате расчетов.

1.5. Работа с данными строкового типа

1.5.1. Общие вопросы

Одним из основных типов данных в языках программирования является строковый тип. К нему относятся величины, значением которых является последовательность символов. Это может быть конкретная строка символов, которая указывается в кавычках (одинарных или двойных, в зависимости от языка программирования, – `'карандаш'`, `"Дмитрий"`, `'h'`, `''` и т. п.[1]), переменная (`s`, `фам` и т. п.) или выражение, результатом которого является последовательность символов (`имя + " " + фамил` и т. п.). В устной речи и при письме, как правило, этот тип данных называют «строковый тип», а сами данные – «строки».

Способы задания значений величинам строкового типа:

- с помощью оператора ввода данных:

```
вывод нс, "Введите строку символов "
ввод стр
```

- с помощью оператора присваивания:

```
имя := "Дмитрий"
```

В правой части оператора присваивания можно указывать также имя переменной величины строкового типа или имя функции (метода) для работы со строками. Примеры:

---

[1] Если строка ограничена одинарными кавычками, внутри нее могут быть двойные кавычки:

```
s2 := 'Я болельщик клуба "Спартак"'
```

и наоборот.

```
клуб := "Спартак"
мой_клуб := клуб
```

Можно также указать несколько таких значений, между которыми записывается знак «+»:

```
кто := имя + фамил
об_мне := "Я - болельщик команды " + клуб
текст := "Меня зовут" + " " + имя
```

В данном случае операция со знаком «+» называется «конкатенация».

Другие операции[1] над строками выполнять нельзя.

Возможно также присваивание значений величинам строкового типа после чтения их (значений) из файла (см. раздел 2.11).

При хранении значения величины строкового типа в памяти компьютера символы строки расположены подряд (в соседних ячейках). Модель памяти при этом можно представить в виде, как на рис. 1.5.1.

	Д	м	и	т	р	и	й
Номер символа	*1*	*2*	*3*	*4*	*5*	*6*	*7*

имя

Рис. 1.5.1

В языке программирования Python и ряде других нумерация отдельных символов в памяти начинается с нуля. Здесь же напомним, что каждый символ в памяти компьютера хранится в виде двоичного числа – его кода.

Строки можно сравнивать между собой, и не только на предмет их равенства или неравенства:

```
если derevo > "дуб"
  то …
```

Вот несколько примеров сравнения строк (во всех случаях результат является истинным):

```
"a123" > "a"
"a2" > "a123"
```

[1] В некоторых языках программирования возможна также операция «умножения» величины строкового типа на целое число.

```
"а123" > "аб"
"аб" > "б"
"б" > "б2"
"б2" > "б2д"
"б2д" > "бб"
"б" > "а"
```

Если выписать перечисленные строки в виде:

```
"а"
"а123"
"а2"
"аб"
"б"
"б2"
"б2д"
"бб"
```

– то можно установить правило, по которому происходит сравнение двух строк: посимвольно слева направо сравниваются коды соответствующих символов до тех пор, пока не нарушится равенство кодов или не закончится одна из строк (или обе сразу), при этом сразу делается вывод о неравенстве и определяется, какая из строк меньше, а какая – больше. Две величины строкового типа являются равными, если они равны по длине и совпадают посимвольно.

Именно по такому правилу происходит сравнение строк при их сортировке. Порядок, в котором окажутся строки в результате сортировки, называют алфавитным (так расположены, например, слова в словарях).

Но как компьютер «знает», что такое алфавитный порядок? И как сравнивать слова, в которых есть и строчные, и прописные буквы, а также цифры и другие символы? Оказывается, при сравнении строк используются коды символов.

В кодовой таблице[1] прописные буквы стоят раньше строчных и поэтому имеют меньшие коды. Цифры стоят в кодовой таблице по порядку, причем раньше, чем латинские буквы; латинские буквы – раньше, чем русские; заглавные буквы (русские и латинские) – раньше, чем соответствующие строчные.

[1] Кодовая таблица – таблица, сопоставляющая каждому символу некоторое двоичное число, его код.

СОВЕТ

Указанные особенности (цифры и буквы алфавита стоят в кодовой таблице по порядку) позволяют вместо многократного сравнения переменной величины со всеми буквами или цифрами использовать сложное условие со знаками сравнения.

Пример задачи: «Дан символ. Определить, является ли он прописной латинской буквой».

Решение

Вместо такого громоздкого варианта:

```
ввод симв
если симв = "A" или симв = "B» или симв = "C" или … или симв =
"Z"
  то
    вывод нс, "Да, является"
  иначе
    вывод нс, "Нет, не является"
все
```

можно использовать следующий:

```
ввод симв
если симв >= "A" и симв <= "Z"
  то
    …
```

Величина строкового типа рассматривается как массив, элементами которого являются отдельные символы. Это значит, что, для того чтобы использовать в программе значение отдельного символа строки, нужно указать имя строковой величины и в скобках (круглых или квадратных, в зависимости от языка программирования) – номер символа в памяти. Примеры на школьном алгоритмическом языке:

```
вывод стр[5]
вывод стр[k]
если стр[к + 1] = "Д"
  то
    …
```

В любой момент выполнения программы длину строки (количество символов в ней) можно определить с помощью стан-

дартной функции. В школьном алгоритмическом языке имя этой функции – длин. Например, так в переменную всего записывается длина строки стр:

```
всего := длин(стр)
```

Эта функция используется, например, когда в операторе цикла с параметром требуется рассмотреть все символы величины строкового типа.

Аналогичные функции в языках Python и Паскаль:

Язык	Имя функции
Python	`len`
Паскаль	`length`

Во всех языках программирования имеется возможность получить так называемый срез строки – некоторую последовательность ее символов.

В школьном алгоритмическом языке для получения среза нужно после имени величины строкового типа указать в квадратных скобках номера первого и последнего символа требуемой части строки через двоеточие.

```
слово := "информатика"
срез := слово[3 : 7]
вывод срез
```

В языке Python в срезе в квадратных скобках указываются:

- номер первого символа требуемой части строки в памяти компьютера;
- число, на 1 большее номера последнего символа требуемой части строки в памяти компьютера.

Пример. Программа для получения слова "menu" (6–9-й символы строки "File menu"):

```
s = "File menu"
s2 = s[5:9]  # Срез строки s
print(s2)
```

В программах на языке Паскаль срез строки возвращает функция Copy. Ее общий вид:

```
Copy(<строка>, <номер начального символа среза>, <количество
символов>
```

В языке Паскаль можно также по-особому получить срез из нескольких первых или нескольких последних символов величины строкового типа. Это можно сделать, соответственно, с помощью функций `LeftStr` и `RightStr`. Первая возвращает первые `k` символов строки `st`, вторая – последние `k` символов:

```
LeftStr(st, k)
RightStr(st, k)
```

Выше уже отмечалось, что в памяти компьютера каждый символ хранится в виде двоичного числа – его кода. Десятичный эквивалент кода символа можно получить с помощью функций, указанных в таблице:

Язык программирования	Имя функции
Школьный алгоритмический язык	`код`
Паскаль	`ord`
Python	`ord`

Символ, код которого возвращается, указывается в кавычках в качестве аргумента функции (в скобках).

Имеются и другие функции и процедуры для работы со строками (их частями, в том числе с отдельными символами). О некоторых из них вы можете узнать ниже, а об остальных – из других источников.

Задания для самостоятельной работы

1. Дано слово. Вывести его буквы в столбик.
2. Даны два слова. Определить, какое из них в словаре, в котором слова расположены в алфавитном порядке, будет записано раньше.
3. Даны две величины строкового типа. Определить, в какой из них больше символов или количество символов в них одно и то же.
4. Дан символ. Определить, является ли он цифрой.
5. Даны фамилия, имя и отчество человека. Сформировать из них:
   а) одну общую величину;
   б) величину в виде: «фамилия и инициалы» (например, так: «Иванов Н. А.»).
6. Даны два слова. Получить из них общую величину, в которой первое слово будет записано прописными буквами, а второе (после пробела) – строчными.
7. Дано слово из четного количества букв. Определить, одинаковы ли его первая и вторая половины.

8. Дана строка. Найти сумму десятичных кодов ее первого и последнего символов.

9. Дано слово. Проверить, является ли оно палиндромом (палиндромом называется слово, которое одинаково читается как слева направо, так и справа налево, например такими словами являются «потоп» и «дед»).

1.5.2. Обработка отдельных символов строк

Во всех рассмотренных далее задачах заданная строка имеет имя стр.

Задача 1. Определить, сколько раз в заданной строке встречается некоторый символ.

Задача решается аналогично задаче 1.2.3. Нужно сравнить с заданным символом (его имя в программе – симв) каждый символ строки и в случае их равенства увеличить значение переменной-счетчика на 1:

```
...
кол := 0                          | Переменная-счетчик
нц для к от 1 до длин(стр)        | Рассматриваем все номера символов
                                  | строки
  если стр[к] = симв              | Сравниваем соответствующий символ
    то                            | с заданным
      кол := кол + 1
  все
кц
| Выводим ответ
если кол > 0
  то
    вывод нс, "Заданный символ встречается в строке ", кол, " раз"
  иначе
    вывод нс, "Такого символа нет"
все
```

В программе на языке Python можно также рассматривать не номера символов, а сами символы:

```
kol = 0
for c in str:  # Рассматриваем все символы с строки str
    if c == simv:
        kol = kol + 1
...
```

В п. 1.2.3 говорилось об имеющейся в языках программирования Паскаль и Python функции count. Она может быть применена

и к величинам строкового типа (с учетом указанных в п. 1.2.3 особенностей).

Задача 2. Определить позицию (номер) первого вхождения некоторого символа в заданную строку.

Сначала обсудим вариант задачи, в котором заведомо известно, что заданный символ в строке имеется.

Так же, как и задача 1.3.5, данная задача может быть решена путем использования оператора цикла с параметром и переменной-флага :

```
встретился_первый := нет
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то | Проверяем, это первый символ?
      если встретился_первый = нет  | Или   если не встретился_первый
        то
          искомый_номер := к
          встретился_первый := да
    все
  все
кц
вывод нс, искомый_номер
```

или (без проверки всех символов строки):

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то
      искомый_номер := к
      | Прекращаем обработку строки
      выход
  все
кц
вывод нс, искомый_номер
```

Если же допускается, что искомого символа в строке может не быть, то задача решается так же, как решалась аналогичная задача применительно к массиву (см. п. 1.3.5). Разработайте соответствующую программу самостоятельно.

В языке Python существует функция для поиска позиции (номера символа), с которой начинается первое вхождение некоторой последовательности символов (и отдельного символа) в строке. Эта функция называется `find()`. В скобках нужно указать образец для поиска:

```
str = input("Введите строку ")
simv = input("Введите символ ")
ind = str.find(simv)        | Функция find() вызывается как метод
```

Эта функция возвращает номер заданного символа в памяти компьютера; если заданного символа в строке нет – возвращается значение –1. Поэтому вывод ответа оформляется следующим образом:

```
if ind != -1:
  print(ind + 1)
else:
  print("Такого символа в строке нет")
```

Например, если `str = "Информатика"`, а `simv == "а"`, то будет выведен ответ 7 (номер заданного символа в памяти компьютера, увеличенный на 1; такой ответ в данном случае более привычен пользователю программы).

Имя аналогичной функции в языке Паскаль – `Pos`. Ее первый аргумент – искомая последовательность символов, второй – строка, в которой происходит поиск. Если заданной последовательности в строке нет – возвращается значение 0.

Обратим внимание на то, что во всех случаях строчные и прописные буквы при поиске отличаются.

Задача 3. Определить, есть ли в заданной строке некоторый символ `симв`.

Прежде всего заметим, что такой вариант решения:

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то
      вывод "Заданный символ в строке есть"
    иначе
      вывод "В строке нет заданного символа"
  все
кц
```

– ошибочный – на экран ответ будет выводиться многократно.

Неправильный результат дает также такой способ, использующий переменную-флаг `есть`, которая фиксирует факт наличия заданного символа в заданной строке:

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
```

```
    то
      есть := да
    иначе
      есть := нет
  все
кц
если есть  | Или          если есть = да
  то
    вывод "Заданный символ в строке есть"
  иначе
    вывод "В строке нет заданного символа"
все
```

Его особенности проанализируйте самостоятельно.

Наиболее простой правильный способ решения задачи такой:

1) подсчитать число вхождений заданного символа в заданную строку (см. задачу 1.5.1);
2) по найденному значению получить ответ.

Можно также не рассматривать всю строку, а прекратить проверки, как только встретится заданный символ:

```
к := 1
найден := нет
нц пока к <= длин(стр) и не найден
  если стр[к] = симв
    то
      найден := да
    иначе | Переходим к следующему символу
      к := к + 1
  все
кц
если найден
  то
    вывод нс, "Заданный символ в строке есть"
  иначе
    вывод нс, "В строке нет заданного символа"
все
```

Можно также для прекращения проверок элементов выйти из тела оператора цикла с помощью соответствующей процедуры (`break` или подобной). В программе на языках Паскаль и Python задача также решается просто благодаря использованию функций, соответственно, `find` и `Pos` (в них предусмотрен вывод результатов поиска при отсутствии в строке заданного символа).

В программе на Python лучше всего использовать оператор проверки принадлежности in, который проверяет, имеется ли символ в заданной строке:

```
if simv in str:
  print("Заданный символ в строке есть")
else:
  print("В строке нет заданного символа")
```

Задача 4. Определить, является ли символ строки с заданным номером цифрой.

Задача может быть решена разными способами.

Во-первых, можно сравнить заданный символ с каждой цифрой:

```
| к - заданный номер символа
если стр[к] = "0" или стр[к] = "1" или … или стр[к] = "9"
  то
    вывод "Да, заданный символ - цифра"
  иначе
    вывод "Нет, заданный символ цифрой не является"
все
```

Во-вторых, можно учесть, что коды символов-цифр расположены в кодовой таблице по порядку. Их десятичные эквиваленты: 48, 49... 57:

```
если код(стр[к]) >= 48 и код(стр[к]) <= 57
  то
```

СОВЕТ

Если вы забудете коды цифр, то можно вместо конкретных значений кода нуля и цифры 6 использовать функцию, возвращающую эти коды:

```
если код(стр[к]) >= код("0") и код(стр[к]) <= код("9")
  то
```

В языке Python код цифры 0 можно определить в интерактивном режиме или в программе так:

```
print(ord("0"))
```

В языке Python возможны два оригинальных способа решения задачи. Один из них заключается в использовании строки vse со всеми цифрами и оператора проверки принадлежности in:

```
vse = "0123456789"
if стр[k] in vse:
  print("Да …")
else:
  print("Нет …")
```

А второй, дающий самое краткое решение, использует метод isdigit(). Этот метод проверяет, состоит ли строка из цифр, и возвращает в качестве результата логическое значение (True или False):

```
if str[k].isdigit():   # В данном случае
                       # проверяется строка из одного символа
  print("Да …")
else:
  print("Нет …")
```

Задача для самостоятельной работы

Определить, есть ли в заданной строке цифры.

1.5.3. Обработка подстрок

Термином «подстрока» условимся называть заданную последовательность символов, представляющую часть строки (используется также термин «цепочка символов»).

Во всех рассмотренных далее задачах заданная строка имеет имя стр, а подстрока – подстр.

Задача 1. Определение длины (количество элементов) подстроки.

Пример: «Дана строка символов, среди которых есть несколько подряд идущих букв "я". Определить их количество».

Решение

Можно рассуждать так. Если очередной символ строки обладает заданными свойствами (является нужной буквой, цифрой или т. п.),

то подстрока таких символов продолжается или началась – увеличиваем ее длину на 1. Как и ранее, имя этой переменной – кол.

```
алг
нач лит стр, цел кол, к
  вывод нс, "Введите строку символов"
  ввод стр
  кол := 0
  нц для к от 1 до длин(стр)
    если стр[к] = "я"
      то
        кол := кол + 1
    все
  кц
  вывод нс, "Длина подстроки из букв "я" равна", кол
кон
```

где длин – функция школьного алгоритмического языка, возвращающая длину строки (количество символов) своего аргумента.

Возможны также задачи рассматриваемого типа, в которых проверка символов подстроки начинается со второго символа.

Пример: «Дана строка символов, в начале которой есть несколько подряд идущих цифр. Определить их количество».

Решение

Здесь начальное значение переменной кол равно 1 (учитываем первый символ-цифру), а обработка символов строки должна начинаться со второго символа:

```
кол := 1
нц для к от 2 до длин(стр)
  если …                | стр[к] - цифра (см. выше)
   то
      кол := кол + 1
```

Задача 2. Определить, сколько раз в заданной строке встречается некоторая подстрока.

Решение задачи во многом аналогично решению задачи 1 в п. 1.5.2. Единственное отличие в том, что проверять надо не по одному символу, а по несколько. При анализе нам понадобится получать и сравнивать с заданной подстрокой несколько последовательных символов строки – срез (см. п. 1.5.1). При этом возникает вопрос – чему равен номер последнего символа строки, начиная с которого следует формировать срез?

Обозначим количество символов в заданной строке – длина1, в подстроке – длина2. Рассмотрим несколько возможных значений:

длина1	длина2	Номер последнего символа строки для формирования среза
10	2	9
10	3	8
15	5	11
15	6	10

Из таблицы можно установить, что номер последнего символа строки для формирования среза определяется по формуле: `длина1 - длина2 + 1`.

Итак, фрагмент программы решения задачи:

```
длина1 := длин(стр)
длина2 := длин(подстр)
кол := 0
нц для к от 1 до длина1 - длина2 + 1
  если стр[к : (к + длина2 - 1)] = подстр  | Срез совпадает с подстрокой
   то
     кол := кол + 1
  все
кц
```

В онлайн-тренажере для подготовки к ЕГЭ по информатике в компьютерной форме [12] предлагаются две похожих задачи:

1) задача, в которой нужно определить количество вхождений в заданную строку двух подстрок;
2) задача: «Дана строка символов. Определить, сколько раз встречаются комбинации “RISS”, при этом до и после этого слова нет буквы “R”. Например, комбинации “RRISS”, “RISSR” и “RRISSR” не должны учитываться».

Первая задача решается аналогично рассмотренной. Искомое количество можно найти как сумму числа вхождения каждой отдельной подстроки. Можно также использовать в программе сложное условие.

Обсудим решение второй задачи.

Анализ показывает, что для «участка» строки от первого символа до предпоследнего:

T	R	I	S	S	R	O	R	I	S	S	O	R	R	I	S	S	R	I	S	S	N	N	R	R	I	S	S	R	A

– количество вхождений с учетом ограничений можно найти так, как это делалось в рассмотренной выше задаче (используя при этом сложное условие для проверки также символов слева и справа от среза):

```
...
нц для ном от 2 до длин(строка) - 4
  если строка[ном : ном + 3] = "RISS" и строка[ном : ном + 4] <> "R"
       и строка[ном - 1] <> "R"
    то

      кол := кол + 1
  все
кц
```

Но при этом не будут проверены первая и последняя подстроки из четырех символов:

R	I	S	S	R	T	O	R	I	S	S	O	R	R	I	S	S	R	I	S	S	N	N	R	E	R	I	S	S

Их надо проверить в программе дополнительно:

```
если строка[1 : 4] = "RISS" и строка[5] <> "R"
   то
     кол := кол + 1
 все
 если строка[длин(строка) - 3 : длин(строка)] = "RISS" и
      строка[длин(строка) - 4] <> "R"
   то
     кол := кол + 1
 все
```

Задача 3. Определить позицию (номер) первого вхождения некоторой подстроки в заданную строку.

Если заведомо известно, что заданная подстрока в строке имеется, то задача решается аналогично задаче 2 в п. 1.5.2:

```
...
длина1 := длин(стр)
длина2 := длин(подстр)
встретилась_первая := нет
нц для к от 1 до длина1 - длина2 + 1
  если стр[к : (к + длина2 - 1)] = подстр
    то | Проверяем, это первый срез, совпадающий с подстрокой?
```

```
      если встретилась_первая = нет  | или если не встретилась_первая
        то
          искомый_номер := к
          встретилась_первая := да
    все
  все
кц
вывод нс, искомый_номер
```

или (без проверки всех символов строки):

```
...
длина1 := длин(стр)
длина2 := длин(подстр )
нц для к от 1 до длина1 - длина2 + 1
  если стр[к : (к + длина2 - 1)] = подстр
    то
      искомый_номер := к
      | Прекращаем обработку строки
      выход
  все
кц
вывод нс, искомый_номер
```

Если же допускается, что искомой подстроки в строке может не быть, то задача решается аналогично тому, как решалась подобная задача применительно к массиву (см. п. 1.3.5). Разработайте соответствующую программу самостоятельно.

В программе на языках Паскаль и Python обсуждаемая задача также решается просто благодаря использованию функций `find` и `Pos` соответственно.

Задача 4. Определить, есть ли в заданной строке некоторая подстрока.

Различные способы решения данной задачи аналогичны описанным применительно к отдельному символу строки.

Задача 5. Удалить из заданной строки все вхождения некоторой подстроки.

В программе на школьном алгоритмическом языке удалить из строки подстроку можно, присвоив срезу строки (см. выше) «пустое» значение, например для подстроки из четырех символов, начиная с 7-го:

```
стр[7:10] := ""
```

Еще две важные особенности программы решения задачи.

1. Поскольку в ходе обработки строки ее длина меняется, использовать оператор цикла с параметром, в конечном значении которого применяется величина длина1 (количество символов в заданной строке), нельзя. Следует применить оператор цикла с условием следующего вида:

```
к <= длин(стр) - длина2 + 1
```

где длина2 – количество символов в удаляемой подстроке.

2. При рассмотрении к начальных номеров строки после нахождения искомой подстроки и ее удаления величина к не меняется (убедитесь в этом!), а в случае, когда искомая подстрока не встретилась, увеличивается на 1.

С учетом сказанного фрагмент программы, решающей задачу, имеет вид:

```
к := 1
нц пока к <= длин(стр) - длина2 + 1
 если стр[к : (к + длина2 - 1)] = подстр
   то | Встретилась искомая подстрока
      | Удаляем ее
      стр[к : (к + длина2 - 1)] := ""
      | Величина к - не меняется
   иначе
    | Переходим к следующему символу для проверки
    к := к + 1
 все
кц
```

В программе решения задачи на языке Python возникает проблема, связанная с тем, что в этом языке величина строкового типа является так называемым неизменяемым объектом. Имеется в виду, что нельзя изменить отдельный символ строки, удалить отдельный символ (или подстроку)[1] и что-то вставить в строку[2]. Но все эти операции можно провести, разработав специальные программы. Например, для удаления части строки str нужно составить новую строку str2, объединив в ней части исходной

[1] Можно удалить один или несколько начальных и/или конечных символов (пробелов и т. п.) с помощью стандартных методов lstrip(), rstrip() и strip().

[2] В то же время величины типа строкового типа можно «удлинять»:

```
st = "Волк"
st = st + "и"
```

строки до и после удаляемого участка (см. рис. 9.2.1). Нужные части можно получить с помощью среза:

```
str2 = str[0 : nach - 1] + str[kon : len(str)]
```

Примечание. Использованы номера nach и kon, соответствующие номерам символов в заданной строке (напомним, что в языке программирования Python нумерация отдельных символов в памяти начинается с нуля).

В данном случае новой строке можно присвоить имя исходной строки:

```
str = str[0 : nach - 1] + str[kon : len(str)]
```

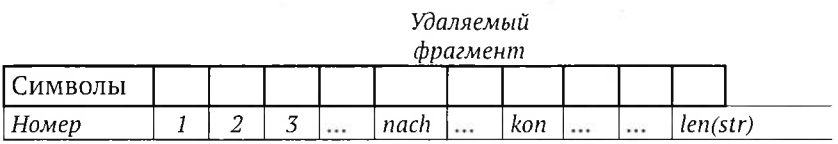


Рис. 9.2.1

Итак, как же удалить из строки все вхождения некоторой подстроки? Вспомним решение задачи 2. В ней многократно формировались срезы заданной строки некоторой длины, которые сравнивались с заданной подстрокой.

В нашей задаче вместо переменной k будем использовать переменную ykaz, которая будет хранить значение номера символа, начиная с которого формируется срез для сравнения с заданной подстрокой podstr. Будем называть эту переменную «указатель».

Схема действий следующая:

```
Повторение действий, начиная с 1-го символа (ykaz == 0):
  Если срез не совпадает с заданной подстрокой:
    Добавляем текущий символ в новую строку
    Смещаем указатель на 1 позицию вправо
  иначе:    # Встретилась искомая подстрока
    Пропускаем ее (смещаем указатель на len(podstr) позиций вправо)
```

Возникает вопрос: сколько раз надо повторить указанные действия? Неизвестно. Значит, оператор цикла с параметром применить нельзя. Но мы знаем (см. задачу 2 в п. 1.5.3), что последнее значение переменной ykaz равно `len(str) - len(podstr)`[1]. Значит, можем использовать оператор цикла с условием:

[1] С учетом особенностей нумерации символов в программе на языке Python.

```
str2 = ""    # Начальное значение новой строки
ykaz = 0     # Начальное значение указателя
while ykaz <= len(str) - len(podstr):
  # Сравниваем срез и заданную подстроку
  if str[ykaz : ykaz + len(podstr)] != podstr:
    # Добавляем текущий символ str[ykaz] в новую строку
    str2 = str2 + str[ykaz]
    # Смещаем указатель на 1 позицию вправо
    ykaz = ykaz + 1
  else:   # Встретилась искомая подстрока
    # Пропускаем ее
    # (смещаем указатель на len(podstr) позиций вправо)
    ykaz = ykaz + len(podstr)
```

После того как «перемещение» указателя прекратится, в конце заданной строки могут остаться символы. Необходимо их также добавить к значению str2:

```
# Добавляем оставшуюся часть (используя срез)
str2 = str2 + str[ykaz : len(str)]
```

Подход к решению задачи, описанный применительно к языку Python (с использованием новой строковой переменной), может быть использован и в программах на других языках программирования.

Отметим интересный момент. В школьном алгоритмическом языке и языке Паскаль имеется процедура, удаляющая из строки заданное количество символов, начиная с символа с некоторым номером:

```
удалить(<строка>, <начальный номер>, <количество символов>)
Delete(<строка>, <начальный номер>, <количество символов>)
```

Если вы попытаетесь применить эту процедуру для решения обсуждаемой задачи, то убедитесь, что использование стандартной процедуры (или функции) в ряде случаев значительно усложняет программу

Задание для самостоятельной работы

Разработайте программы решения обсуждаемой задачи:

а) по описанной методике;

б) с использованием процедуры удаления подстроки (при ее наличии в используемом языке программирования).

Задача 6. Заменить в заданной строке все вхождения некоторой подстроки на другую подстроку.

Используем в программе величины:

- стр – заданная строка;
- подстр1 – заменяемая подстрока;
- подстр2 – новая подстрока;
- длина1 – количество символов в величине подстр1;
- длина2 – то же в величине подстр2.

Приводя фрагмент программы решения задачи, обращаем внимание на условие в операторе цикла (см. предыдущую задачу), а также на особенности изменения величины к:

```
...
длина1 := длин(подстр1)
длина2 := длин(подстр2)
к := 1
нц пока к <= длин(стр) - длина1 + 1
 если стр[к : (к + длина1 - 1)] = подстр1
  то | Встретилась искомая подстрока
   | Заменяем ее
   стр[к : (к + длина1 - 1)] := подстр2
   | Изменяем значение величины к
   к := к + длина2
  иначе
    | Переходим к следующему символу для проверки
    к := к + 1
 все
кц
```

В школьном алгоритмическом языке имеется процедура, заменяющая в строке все вхождения заданной подстроки на другую подстроку. Аналогичные функции с именем replace в языках Паскаль и Python используются как метод.

Задание для самостоятельной работы

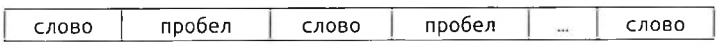
Разработайте программы решения обсуждаемой задачи:

а) по описанной методике;

б) с использованием процедуры замены подстроки (при ее наличии в используемом языке программирования).

1.5.4. Выделение слов предложения

В рассмотренных в данном разделе задачах обрабатывается задаваемое во время выполнения программы предложение пред, имеющее в общем случае следующую структуру:



Задача 1. Выделение первого слова.

Сначала заметим, что первые n символов заданной строки стр можно объединить в подстроку подстр, рассмотрев каждый из них с помощью оператора цикла с параметром и используя конкатенацию (см. п. 1.5.1):

```
подстр := ""  | Начальное значение формируемой подстроки
нц для к от 1 до n
  подстр := подстр + предл[к]
кц
```

В нашей же задаче количество символов для конкатенации заранее неизвестно. Но мы знаем, что после первого слова находится пробел. Значит, для решения задачи нужно проводить конкатенацию всех начальных символов предложения, отличающихся от пробела. Для таких повторяющихся действий следует применить оператор цикла с предусловием:

```
вывод нс, "Введите предложение "
ввод предл
слово1 := ""  | Формируемое слово (начальное значение)
к := 1        | Номер очередного символа строки (начальное значение)
нц пока предл[к] <> " " | Пока не встретится 1-й пробел
  | Добавляем текущий символ к "старому" значению величины слово1
  слово1 := слово1 + предл[к]
  | Переходим к следующему символу
  к := к + 1
кц
| Встретился пробел, то есть первое слово закончилось
| Выводим его
...
```

Можно также решить задачу так:

1) найти позицию (номер в строке) первого пробела. Это можно сделать, используя соответствующую функцию (см. задачу 2 в п. 1.5.2);

2) получить искомое слово, используя срез символов от первой буквы предложения до буквы с номером, на один меньшим найденного на этапе 1.

Соответствующую программу разработайте самостоятельно.

Задача 2. Выделение второго слова.

Для решения задачи сначала следует пропустить первое слово:

```
к := 1
нц пока предл[к] <> " "
  | Переходим к следующему символу
  к := к + 1
кц
| Величина к указывает на первый пробел
```

После этого можно сформировать второе слово:

```
...
слово2 := ""  | Формируемое слово (начальное значение)
| Переходим к 1-му символу 2-го слова
к := к + 1
нц пока строка[к] <> " "   | Пока не встретится 2-й пробел
  | Добавляем текущий символ к "старому" значению величины слово2
  слово2 := слово2 + строка[к]
  | Переходим к следующему символу
  к := к + 1
кц
...
```

Можно также решить задачу, определяя позиции первого и второго пробелов и используя срез. Для определения позиции второго пробела можно в функциях, возвращающих позицию первого вхождения подстроки, указать номер символа, начиная с которого проводить поиск. В школьном алгоритмическом языке поиск с символа с номером начало проводит функция поз после:

```
позиц_пробела := поз после(начало, " ", предл)
```

Поиск с определенной позиции возможен и в программе на языке Паскаль. Для этого в функции Pos следует указать номер символа, начиная с которого проводить поиск, в качестве третьего аргумента функции.

Соответствующий вариант программы разработайте самостоятельно.

Мы же обсудим частный случай задачи – выделение двух слов в предложении из двух слов[1].

[1] Такая задача возникает при выполнении заданий 26 и 27 (см. главу 3).

Здесь решение достаточно очевидно:

```
...
слово1 := ""
| Ищем позицию пробела между словами
к := 1
нц пока предл[к] <> " "
  слово1 := слово1 + предл[к]
  | Переходим к следующему символу
  к := к + 1
кц
| Величина к указывает на пробел
| Пропускаем его
к := к + 1
| Формируем второе слово:
слово2 := ""
нц для ном от к до длин(предл)
  слово2 := слово2 + предл[ном]
  к := к + 1
кц
```

Можно также после нахождения позиции пробела выделить первое и второе слова, используя срезы.

Задача 3. Выделение всех слов.

Рассмотрим сначала случай, когда количество слов в предложении известно и равно n. В этом случае надо выделить первые (n – 1) слов, после которых стоит пробел. Такие отдельные слова мы выделять научились (см. предыдущие задачи). Чтобы применить оператор цикла с параметром провести (n – 1) повторений, надо вместо имен величин слово1 и слово2 использовать одно имя – слово.

Начальная часть соответствующей программы, формирующая и печатающая первые (n – 1) слов с использованием конкатенации:

```
n := ...
вывод нс, "Введите предложение из ", n, " слов"
ввод предл
к := 1
нц для к от 1 до n - 1      | (n - 1)  раз
  | Формируем очередное слово
  слово := ""  | Начальное значение
  нц пока предл[к] <> " "
    | Пока символ - не пробел
    | Добавляем его к величине слово
```

```
    слово := слово + предл[к]
    | Переходим к следующему символу
    к := к + 1
  кц
  | Встретился пробел - слово закончилось
  | Печатаем его
  вывод слово, " "
  | Переходим к следующему символу - началу следующего слова
  к := к + 1
кц
```

Можно также выделять слова, используя срез, предварительно определяя позицию очередного пробела. Если последнюю переменную величину обозначить позиц, то фрагмент, в котором выделяются и выводятся на экран n - 1 слов предложения, оформляется так:

```
n := …
вывод нс, "Введите предложение из ", n, " слов"
ввод предл
начало := 1  | Начальный номер символа, начиная с которого
             | определяется позиция ближайшего пробела
нц для к от 1 до n - 1    | (n - 1)раз
  | Получаем очередное слово
  | Определяем позицию ближайшего пробела
  позиц := поз после(начало, " ", предл) | начиная с позиции начало
  | Определяем слово, используя срез
  слово := предл[начало : позиц - 1]
  вывод слово, " "
  | "Переходим" на первую букву следующего слова
  начало := позиц + 1
кц
```

Осталось получить и вывести на экран последнее слово. При использовании среза это можно сделать, учитывая, что мы знаем порядковый номер первой буквы последнего слова (начало) и номер последней буквы предложения:

```
…

| Последнее слово
слово := предл[начало + 1 : длин(предл)]
вывод слово
```

В программе с конкатенацией завершающая часть, в которой формируется и выводится последнее слово, будет такой («складываем» буквы от буквы с номером к до последней):

```
…
```

```
слово := ""  | Начальное значение
нц пока к <= длин(предл)
    слово := слово + предл[к]
    | Переходим к следующему символу
    к := к + 1
кц
| Закончилось последнее слово
| Печатаем его

вывод слово
```

Но, как правило, количество слов в предложении заранее неизвестно. Как решить задачу в таком случае? Ответ – при многократном выделении отдельных слов использовать только оператор цикла с предусловием.

В программе с конкатенацией условие, которое записывается в таком операторе, должно быть аналогичным тому, которое использовалось при выделении последнего слова (см. выше):

```
вывод нс, "Введите предложение "
ввод предл
к := 1
слово := ""
нц пока к <= длин(предл)
  | Формируем очередное слово
  если предл[к] <>  " "    | Если символ - не пробел
    то
      | Добавляем его к величине слово
      слово := слово + предл[к]
      | Переходим к следующему символу
      к := к + 1
      иначе | Встретился пробел - слово закончилось
            | Печатаем его
        вывод слово, " "
        | Готовимся формировать новое слово
        слово := ""
        | Переходим к следующему символу - началу следующего слова
        к := к + 1
  все
кц
| В конце работы оператора цикла было сформировано,
| но не напечатано последнее слово
| Печатаем его
вывод слово
```

В программе, использующей срез, рассуждения немного сложнее. Они приведены в комментариях:

```
вывод нс, "Введите предложение "
ввод предл
начало := 1
нц пока начало <= длин(предл)
    | Получаем очередное слово
    | Если есть ближайший пробел, начиная с позиции начало
    если поз после(начало, " ", предл) <> 0
       то
        | Определяем его номер
        позиц := поз после(начало, " ", предл)
        | Определяем очередное слово
        слово := предл[начало : позиц]
        | Печатаем его
        вывод слово, " "
        | "Переходим" на первую букву следующего слова
        начало := позиц + 1
      иначе | Больше пробелов нет - осталось последнее слово
            | Определяем его
        слово := предл[начало : длин(предл)]
        | и печатаем
        вывод слово
        | Меняем значение начало так,
        | чтобы оператор цикла больше не выполнялся
        начало := длин(предл) + 1
    все
кц
```

Какой вариант решения задачи лучше, решите сами.

Задача для самостоятельной работы

Дан набор слов, отделенных друг от друга запятой и пробелом. Вывести на экран все слова.

Дополнение

В практических задачах часто нужно преобразовать число, записанное в виде цепочки символов-цифр, в числовое значение и наоборот.

Начнем с первой задачи. Сразу же обратим внимание на то, что такая задача возникает, например, при чтении числового значения, записанного в текстовом файле (см. методику выполнения заданий 25–27 из [1] в разделах 3.13–3.15).

В школьном алгоритмическом языке предусмотрены две функции лит_в_цел и лит_в_вещ.

Общий вид первой из них:

```
лит_в_цел(<строка>, <успешно>)
```

Она переводит строку строка в целочисленное значение. Возвращаемый параметр успешно логического типа говорит о том, успешно ли произошло преобразование (ясно, что если в исходной строке есть, например, не только символы-цифры, то преобразования не произойдет).

Пример:

```
стр := "12345"
n := лит_в_цел(стр, усп)
```

Назначение и формат функции лит_в_вещ аналогичны.

Соответствующие функции в языке Python:

- int – переводит строку, указанную в скобках, в целое число;
- float – переводит строку в вещественное число.

Пример:

```
str = “666”
n = int(str)
```

Если строку не удалось преобразовать в число (например, если в ней содержатся буквы), возникает ошибка, и выполнение программы завершается.

Аналогичные функции в языке Паскаль: IntToStr и FloatToStr.

Функции, которые преобразовывают числовые величины в величины строкового типа (вторая задача, о которой шла речь в начале дополнения):

- в школьном алгоритмическом языке: цел_в_лит и вещ_в_лит;
- в языке Паскаль – IntToStr и FloatToStr.

В языке Python для перевода целого или вещественного числа в строку предназначена функция str.

Преобразование числа в строку позволяет просто решить многие задачи по обработке натурального (и не только) числа. Решение основано на том, что, как уже отмечалось, величина строкового типа рассматривается как массив, элементами которого являются отдельные символы.

В приведенных далее фрагментах программ, основанных на таком подходе, использованы следующие основные величины:

- n – заданное число;
- n_стр – его строковое представление;
- ном – номер символа в строковом представлении;
- m – номер цифры в числе (при счете справа налево или слева направо).

Д1. Определение количества цифр заданного натурального числа

```
ввод n
кол_цифр := длин(n)
| Вывод ответа
...
```

Без преобразования числа в строку нужно использовать оператор цикла с условием, операции определения остатка и др (см. раздел 1.1).

Д2. Выделение цифр

Задача формулируется так: «Дано натуральное число. Вывести его цифры в столбик».

Без преобразования числа в строку также следует использовать оператор цикла с условием, операции определения остатка и др. А вот как – с использованием такого преобразования:

```
ввод n
| Преобразовываем число в строку
n_стр := цел_в_лит(n)
| Выводим каждый символ строки
нц для ном от 1 до длин(n_стр)
  вывод нс, n_стр[ном]
кц
```

Д3. Определение m-й справа цифры натурального числа

Без преобразования числа в строку задачу можно решить так. Последовательно в цикле выделяем цифры, используя операции определения остатка и целочисленного деления, одновременно считаем выделенные цифры, используя переменную-счетчик. Когда значение счетчика станет равно m, выводим последнюю выделенную цифру. Соответствующий фрагмент:

```
ввод n
ввод m
к := 0                    | Переменная-счетчик
нц пока n > 0
```

```
  посл := n mod 10
  к := к + 1              | Увеличиваем счетчик
  если к = m              | Встретилась m-я цифра
    то
      вывод нс, посл      | Выводим ее
   все
   n := n div 10
кц
```

Можно уменьшить размер программы, учитывая, что мы знаем, сколько раз нужно выделять цифры (m раз). Поэтому можно использовать цикл с параметром:

```
...
нц для к от 1 до m
  посл := n mod 10
  n := n div 10
кц
| Выводим последнюю выделенную (m-ю) цифру
вывод нс, посл
```

А теперь – вариант с преобразованием числа в строку:

```
ввод n
ввод m
n_стр := цел_в_лит(n)
вывод нс, "Это цифра ", n_стр[длин(n_стр) - m + 1]
```

Д4. Определение m-й слева цифры натурального числа

Без преобразования числа в строку данную задачу за один проход решить нельзя. Сначала надо определить общее количество цифр. Потом установить, какой номер имеет m-я слева цифра при счете справа налево и еще раз «пройти по числу» и выделить нужную цифру.

А вот программа с преобразованием числа в строку:

```
ввод n
ввод m
n_стр := цел_в_лит(n)
вывод нс, "Это цифра ", n_стр[m]
```

Как вам она, уважаемый читатель?

Д5. Определение цифры с заданным номером вещественного числа

Программу решения задачи предлагаем читателю разработать самостоятельно.

1.6. Разные задачи

1.6.1. Обмен значениями переменных величин

Задача формулируется так: «Даны значения двух переменных величин *a* и *b*. Произвести обмен их значений».

Решение

Кажущееся очевидным решение

```
a := b
b := a
```

или

```
b := a
a := b
```

требуемого результата не даст (убедитесь в этом!). Как же быть? А так, как происходит обмен содержимого двух чашек, в одной из которых находится молоко, а в другой – чай. Нужна третья чашка[1]! То есть в нашей задаче для решения требуется третья, вспомогательная переменная. С ее использованием обмен может быть проведен следующим образом:

```
c := a | Запоминаем значение величины a
a := b | Величине a присваиваем значение величины b
b := c | Величине b присваиваем "старое" значение величины a
```

или

```
c := b
b := a
a := c
```

1.6.2. Обмен значениями двух элементов массива

Здесь, как и при обмене значениями двух простых величин, необходимо использовать вспомогательную переменную. Если индексы обмениваемых элементов – m1 и m2, то соответствующий фрагмент программы выглядит так:

```
всп := a[m1]
a[m1] := a[m2]
a[m2] := всп
```

[1] Можно, конечно, использовать и две дополнительные чашки, но это уже, так сказать, слишком.

1.6.3. Перестановка всех элементов массива в обратном порядке

Решение

Ясно, что при решении будут повторяться обмены, т. е. в программе можно будет применить оператор цикла с параметром. Но какими должны быть начальное и конечное значение параметра цикла? Для ответа на этот вопрос составим табл. 1.1.

Таблица 1.1

Индекс элемента	С каким элементом он меняется значениями
1	С последним (n-м)
2	С предпоследним, (n - 1)-м
...	...
n - 1	Со вторым
n	С первым

Правильно? Нет, конечно! При таком обмене каждый элемент будет менять значения дважды, и в результате массив не изменится. Менять значения (в левом столбце таблицы) нужно только до половины массива (табл. 1.2).

Таблица 1.2

Индекс элемента	С каким элементом он меняется значениями
1	С n-м
2	С (n - 1)-м
...	...
n div 2 - 1	С (n - n div 2 + 2)-м
n div 2	С (n - n div 2 + 1)-м

Внимание!

Индекс в последней строке левого столбца таблицы нельзя рассчитывать как n/2, так как значение индекса элемента массива может быть только целым. Здесь возникает также вопрос: а что будет, если n – нечетное число? Ответ – в этом случае значения в табл. 1.2 не изменятся, а средний элемент (его индекс – n div 2 + 1) меняться не будет (убедитесь в этом, рассмотрев конкретные значения n).

Для записи соответствующего фрагмента программы необходимо знать, с каким элементом будет меняться значениями i-й элемент. Анализ табл. 1.2 показывает, что сумма индексов обмениваемых элементов равна n + 1. Значит, i-й элемент будет меняться с (n + 1 - i)-м.

Итак, задача решается с помощью такого оператора цикла:

```
нц для i от 1 до div(n, 2)
  | Меняем местами i-й и (n + 1 - i)-й элементы
  вспом := a[i]
  a[i] := a[n + 1 - i]
  a[n + 1 - i] := вспом
кц
```

1.6.4. Рассмотрение всех вариантов сочетания по одному элементу из нескольких наборов

Задачу этого типа в общем виде можно сформулировать следующим образом: «Даны несколько наборов значений. Рассмотреть все варианты сочетания по одну элементу из каждого набора и для каждого сочетания выполнить какие-то действия». Пример такой задачи: «Дан двумерный массив. Рассмотреть по одному элементу из каждой строки и для каждого сочетания найти сумму значений элементов и записать соответствующую сумму в одномерный массив». Например, для массива вида

10	2
6	4
2	5

результирующий одномерный массив будет таким:

18 (10 + 6 + 2)	21 (10 + 6 + 5)	16 (10 + 4 + 2)	19 (10 + 4 + 5)	10 (2 + 6 + 2)	13 (2 + 6 + 5)	8 (2 + 4 + 2)	11 (2 + 4 + 5)

Примечание. В скобках приведены элементы двумерного массива, из которых слагается та или иная сумма.

Решение

Ясно, что использование в программе вложенного цикла

```
сум := 0
нц для i от 1 до ...
```

```
  нц для j от 1 до …
    сум := сум + a[i, j]
  кц
кц
```

позволит определить сумму *всех* элементов массива, в то время как требуется другое.

Правильное решение – перебор элементов каждой строки массива и для каждого сочетания по одному элементу строки – определение суммы значений.

Для двумерного массива из шести строк и двух столбцов соответствующий фрагмент оформляется так:

```
сум := 0
k := 0 | Индекс элемента одномерного массива с суммами
       | (начальное значение)
| Рассматриваем все индексы каждой строки
нц для i1 от 1 до 2
  нц для i2 от 1 до 2
    нц для i3 от 1 до 2
      нц для i4 от 1 до 2
        нц для i5 от 1 до 2
          нц для i6 от 1 до 2
            k := k + 1 | Увеличиваем индекс k
            | Суммируем соответствующие элементы каждой строки
            | и записываем их в одномерный массив сумма на k-е место
            сумма[k] := a[1, i1] + a[2, i2] + a[3, i3] + a[4, i4] +
                        a[5, i5] + a[6, i6]
          кц
        кц
      кц
    кц
  кц
кц
```

1.6.5. Рассмотрение всех пар элементов массива

Возможны три варианта указанной задачи:

1) когда речь идет о соседних элементах. Например, для массива

13 5 67 8 10 13 9 29 12 34

это будут пары 13 и 5, 5 и 67... 12 и 34;

2) когда имеются в виду «непересекающиеся» пары соседних элементов, т. е. первая пара – 1-й и 2-й элементы, вторая

пара – 3-й и 4-й элементы и т. д. На языках Си и Python, соответственно, первая пара – 0-й и 1-й элементы, вторая пара – 2-й и 3-й элементы и т. д.;

3) когда речь идет о всех возможных парах элементов массива. Например, для массива

13 5 67 8 10 13 9 29 12 34

это будут пары 13 и 5, 13 и 67, 13 и 8... 13 и 34, 5 и 67, 5 и 8... 5 и 34, 67 и 8... 12 и 34.

Приведем операторы цикла, с помощью которых можно рассмотреть все пары элементов массива a из n элементов.

Первый вариант задачи

В этом случае рассматриваются пары элементов с индексами 1 и 2, 2 и 3, ..., (n - 1) и n. Если в качестве параметра цикла принять индекс i левого элемента в каждой паре, то оператор цикла должен быть оформлен в виде:

```
нц для i от 1 до n - 1
  ...      | Рассматриваются пары a[i] и a[i + 1]
кц
```

Второй вариант задачи

Отличие данного варианта от предыдущего в том, что рассматривать надо не пары всех соседних элементов, а только пары, элементы которых «отстоят» от соответствующих элементов предыдущей пары на 2 позиции.

```
нц для i от 1 до n - 1 шаг 2
  ...      | Рассматриваются и обрабатываются пары a[i] и a[i + 2]
кц
```

Возможен такой оригинальный способ рассмотрения всех непересекающихся пар элементов в цикле с шагом, равным 1:

```
нц для i от 1 до div(n, 2)
  | Рассматриваются и обрабатываются пары m[2 * i - 1] и m[2 * i]
кц
```

Третий вариант задачи

В этом случае рассмотреть все возможные пары элементов с индексами i и j можно с помощью такого вложенного оператора цикла:

```
нц для i от 1 до n - 1
  нц для j от i + 1 до n
    ...       | Рассматриваются и обрабатываются пары a[i] и a[j]
  кц
кц
```

1.6.6. Вставка значения в массив со сдвигом элементов влево

Подробная формулировка задачи: вставить заданное число на последнее место в массиве, при этом сдвинув второй, третий, ..., последний элементы на одну позицию влево.

Пример изменения массива:

5	12	8	0	14	5	1	3	23	4	8	0	9

а) исходный массив;

12	8	0	14	5	1	3	23	4	8	0	9	99

б) конечный массив.

Решение

Ясно, что сразу записать в программе: `a[n] := 99` нельзя (исходное значение `a[n]` будет утеряно). Сначала следует провести сдвиг элементов влево, начиная со второго[1]:

```
a[1] := a[2]
a[2] := a[3]
...
a[n - 1] := a[n]
```

Если индекс элементов слева от знака присваивания обозначить `i`, то можно использовать оператор цикла с параметром:

```
нц для i от 1 до n - 1
  a[i] := a[i + 1]
кц
| Запись нового числа в конец массива
a[n] := новое_число
```

Можно также сдвиг элементов провести так:

```
нц для i от 2 до n
  a[i - 1] := a[i]
кц
```

[1] «Бывший» 1-й элемент массива уже не понадобится.

Глава 2

Готовимся выполнять задания из ЕГЭ

2.1. Задание 5

В Спецификации контрольных измерительных материалов для проведения в 2021 году единого государственного экзамена по информатике и ИКТ [13][1] применительно к заданию 5 в качестве проверяемого элемента содержания указано: «Формальное исполнение алгоритма, записанного на естественном языке, или умение создавать линейный алгоритм для формального исполнителя с ограниченным набором команд».

Обсудим методику выполнения аналогичных заданий из демонстрационных вариантов ЕГЭ нескольких последних лет.

2.1.1. Задание из [8]

Условие

Автомат получает на вход трехзначное число. По этому числу строится новое число по следующим правилам.

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

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

Пример. Исходное число: 348. Суммы: 3 + 4 = 7; 4 + 8 = 12. Результат: 127.

Укажите наименьшее число, в результате обработки которого автомат выдаст число 1711.

Решение

Так как результат работы алгоритма – четырехзначный, то одна из сумм равна 17, а вторая – 11. Число 17 может быть суммой только двух цифр – 8 и 9. Одна из этих цифр участвует во второй сумме (11), т. е. является «средней» в записи входного числа. Это значит, что возможен один из двух вариантов входного числа:

1) ?8?

2) ?9?

где символом ? обозначена некоторая цифра.

Поскольку вторая сумма цифр равна 11, то неизвестная третья цифра входного числа может быть равна, соответственно, 3 или 2.

По условию следует найти минимальное значение входного числа, т. е. именно одна из этих цифр является в нем первой. Следовательно, возможны два варианта искомого числа:

[1] В дальнейшем для краткости будем называть этот документ «Спецификацией».

1) 389,
2) 298,
из которых минимальное значение – второе.
Итак, ответ: 298.

Задание для самостоятельной работы

Автомат получает на вход четырехзначное число. По этому числу строится новое число по следующим правилам.
1. Складываются первая и вторая, а также третья и четвертая цифры исходного числа.
2. Полученные два числа записываются друг за другом в порядке убывания (без разделителей).
Пример. Исходное число: 3165. Суммы: 3 + 1 = 4; 6 + 5 = 11. Результат: 114.
Укажите наименьшее число, в результате обработки которого автомат выдаст число 1311.

2.1.2. Задание из [7]

Условие
На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:
1) строится двоичная запись числа N;
2) к этой записи дописываются справа еще два разряда по следующему правилу:
   а) складываются все цифры двоичной записи числа N, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;
   б) над этой записью производятся те же действия – справа дописывается остаток от деления суммы ее цифр на 2.

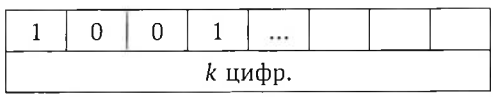
Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью искомого числа R.
Укажите минимальное число R, которое превышает число 83 и может являться результатом работы данного алгоритма. В ответе это число запишите в десятичной системе счисления.

Решение
Проанализируем условие. В нем фигурируют три десятичных числа:

1) исходное число N;
2) результат работы алгоритма R;
3) некоторое целое число (в данном случае – 83), которое назовем «граничным» и обозначим G.

Если соответствующие двоичные записи перечисленных чисел обозначим $N2$, $R2$ и $G2$, то решение задачи сводится к двум этапам:
1) нахождение такого минимального значения $N2$, которое в результате выполнение алгоритма даст значение $R2$, большее $G2$;
2) определение десятичного эквивалента найденного значения $N2$.

Значение $G2$ получить легко. Пусть оно имеет вид (условный):



Про значение $R2$ мы знаем, что оно имеет вид:

где оттененная часть совпадает со значением $N2$.

Сделаем важное для решения утверждение. Длина (количество цифр) значения $R2$, соответствующего искомому R, должна быть не менее длины числа $G2$. Это значит, что поиск значения $N2$ можно начинать с числа, состоящего из (k – 2) цифр значения $G2$.

Переведем 83 в двоичную систему – получим значение 1010011.

Согласно утверждению, сделанному чуть выше, минимально возможное значение $N2$ равно 10011. Исследуем это значение:

$N2$	**Кол-во единиц**	**Двоичная запись результата ($R2$)**	**Значение R**
10100	Четное	1010000	80
10101	Нечетное	1010110	86

Так как 86 > 83, то решение найдено.

Следовательно, ответ: 86.

Итак, алгоритм решения обсуждаемой задачи можно сформулировать следующим образом:

1) по заданному граничному числу G получить $G2$;
2) отбросить в $G2$ две последние цифры;

3) принимая новое и последующие за ним двоичные числа в качестве числа $N2$, приписывать к каждому две новые цифры по правилу:

если количество единиц в числе $N2$ нечетное,
то
к нему приписываются цифры 1 и 0,
иначе
к нему приписываются цифры 0 и 0,
все.

Первое число $N2$, дающее результат $R2 > G2$, определяет искомое значение R.

Задания для самостоятельной работы

1. Для приведенного выше условия определите такое наименьшее число R, которое может являться результатом работы данного алгоритма и превышает:
а) 25;
б) 41;
в) 88.

2. На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:
1) строится двоичная запись числа N;
2) к этой записи дописываются справа еще два разряда по следующему правилу:
а) если остаток от деления суммы всех цифр двоичной записи числа N на 2 равен нулю, то в конец числа (справа) дописывается 1, в противном случае – 0. Например, запись 11000 преобразуется в запись 110001;
б) над новой записью производятся те же действия.

Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью числа R.

Укажите минимальное число R, которое может являться результатом работы данного алгоритма и превышает:
а) 25;
б) 41;
в) 88.

2.1.3. Задание из [6]

Условие

На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:

1) строится двоичная запись числа N;
2) к этой записи дописываются справа еще два разряда по следующему правилу: если N четное, в конец числа (справа) дописывается сначала ноль, а затем единица. В противном случае, если N нечетное, справа дописывается сначала единица, а затем ноль.

Например, двоичная запись 100 числа 4 будет преобразована в 10001, а двоичная запись 111 числа 7 будет преобразована в 11110. Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью числа R – результата работы данного алгоритма.

Укажите минимальное число R, которое больше 102 и может являться результатом работы данного алгоритма. В ответе это число запишите в десятичной системе счисления.

Решение

Анализ показывает, что указанная задача отличается от рассмотренной в предыдущем разделе чуть измененным алгоритмом преобразования заданного числа, и решать ее можно аналогично.

Прежде чем проводить анализ, заметим, что у четного числа последняя двоичная цифра – 0, у нечетного – 1.

Переведем 102 в двоичную систему – получим значение 1100110.

Таблица для анализа:

$N2$	**Число N**	**Двоичная запись результата ($R2$)**	**Значение R**
11001	Нечетное	1100110	102
11010	Четное	1101001	105

Следовательно, ответ: 105.

2.1.4. Задание из [5]

Условие

На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:

1) строится двоичная запись числа N;
2) к этой записи дописываются справа еще два разряда по следующему правилу:
   а) складываются все цифры двоичной записи числа N, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;
   б) над этой записью производятся те же действия – справа дописывается остаток от деления суммы ее цифр на 2.

Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью искомого числа R.

Укажите минимальное число R, которое превышает число 97 и может являться результатом работы данного алгоритма. В ответе это число запишите в десятичной системе счисления.

Решение

Оно аналогично решению задачи в п. 2.1.2.

Переведем 97 в двоичную систему – получим значение 1100001.

Таблица для анализа:

N2	**Кол-во единиц**	**Двоичная запись результата (R2)**	**Значение R**
11000	Четное	1100000	80
11001	Нечетное	1100110	102

Итак, ответ: 102.

2.2. Задание 6

В Спецификации [13] применительно к заданию 6 в качестве проверяемого элемента содержания указано: «Знание основных конструкций языка программирования, понятия переменной, оператора присваивания».

Обсудим сначала методику выполнения аналогичных заданий из демонстрационных вариантов ЕГЭ нескольких последних лет[1].

[1] В некомпьютерных вариантах экзамена эти задания имели номер 8.

Задание из [5]

Условие

Запишите число, которое будет напечатано в результате выполнения следующей программы. Для вашего удобства программа представлена на четырех языках программирования.

Алгоритмический язык

```
алг
нач
  цел n, s
  s := 0
  n := 1
  нц пока s < 51
    s := s + 11
    n := n * 2
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
   s := 0;
   n := 1;
   while s < 51 do
      begin
        s := s + 11;
        n := n * 2
      end;
   writeln(n)
END.
```

Python

```
s = 0
n = 1
while s < 51:
   s = s + 11
   n = n * 2
print(n)
```

C++

```
# include <iostream>  using namespace std;
int main()
{
int s = 0, n = 1;
while (s < 51) { s = s + 11; n = n * 2;
}
cout << n << endl;
return 0;
}
```

Решение

Анализ условия показывает, что:

1) меняются значения величин s и n;
2) в условии продолжения работы оператора используется значение переменной s;
3) результатом выполнения программы является окончательное значение величины n.

Сколько раз будет выполнено тело оператора цикла с условием `s < 51`? Ответить на этот вопрос можно, сделав расчеты буквально на пальцах (это будет для значений s, равных 0, 1, 22, 33 и 44,

т. е. 5 раз). Значение n увеличивается в 2 раза при начальном значении, равном 1, т. е. рассчитываются степени числа 2. Пятая степень равна 32.

Ответ: 32.

Задание из [8]

Условие

Определите, какое число будет напечатано в результате выполнения программы, записанной ниже на четырех языках программирования.

Алгоритмический язык

```
алг
нач цел n, s
  n := 1
  s := 0
  нц пока n <= 150
    s := s + 30
    n := n * 5
  кц
  вывод s
кон
```

Паскаль

```
var n, s: integer;
BEGIN
  n := 1;
  s := 0;
  while n <= 150 do
    begin
      s := s + 30;
      n := n * 5
    end;
  write(s)
END.
```

Python

```
n = 1
s = 0
while n <= 150:
  s = s + 30
  n = n * 5
print(s)
```

C++

```
# include<stdio.h>
int main()
{
int n, s;
n = 1;
s = 0;
while (n <= 150)
{
s = s + 30;
n = n * 5;
}
printf("%d", s);
return 0;
}
```

Решение

Анализ условия показывает, что:

1) меняются значения величин s и n;
2) в условии продолжения работы оператора используется значение переменной n;
3) результатом выполнения программы является окончательное значение величины s.

В теле оператора цикла значение величины n увеличивается в 5 раз (начальное значение 1), т. е. рассчитываются степени числа 5. Сколько раз будет выполнено тело цикла с условием n <= 150? Максимальная степень степени числа 5, не превышающая 150, – число 125 (5^3). Значит, тело оператора цикла будет выполнено 3 + 1 = 4 раза. Следовательно, и величина s тоже будет изменена 4 раза, т. е. ее окончательное значение будет равно 30 × 4 = 120.

Ответ: 120.

Задание из [7]

Условие

Запишите число, которое будет напечатано в результате выполнения следующей программы. Для вашего удобства программа представлена на четырех языках программирования.

Алгоритмический язык

```
алг
  нач цел n, s
  s := 260
  n := 0
  нц пока s > 0
    s := s - 15
    n := n + 2
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  s := 260;
  n := 0;
  while s > 0 do
    begin
      s := s - 15;
      n := n + 2
    end;
  writeln(n)
END.
```

Python

```
s = 260
n = 0
while s > 0:
  s = s - 15
  n = n + 2
print(n)
```

C++

```
# include <iostream> using
namespace std;
int main() {
int s = 260, n = 0;
while (s > 0) {
s = s - 15;
```

```
    n = n + 2;
    }
    cout << n << endl;
    return 0;
    }
```

Решение

Анализ условия показывает, что:

1) меняются значения величин s и n;
2) в условии продолжения работы оператора цикла используется значение переменной s;
3) результатом выполнения программы является окончательное значение величины n.

Значение величины s уменьшается на 15 (начальное значение 260). Сколько раз будет выполнено тело оператора цикла с условием s > 0? Для ответа найдем целую часть частного от деления 260 на 15 – 17, т. е. общее число повторений тела оператора цикла будет равно 1 + 16 = 18. Так как величина n увеличивается на 2 (начальное значение 0), то ее окончательное значение будет равно 18 × 2 = 36.

Ответ: 36.

Задание из [6]

Условие

Запишите число, которое будет напечатано в результате выполнения следующей программы. Для вашего удобства программа представлена на четырех языках программирования.

Алгоритмический язык

```
алг
  нач цел n, s
  s := 0
  n := 75
  нц пока s + n < 150
    s := s + 15
    n := n - 5
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  s := 0;
  n := 75;
  while s + n < 150 do
    begin
      s := s + 15;
      n := n - 5
    end;
  writeln(n)
END.
```

Python

```
s = 0
n = 75
while s + n < 150:
  s = s + 15
  n = n - 5
print(n)
```

C++

```
# include <iostream> using
namespace std;
int main() {
int s = 0, n = 75;
while (s + n < 150) {
s = s + 15;
n = n - 5;
}
cout << n << endl;
return 0;
}
```

Решение

Анализ условия показывает, что:

1) меняются значения величин s и n;
2) результатом выполнения программы является окончательное значение величины n.

Особенность данной задачи в том, что в операторе цикла в условии используется сумма меняющихся значений n и s.

Сколько раз будет выполнено тело цикла с условие `s + n < 150`? Здесь для ответа надо учесть, что при каждом повторении сумма `s + n` будет увеличиваться на 15 – 5 = 10. Так как начальное значение этой суммы равно 0 + 75 = 75, то тело оператора цикла будет выполнено для таких значений суммы `s + n`: 75, 85... 145, т. е. 8 раз. Следовательно, и величина `n` тоже будет изменена (уменьшена на 5) 8 раз, т. е. ее окончательное значение будет равно 75 – 8 × 5 = 35.

Ответ: 35.

Итак, общие рекомендации по выполнению задания 6 можно сформулировать следующим образом.

1. Установить, какие переменные и как меняются в теле оператора цикла.

2. Определить, окончательное значение какой величины выводится на экран.

3. Рассчитать количество повторений тела оператора цикла, учитывая начальные значения меняющихся величин, особенности их изменения в теле оператора и условие продолжения его (оператора) работы.

4. Определить окончательное значение величины, значение которой выводится на экран, учитывая ее начальное значение,

особенности его изменения и количество повторений тела оператора цикла.

В заключение заметим, что в [12] задания 6 приведены в несколько другой формулировке. В них требуется определить, какое максимальное или минимальное значение необходимо ввести, чтобы в результате работы программы на экран было выведено некоторое значение.

Пример 1

Какое максимальное число s можно ввести, чтобы в результате работы программы на экране было напечатало 256? Для вашего удобства программа представлена на нескольких языках программирования.

Алгоритмический язык

```
алг
нач цел s, n
  n := 1
  ввод s
  нц пока s <= 180
    s := s + 30
    n := n * 4
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  n := 1;
  readln(s);
  while s <= 180 do
    begin
      s := s + 30;
      n := n * 4
    end;
  writeln(n)
END.
```

Python

```
n = 1
s = input()
while s + n < 180:
  s = s + 30
  n = n * 4
print(n)
```

Решение

Анализ показывает, что в программе на экран выводится число n, являющееся некоторой степенью числа 4. Показатель этой степени равен количеству повторений тела оператора цикла. Число 256 в условии – это четвертая степень числа 4. Значит, количество повторений равно 4. Искомое максимальное значение можно найти, вычитая из 180 число 30 четыре раза: 180, 150, 120, 90.

Ответ: 90.

Пример 2

Какое минимальное число s можно ввести, чтобы в результате работы программы на экране было напечатано четырехзначное число? Для вашего удобства программа представлена на нескольких языках программирования.

Алгоритмический язык

```
алг
  нач цел s, n
  n := 44
  ввод s
  нц пока n <= 180
    s := s + 110
    n := n + 25
  кц
  вывод s
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  n := 44;
  readln(s);
  while s <= 180 do
    begin
      s := s + 110;
      n := n + 25
    end;
  writeln(s)
END.
```

Python

```
n = 44
s = input()
while s + n < 180:
  s = s + 110
  n = n + 25
print(n)
```

Решение

Определим, сколько раз будет выполняться тело оператора цикла. Это количество равно целой части частного от деления разности 180 и 44 на 25, увеличенной на 1, т. е. 6 (при значениях n = 44, 69, 94, 119, 144, 169).

Так как величина s увеличивается на 110, то ее минимальное значение, дающее четырехзначный результат, равно $1000 - 6 \times 110 = 340$.

Ответ: 340.

Пример 3

Какое минимальное значение необходимо ввести, чтобы в результате работы программы на экране было выведено 202? Для вашего удобства программа представлена на нескольких языках программирования.

Алгоритмический язык

```
алг
нач цел k, s, d
  s := 4
  k := 0
  ввод d
  нц пока s < d
    k := k + 3
    s := s + k
  кц
  вывод s
кон
```

Паскаль

```
var k, s, n: integer;
BEGIN
  s := 4;
  k := 0
  readln(d);
  while s < d do
    begin
      k := k + 3;
      s := s + k
    end;
  writeln(s)
END.
```

Python

```
s = 4
k = 0
d = input()
while s < d:
  k = k + 3
  s = s + k
print(s)
```

Решение

Это задание – самое сложное из рассмотренных.

Определим, какие значения могут выводиться на экран. Для этого составим такую таблицу изменения значений переменных k и s:

k	0	3	6	9	12	15	18	21	24	27	30	33	36	39
s	4	7	13	22	34	49	67	88	112	139	169	202	238	277

Далее рассуждения такие. Пусть есть некоторое значение переменной d. Какое значение будет при этом выведено? Ответ – число, следующее за максимальным значением s в таблице, которое меньше d. Например,

- при d = 87 будет выведено число 88 (следующее за 88);
- при d = 88 будет выведено число 88 (следующее за 67);
- при d = 89 будет выведено число 112 (следующее за 88);
- при d = 90 будет выведено число 112 (следующее за 88).

Анализ приведенных примеров показывает, что для вывода числа 202 минимальное значение d должно быть на один больше предшествующего ему числа 169, т. е. 170.

Ответ: 170.

2.3. Задание 10

В Спецификации [13] применительно к заданию 10 в качестве проверяемого элемента содержания указано: «Информационный поиск средствами операционной системы или текстового процессора».

В [1] в задании требуется определить, сколько раз, не считая сносок, встречается слово «долг» или «Долг» в тексте романа в стихах А. С. Пушкина «Евгений Онегин». Другие формы слова «долг», такие как «долги», «долгами» и т. д., учитываться не должны.

Указанная задача может быть достаточно быстро решена путем использования такой возможности текстового редактора Microsoft Word, как поиск в документе нужных фрагментов текста (слов или их части). Запрос на поиск может быть достаточно сложным, включающим учет регистра букв (строчные или прописные), длину слов, комбинацию символов и т. д.

Обсудим основные, простые возможности.

Для поиска с простым запросом достаточно вызвать панель **Навигация**. Для этого в ленте Word надо переключиться на вкладку **Главная** и нажать кнопку **Заменить** в правой части вкладки (см. рис. 2.3.1).

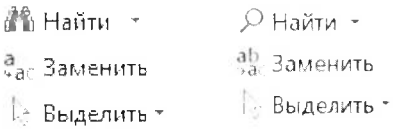


Рис. 2.3.1

Можно также нажать комбинацию клавиш **Ctrl** и **F**.
В результате в левой части экрана появится панель **Навигация**:

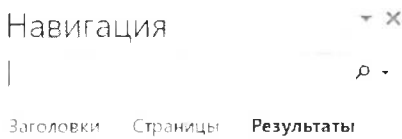


Рис. 2.3.2

То, что надо найти, указывается в поле, в котором изображена лупа. Можно также выделить нужное слово или его часть, после чего вызвать панель **Навигация**.

Итак, начнем.

Для поиска всех вхождений, например последовательности букв *порт*, следует записать эту последовательность в окне поиска[1] и нажать клавишу **Enter**. Будут найдены и выделены цветом все имеющиеся в документе слова *порт*, а также другие слова, в которых имеется указанная последовательность букв (*спорт*, *портал*, *паспорт*, *портативный* и т. п.). Их общее количество (нужное в нашем случае значение) будет указано в верхней части панели **Навигация** (рис. 2.3.3).

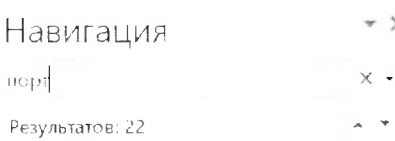


Рис. 2.3.3

Обратим внимание на то, что слова, в которых буква *п* – прописная (Порт, Портал и т. п.), учтены не будут. Чтобы учесть только их, следует в поле поиска записать *Порт*.

---

[1] Или, как указывалось чуть выше, перед открытием панели **Навигация** предварительно выделить нужную последовательность букв в тексте.

Обсудим, что будет найдено, если записать в поле поиска *порт?*. Как, очевидно, известно читателю, в масках (шаблонах) для поиска файлов по их именам символ ? означает «один любой символ». Такой же смысл он имеет и при поиске, т. е. будут найдены и выделены, например, такие фрагменты: *порт*алы, с*порт*ивный, *порт*.[1] и т. п.

А если записать в поле поиска *порт**? Применительно к именам файлов символ «звездочка» означает «любая допустимая последовательность символов». При поиске этот символ также имеет такой же смысл, только в результате поиска будут выделены лишь последовательности букв *порт*, т. е. так же, как и при поиске без символа * (естественно, что и количество найденных вхождений будет тем же).

Хотя выше уже отмечалось, как определить количество вхождений, начинающихся с прописной буквы, расскажем о другом способе, который понадобится нам для более сложных поисков. В этом случае возможны два варианта подготовительных действий:

1) после открытия панели **Навигация** путем нажатия клавиш **Ctrl** и **F** щелкнуть на треугольничке справа от изображения лупы и в появившемся списке выбрать **Расширенный поиск...** (см. рис. 2.3.4);

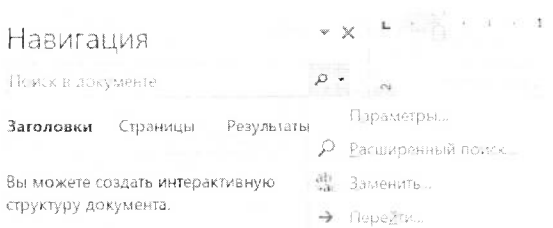


Рис. 2.3.4

Появится окно **Найти и заменить** с активной вкладкой **Найти** (см. рис. 2.3.5).

[1] Точка также будет выделена цветом.

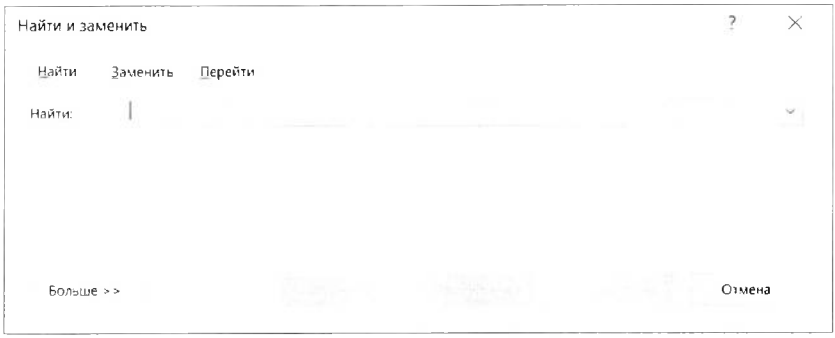


Рис. 2.3.5

2) в ленте Word переключиться на вкладку **Главная** и нажать кнопку **Заменить** в правой части вкладки – будет открыто окно **Найти и заменить**, в котором активной будет вкладка **Заменить**.

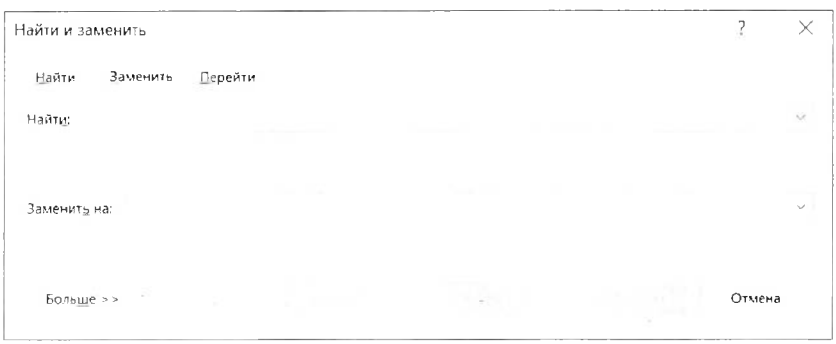


Рис. 2.3.6

Для поиска можно работать с активной вкладкой[1], но лучше перейти на вкладку **Найти**.

В обоих вариантах подготовительных действий, чтобы при поиске учитывались тексты с прописными буквами (например, *Порт*, *Портал*, *Порт*фель и т. п.), следует:

1) в поле ввода записать нужный текст;
2) поставить галочку в переключателе с надписью **Учитывать регистр** (см. рис. 2.3.7).

[1] В этом случае подсчет искомых значений произойдет при одновременной их замене на «пустое» значение, т. е. указанные значения будут удалены.

При этом активной станет кнопка **Найти в**, по щелчку на которой (в пункте **Основной документ**) будут показаны все результаты поиска (рис. 2.3.7).

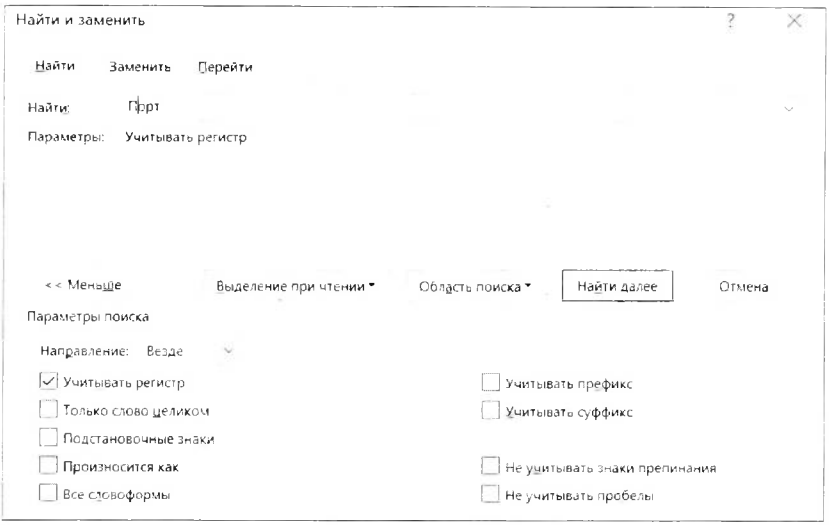


Рис. 2.3.7

Более сложный поиск проводится с использованием так называемых регулярных выражений. Регулярное выражение – это запись, которая представляет собой условное обозначение критериев, которым должен соответствовать искомый текст. С помощью регулярных выражений можно найти множество слов или их частей, соответствующих заданным, при необходимости достаточно сложным условиям.

Опишем запросы на поиск с достаточно простыми условиями в регулярных выражениях.

Регулярное выражение для поиска записывается в поле **Найти**. В нем используются так называемые подстановочные знаки. Их полный перечень можно увидеть, поставив в окне поиска галочку в переключателе с надписью **Подстановочные знаки**, после чего щелкнуть по кнопке **Специальный** в нижней части окна.

Можно также записывать знаки вручную (с помощью клавиатуры), но при использовании подстановочных знаков указанный переключатель должен быть обязательно включен.

Приведем несколько примеров.

1. Чтобы найти все слова, *начинающиеся* на «порт» (*порт*ативный, *порт*ал, *порт* и т. п., но не спорт), регулярное выражение должно быть таким:

```
<порт*
```

Внимание! При включенном переключателе с надписью **Подстановочные знаки** переключатель с надписью **Учитывать регистр** (рис. 2.3.7) будет недоступен, т. е. слова, *начинающиеся* на «Порт» (*Порт*ативный, *Порт*ал, *Порт* и т. п.), учтены не будут.

Чтобы найти их (только!), регулярное выражение должно быть таким:

```
<Порт*
```

2. Регулярное выражение

```
а>
```

найдет все слова, оканчивающиеся на букву *а*.

Как, конечно, уже понял читатель, знак < (меньше) означает «начало слова», а знак > (больше) – «конец слова».

3. Чтобы найти все слова, начинающиеся на любую букву, за которой следует буква *а*, регулярное выражение имеет вид:

```
<?а
```

Задания для самостоятельной работы

Определите количество вхождений в документ текстового редактора Microsoft Word:

1) буквы *ю*;
2) последовательностей букв *поле*, *поли*, *полк*, *пола*, *полу*, *полы*, *поля*;
3) начинающихся на *без*;
4) оканчивающихся на *лан*;
5) начинающихся на две любые буквы, за которыми следует буква *о*;
6) слов *мир*;
7) слов *марка* и *Марка*.

Документы для выполнения заданий подготовьте самостоятельно.

2.4. Задание 12

В Спецификации [13] применительно к заданию 12 в качестве проверяемого элемента содержания указано: «Умение анализировать результат исполнения алгоритма».

Общий вид задания

В задании речь идет о выполнении программы исполнителем **Редактор**, который получает на вход строку цифр или строку букв и преобразовывает ее.

Редактор может выполнять две команды, в обеих командах `v` и `w` обозначают цепочки цифр.

1) `заменить(v, w)`.

Эта команда заменяет в строке первое слева вхождение цепочки *v* на цепочку *w*. Например, выполнение команды `заменить(111, 27)` преобразует строку 05111150 в строку 0527150.

Если в строке нет вхождений цепочки *v*, то выполнение команды `заменить(v, w)` не меняет эту строку.

2) `нашлось(v)`.

Эта команда проверяет, встречается ли цепочка `v` в строке исполнителя **Редактор**. Если она встречается, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Строка исполнителя при этом не изменяется.

Цикл

```
ПОКА условие
  последовательность команд
КОНЕЦ ПОКА
```

выполняется, пока условие истинно.

В конструкции

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

выполняется *команда*1 (если условие истинно) или *команда*2 (если условие ложно).

Приводится программа для указанного исполнителя вида:

```
НАЧАЛО
  ПОКА нашлось(цепочкаА) ИЛИ нашлось(цепочкаБ)
    ЕСЛИ нашлось(цепочкаА)
      ТО заменить(цепочкаА, чем_заменяется_цепочкаА)
      ИНАЧЕ заменить(цепочкаБ, чем_заменяется_цепочкаБ)
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

где *цепочкаА* и *цепочкаБ* – цепочки, соответственно, из цифр (или букв), обозначенных в общем виде *А* и *Б*.

Требуется определить, какая строка получится в результате применения приведенной ниже программы к строке, состоящей из N идущих подряд цифр (или букв) *А* (или *Б*).

В ответе следует записать полученную строку.

Обратим внимание на то, что в [13] указывается, что использование специализированного программного обеспечения для выполнения задания не требуется.

Решение

Обсудим сначала простейший вариант, при котором *цепочкаА* заменяется на половину *цепочкиБ*, а *цепочкаБ* – на половину *цепочкиА*, как, например, в такой программе:

```
НАЧАЛО
  ПОКА нашлось(5555) ИЛИ нашлось(0000)
    ЕСЛИ нашлось(5555)
      ТО заменить(5555, 00)
      ИНАЧЕ заменить(0000, 55)
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

когда требуется определить, какая строка получится в результате применения приведенной ниже программы к строке, состоящей из 64 идущих подряд цифр 5.

Анализ программы показывает, что при заменах одной цепочки на другую общая длина заменяемой части строки уменьшается в два раза. Значит, в данном случае (только!) количество цифр в строке будет таким: 64 32 16 8 4 2. При этом исходная цифра многократно меняется на вторую, и наоборот. Учитывая это, можно составить таблицу:

Длина строки	Повторяющаяся цифра	Условие в цикле Пока
64	5	Истинно[1]
32	0	Истинно
16	5	Истинно
8	0	Истинно
4	5	Истинно
2 (не меняется)	0	Ложно

из которой следует, что результатом выполнения программы является строка 00.

Итак, *ответ*: 00.

Обратим внимание на то, что если бы исходное число цифр 5 в строке было равно 128, то после окончания работы алгоритма осталась бы цепочка из двух цифр 5.

Теперь – более сложный вариант задачи. Пусть для той же программы исходная строка состояла из 66 идущих подряд цифр 5.

Решение

Сначала 64 цифры 5 заменятся на 32 цифры 0, и при этом в конце останутся две цифры 4. Удобно для анализа составить такую таблицу:

Цифра в начале строки	Меняется (количество цифр)	Не меняется (количество цифр)	Последние цифры строки
5	64 → 32	Две	55
0	32		55[2]
5	16		55
0	8		55
5	4		55
0	2		55

Видно, что после первой итерации (первого шага повторений действий) меняющееся количество цифр стало равно степени двойки, это позволяет исследовать работу алгоритма так, как это делалось применительно к предыдущему примеру.

---

[1] На экзамене, с целью сокращения времени выполнения задания, вместо слова *Истина* можно записывать букву *И*.

[2] Можно повторяющиеся значения не записывать.

Из таблицы следует, что результат работы программы – строка 0055.

Решение для случая, когда общее число цифр во входной строке – 134:

Цифра в начале строки	Меняется (количество цифр)	Не меняется (количество цифр)	Последние цифры строки
5	132 → 66	Две	55
0	64 → 32	Две	0055
5	32		0055
0	16		0055
5	8		0055
0	4		0055
5	2		0055

Ответ: 550055.

И еще один пример – когда общее число цифр во входной строке – 22.

Цифра в начале строки	Меняется (количество цифр)	Не меняется (количество цифр)	Последние цифры строки
5	20 → 10	Две	55
0	10 → 4	Две	0055
5	4 → 2		0055
0	2		0055

После перечисленных итераций строка будет иметь вид: 000055. Это означает, что в программе условие в цикле Пока будет истинным и итерации продолжатся:

000055 → 5555 → 00.

В этом случае, самом сложном, *ответ*: 00.

Задание для самостоятельной работы

Применительно к последней программе определите, какая строка получится в результате, если исходная строка состоит из 71 идущих подряд цифр 5.

Другие задания для самостоятельной работы

1. Дана программа для исполнителя Редактор:

```
НАЧАЛО
  ПОКА нашлось(4444) ИЛИ нашлось(777)
    ЕСЛИ нашлось(4444)
      ТО заменить(4444, 77)
      ИНАЧЕ заменить(777, 4)
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

Какая строка получится в результате применения приведенной выше программы к строке, состоящей из 204 идущих подряд цифр 4?

2. Дана программа для исполнителя Редактор:

```
НАЧАЛО
  ПОКА нашлось(25) ИЛИ нашлось(355) ИЛИ нашлось(4555)
    ЕСЛИ нашлось(25)
      ТО заменить(25, 4)
    КОНЕЦ ЕСЛИ
    ЕСЛИ нашлось(355)
      ТО заменить(355, 2)
    КОНЕЦ ЕСЛИ
    ЕСЛИ нашлось(4555)
      ТО заменить(4555, 3)
    КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

Какая строка получится в результате применения приведенной выше программы к строке, состоящей из цифры 2 и следующих за ними 81 цифр 5?

2.5. Задание 13

В Спецификации [13] применительно к заданию 13 в качестве проверяемого элемента содержания указано: «Умение представлять и считывать данные в разных типах информационных моделей (схемы, карты, таблицы, графики и формулы)».

В демонстрационных вариантах ЕГЭ по информатике нескольких последних лет задания имели следующий общий вид:

1) «На рисунке <*приводился рисунок*> представлена схема дорог, связывающих города <*приводился перечень городов*>. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города <*указывался город*> в город <*указывался город*>?»

или

2) «На рисунке <*приводился рисунок*> представлена схема дорог, связывающих города <*приводился перечень городов*>. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города <*указывался город*> в город <*указывался город*>, проходящих через город <*указывался город*>?»

Задания такого типа называются «Подсчет количества путей в ориентированном графе». *Граф* в информатике – это конечное число точек на плоскости (называемых узлами, или вершинами), соединенных отрезками кривых линий (называемых ребрами).

Такие задачи могут быть решены различными способами. Рассмотрим способ, описанный в [14].

Представьте, что однажды Чеширскому Коту[1] нужно было подсчитать количество маршрутов, которыми он сможет добраться из пункта A в пункт H на графе, схема которого показана на рис. 2.5.1.

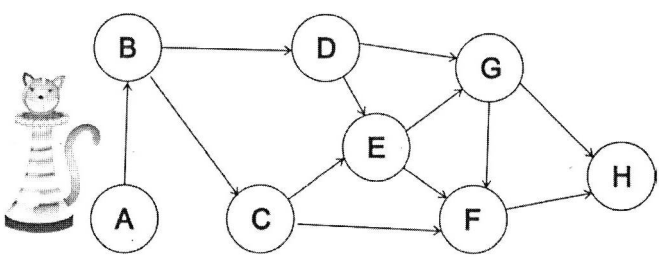


Рис. 2.5.1

Чтобы многократно не ходить к цели и обратно, Кот придумал оригинальный метод подсчета. Он отправил из пункта A по маршруту... свою копию. Далее он поступал так.

В тех пунктах, куда приходит одна дорога, по которой пришла одна копия, а выходят две дороги (например, в пункте B), он добавлял еще одну свою копию, каждая из которых шла по своей дороге:

[1] Надеемся, что читателю известен этот литературный герой.

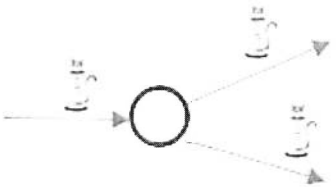


Рис. 2.5.2

Если же в таком случае выходят три и более дорог, то и число копий увеличивается соответственно.

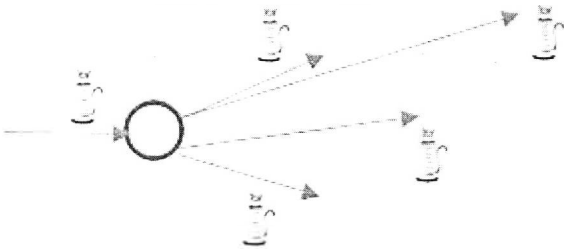


Рис. 2.5.3

Когда в какой-то пункт пришли две копии нашего «умника», а выходящих дорог тоже две (см. рис. 2.5.4), то и число копий удваивается, и каждая пара идет по своей дороге.

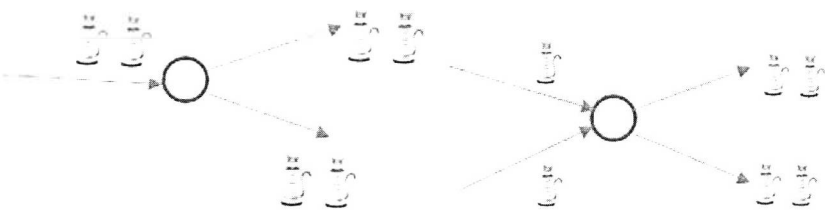


Рис. 2.5.4

В общем случае, когда в некоторый пункт пришли (по всем дорогам) n копий котов, а из него выходят несколько дорог, то по каждой из них должны уйти по n копий.

При таком решении каждый кот-копия будет идти своей дорогой и не будет котов, прошедших по одному и тому же маршруту. А число котов, пришедших в точку H, и будет равно искомому числу маршрутов.

Итак, пункт A – начало маршрута, из которого кот отправил в путешествие свою первую копию. Из A выходит единственная дорога, которая приведет ее в пункт B. Далее дороги расходятся, и из пункта B далее выйдут два кота – один отправится в направлении пункта C, другой – в пункт D. В каждом из этих пунктов произойдет то же самое – число выходящих копий удвоится. Поэтому в пункт E придут два кота, а выйдут – 2 × 2 = 4 (два из них отправятся в пункт G, и два кота – в пункт F). В результате в пункт G придут два кота, вышедшие из E, и один кот из пункта D. Подпишем на графе количество котов, пришедших в тот или иной пункт. У первого пункта (на старте) стоит единица, а далее числа получаются в результате сложения чисел, соответствующих пунктам, из которых можно попасть в данный пункт (см. рис. 2.5.5).

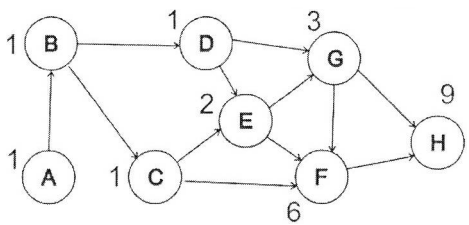


Рис. 2.5.5

Из рис. 2.5.5 следует, что к финишу придут 9 котов-копий, т. е. искомое количество маршрутов перемещения из пункта A в пункт H равно 9.

Можно также, начиная с первого пункта, записывать число копий, пришедших в тот или иной пункт, в таблицу:

Пункт	A	B	C	D	E	F	G	H
Число копий	1	1	1	1	2	...		

Можно также записывать на графе, на его ребрах (в данном случае – на дорогах, соединяющих пункты), число, равное количеству копий котов, идущих по данной дороге, – см. рис. 2.5.6. Искомое число будет равно сумме чисел на дорогах, ведущих в пункт H, т. е. также 9. Какой метод удобнее – решать вам, уважаемый читатель.

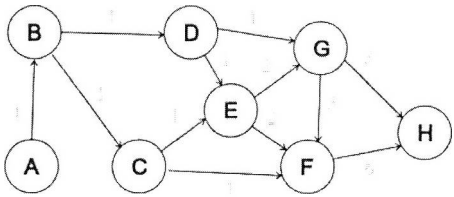


Рис. 2.5.6

Заметим, что если дорога имеет участки-петли, возвращающие в пункт, в котором уже были, то подсчитать количество путей будет невозможно, так как всегда будет копия, отправляющаяся на участок с возвратом. А если таких участков нет, то рано или поздно все копии соберутся в конечном пункте.

Задания для самостоятельной работы

1. На рис. 2.5.7 приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город Ж?

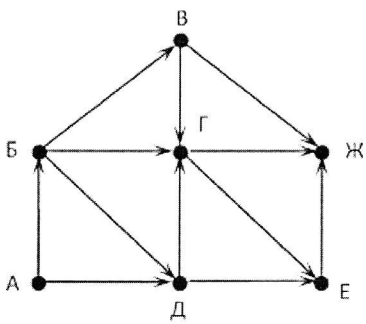


Рис. 2.5.7

2. На рис. 2.5.8 приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж и К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?

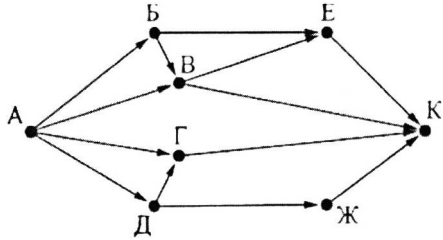


Рис. 2.5.8

3. На рис. 2.5.9 приведена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К и Л. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город Л?

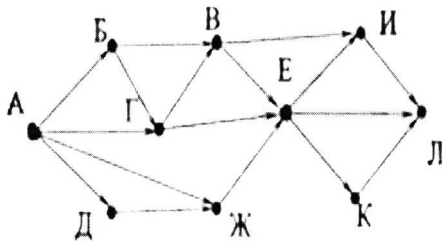


Рис. 2.5.9

Теперь о заданиях второго типа, указанного в начале раздела. Его отличие в том, что в путях, количество которых должно быть определено, должен обязательно присутствовать заданный «промежуточный» город.

В этом случае задача разбивается на две подзадачи:

1) определение количества путей, ведущих из исходного города в «промежуточный»;
2) определение количества путей, ведущих из «промежуточного» города в пункт назначения.

Если первое количество равно N_1, а второе – N_2, то искомое в основной задаче число путей равно $N_1 \times N_2$ (каждый из путей первой подзадачи можно объединить с каждый из путей второй подзадачи).

Аналогично решаются задачи, в которых фигурируют два и более «промежуточных» города.

Задание для самостоятельной работы

4. На рис. 2.5.10 представлена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К, Л, М. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город М, проходящих через город Ж?

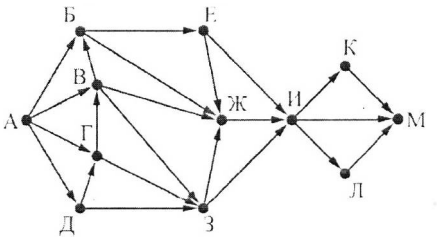


Рис. 2.5.10

2.6. Задание 16

В Спецификации [13] применительно к заданию 16 в качестве проверяемого элемента содержания указано: «Вычисление рекуррентных выражений».

Прежде всего обратим внимание на то, что в задании 16 в [1] и в аналогичных заданиях демонстрационных вариантов экзамена нескольких последних лет речь идет об использовании так называемых рекурсивных функций.

Рекурсивными называют функции, которые используют (вызывают) как вспомогательную саму себя. Непонятно? Тогда – по порядку.

Что делают известные вам стандартные функции `sin`, `sqrt` и другие при их использовании в программах? Возвращают какое-то значение (результат расчетов или т. п.). Так вот, с такой же целью можно также создавать и использовать собственные, нестандартные функции. Пусть, например, в программе надо рассчитать значения x:

$$x = \frac{2+\sqrt{2}}{5+\sqrt{5}} + \frac{5+\sqrt{5}}{13+\sqrt{13}} + \frac{11+\sqrt{11}}{8+\sqrt{8}} + \frac{14+\sqrt{14}}{7+\sqrt{7}}.$$

Видно, что в выражении имеются похожие фрагменты – дроби. Желательно для расчета дробей оформить функцию, которая вычисляла бы значения отдельных дробей вида $\frac{a+\sqrt{a}}{b+\sqrt{b}}$ при любых значениях a и b, а затем использовать ее для расчетов.

Правила оформления функций в разных языках программирования различаются. В школьном алгоритмическом языке общий вид функций такой:

```
алг <тип результата> <имя функции> (<список формальных параметров>)
нач <описание переменных величин, используемых в функции>
  <тело функции>
кон
```

причем последним оператором тела функции должен быть оператор присваивания, в левой части которого должно быть указано служебное слово знач.

В нашем случае функция для расчета значения дробей указанного чуть выше вида оформляется так:

```
алг вещ Др(арг цел a, b)
нач
  знач := (a + sqrt(a))/(b + sqrt(b))
кон
```

Как и стандартные, функция, созданная программистом, используется в правой части оператора присваивания[1]. Для этого надо указать ее имя, а в скобках – значения фактических параметров:

```
x := Др(2,5) + Др(5,13) + Др(13, 8) + Др(14,7)
```

Видно, что благодаря использованию функции оператор для расчета значения x гораздо компактнее такого варианта:

```
x := (2 + sqrt(2))/(5 + sqrt(5)) + (5 + sqrt(5))/(13 + sqrt(13)) +
  (13 + sqrt(13))/(8 + sqrt(8)) + (14 + sqrt(14))/(7 + sqrt(7))
```

Кроме того, в последнем варианте выше вероятность появления ошибки при записи оператора.

Из описания функции видно, что в ней в качестве вспомогательной используется стандратная функция sqrt. Так вот, в собственной функции в качестве вспомогательной может быть использована эта же самая функция (такое использование и является примером рекурсии, обращению функции к самой себе). В качестве примера приведем функцию для расчета факториала натурального числа n (факториал числа n, обозначаемый $n!$, равен $1 \times 2 \times 3 \times \ldots \times n$). Такую функцию можно создать, имея в виду, что $n! = (n - 1)! \times n$. На школьном алгоритмическом языке соответствующая функция имеет вид:

[1] От лат. *recursio* – возвращение.

```
алг цел Факториал(цел n)
нач
  знач := Факториал(n - 1) * n     | Рекурсивный вызов функции Факториал
  | Служебное слово знач означает "значение функции"
кон
```

На самом деле эта функция оформлена не по правилам, и при выполнении программы появится сообщение об ошибке. Дело в том, что при каждом вызове функции для нее отводится место в оперативной памяти, которое освобождается после завершения ее (функции) работы. Но если во вспомогательной функции имеется рекурсия, то вызовы функции будут продолжаться до тех пор, пока место в памяти не будет исчерпано. Чтобы устранить этот недостаток, необходимо так оформлять функции, чтобы рекурсивные вызовы осуществлялись по условию, которое когда-то станет ложным. Например, для приведенной функции это условие записывается так:

```
алг цел Факториал(цел n)
нач
  если n > 1
    то
      | Рекурсивный вызов функции Факториал
      знач := Факториал(n - 1) * n
    иначе | Значение функции известно (рекурсивного вызова нет)
      знач := 1
  все
кон
```

или

```
алг цел Факториал(цел n)
нач
  если n = 1
    то
      знач := 1   | Известное значение функции
    иначе         | Рекурсивный вызов функции Факториал
      знач := Факториал(n - 1) * n
  все
кон
```

Алгоритм вычисления значения функции F(n), использующий рекурсию, может быть описан по-другому – с указанием двух или более вариантов значения, например, в виде:

```
F(2) = 10
F(n) = F(n - 1) * 3 при при n > 2
```

или

```
F(3) = 5
F(n) = F(n - 1) + 5 при n > 3 и n, кратном 3
F(n) = F(n - 1) - 3 при n > 3 и n, не кратном 3
```

или

```
F(12) = 15
F(n) = F(n + 1) - 1 при n < 12
```

или

```
F(12) = 25
F(13) = 7
F(n) = F(n + 1) - 1 при n < 12
```

и т. п.

Видно, что в двух первых случаях рекурсивные вызовы осуществляются с меньшими значениями аргумента, в остальных – с большими. Во всех случаях когда-то рекурсивные вызовы прекратятся (убедитесь в этом!), и начнется передача результатов в вызывающую функцию.

Пример работы программы при вычислении значения факториала числа 4 показан на рис. 2.6.1–2.6.2.

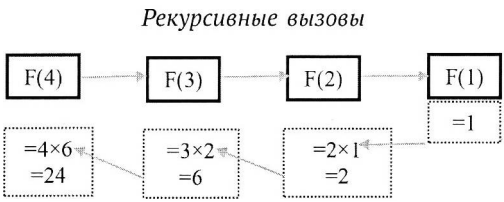


Рис. 2.6.1

F(n)	F(4)	F(3)	F(2)	F(1)
Значение функции	24 ← 4 × 6	6 ← 3 × 2	2 ← 2 × 1	1
n	4	3	2	1

Рис. 2.6.2

Ответ: 24.

Рассмотрим задачу, связанную с ЕГЭ косвенно, но очень показательную: «Определить 50-й член последовательности Фибоначчи (последовательность Фибоначчи образуют числа 1, 1, 2, 3, 5, 8, 13 ...)».

Достаточно очевидный метод решения задачи такой – использовать массив из 50 элементов, записать в него значения двух первых элементов (равных 1), а затем последовательно вычислять остальные как сумму двух предшествующих значений:

```
алг Расчет
нач цел таб мас[1 : 50], цел к
  мас[1] := 1
  мас[2] := 1
  нц для к от 3 до 50
    мас[к] := мас[к - 2] + мас[к - 1]
  кц
  вывод мас[50]
кон
```

Недостатком такого метода является то, что, хотя нас интересует лишь один член последовательности, в процессе выполнения программы компьютер вычисляет и хранит в памяти все члены последовательности от первого до 50-го.

Этого недостатка лишен второй метод вычислений – без использования массивов. Действительно, если для вычисления очередного члена последовательности Фибоначчи нужно знать только значения двух предыдущих, то можно не запоминать все члены последовательности в массиве, а хранить две величины:

- пред – элемент, предшествующий очередному члену последовательности;
- предпред – элемент, предшествующий элементу пред.

Кроме них, используем в программе величину очер – очередной рассчитываемый элемент последовательности.

Сначала имеем:

```
пред := 1
предпред := 1
```

Затем рассчитываем очередной (третий) элемент:

```
след := пред + предпред,
```

после чего «готовимся» к расчету следующего элемента:

```
предпред := пред  | Именно в
пред := след      | таком порядке!
```

– и определяем его:

```
очер := пред + предпред
```

и т. д.

Ясно, что для определения 50-го члена последовательности необходимо провести 48 повторений описанных действий. Итак, программа:

```
алг Расчет_2
нач очер, пред, предпред, к
  пред := 1
  пред пред := 1
  нц для к от 1 до 48
    очер := пред + предпред
    предпред := пред
    пред := очер
  кц
  вывод очер
кон
```

Для решения обсуждаемой задачи может быть использована рекурсия. Вот как просто и логично выглядит рекурсивный вариант функции:

```
алг цел Фиб(цел к)
нач
  если к = 1 или к = 2
    то
      знач := 1
    иначе
      знач := Фиб(к - 2) + Фиб(к - 1)
  все
кон
```

Такое оформление функции полностью соответствует закону построения последовательности Фибоначчи – очередной элемент последовательности равен сумме двух предыдущих. При нем не требуется применять оператор цикла и думать над последовательностью расчета значений пред и предпред.

Обращаем внимание на то, что в созданной функции имеют место два рекурсивных вызова.

Разработаем рекурсивные функции для случаев, когда функция задана с указанием двух или более вариантов значения (см. выше).

Задача 1

Функция F(n), где n – натуральное число, задана следующими соотношениями:

```
F(1) = 10
F(n) = F(n - 1) + 3 при n > 1.
```

Чему равно значение функции F(10)?

Решение

Соответствующая функция в программе, использующая рекурсию, оформляется следующим образом:

```
алг цел F(цел n)
нач
  если n <= 2
    то
      знач := 10
    иначе
      знач := F(n - 1) * 3
  все
кон
```

а основная программа для нахождения искомого значения:

```
алг цел F(цел n)
нач
  вывод F(10)
кон
```

Задача 2

Алгоритм вычисления значения функции F(n), где n – натуральное число, задан следующими соотношениями:

```
F(3) = 5
F(n) = F(n - 1) + 5 при n > 3 и n, кратном 3
F(n) = F(n - 1) - 3 при n > 3 и n, не кратном 3.
```

Чему равно значение F(16)?

Вся программа решения:

```
алг цел F(цел n)
нач
  вывод F(10)
кон
алг цел F (цел n)
нач
  если n = 3
    то
      знач := 5
    иначе
```

```
      если mod(n, 3) = 0
        то
          знач := F(n - 1) + 5
        иначе
          знач := F(n - 1) - 3
      все
  все
кон
```

Задания для самостоятельной работы

1. Алгоритм вычисления значения функции F(n), где n – натуральное число, задан следующими соотношениями:

```
F(12) = 15
F(n) = F(n + 1) - 1 при n < 12.
```

Чему равно F(0)?

2. Алгоритм вычисления значения функции F(n), где n – натуральное число, задан следующими соотношениями:

```
F(12) = 25
F(13) = 7
F(n) = F(n + 1) - 1 при n < 12.
```

Чему равно F(7)?

2.7. Задания 17

В Спецификации [13] применительно к заданию 17 в качестве проверяемого элемента содержания указано: «Умение создавать собственные программы (20–40 строк) для обработки целочисленной информации».

В общем виде задание формулируется так: «Рассматривается множество целых чисел, принадлежащих отрезку [...; ...], которые <*условие, связанное со значением чисел*>. Определить[1]:

- количество таких чисел и максимальное из них. В ответе записать два числа через пробел: сначала количество, затем максимальное число (обозначим такой тип задач – "тип 1");
- количество таких чисел и минимальное из них. В ответе за-

[1] Указывается один или несколько перечисленных вариантов.

писать два числа через пробел: сначала количество, затем минимальное число (“тип 2”);
- сумму таких чисел и максимальное из них. В ответе записать два числа через пробел: сначала сумму, затем максимальное число. Гарантируется, что искомая сумма не превосходит ... (“тип 3”);
- среднее арифметическое таких чисел. Ответ записать с... значащими цифрами после запятой (“тип 4”);
- разность между максимальным и минимальным из них (“тип 5”);
- медиану[1] таких чисел (“тип 6”)».

Пример задачи типа 1. Рассматривается множество целых чисел, принадлежащих отрезку [792; 6737], которые делятся на 5 и не делятся на 11, 17, 19 и 23. Найти количество таких чисел и максимальное из них. В ответе записать два числа через пробел: сначала количество, затем максимальное число.

Пример задачи типа 2. Рассматривается множество целых чисел, принадлежащих отрезку [1067; 8537], которые делятся на 3 и не делятся на 13, 21, 29 и 31. Найти количество таких чисел и минимальное из них. В ответе записать два числа через пробел: сначала количество, затем минимальное число.

Общая схема программы решения задач двух указанных типов следующая:

1) при поиске максимального числа:

```
1. Присваивание переменной-счетчику начального значения
2. цикл для каждого числа из заданного диапазона в порядке возрастания
     если указанное в задаче условие соблюдается
       то
         Увеличить значение переменной-счетчика на 1
         Принять в качестве максимального значения текущее число
     все
   конец цикла
3. Вывод ответов
```

где переменная-счетчик – переменная, соответствующая искомому количеству чисел.

Обратим внимание на то, что находить максимальное значение так, как описывалось в п. 1.4.1:

[1] О понятии «медиана» – см. в конце раздела 1.1.

```
1. Присваивание переменной-счетчику начального значения
2. Присваивание начального значения переменной,
   соответствующей искомому максимальному значению
3. цикл для каждого числа из заданного диапазона в порядке возрастания
     если указанное в задаче условие соблюдается
       то
         Увеличить значение переменной-счетчика на 1
         Сравнить текущее число с максимальным значением
         если оно больше
           то
             Принять в качестве максимального значения текущее число
         все
    все
   конец цикла
4. Вывод ответов
```

– необходимости нет. Поскольку числа рассматриваются в порядке их возрастания, то последнее найденное число и будет искомым результатом;

2) при поиске минимального числа:

```
1. Присваивание переменной-счетчику начального значения
2. цикл для каждого числа из заданного диапазона в порядке убывания
     если указанное в задаче условие соблюдается
       то
         Увеличить значение переменной-счетчика на 1
         Принять в качестве минимального значения текущее число
     все
   конец цикла
3. Вывод ответов
```

Здесь также использована особая методика нахождения минимального значения.

В программе решения задачи примера задачи типа 1 используем следующие переменные величины:

- n – исследуемые натуральные числа;
- кол – количество чисел, удовлетворяющих условию;
- макс – максимальное из чисел, удовлетворяющих условию.

Прежде чем приводить программу, напомним (см. п. 1.1.9), что проверить кратность одного целого числа другому можно с помощью функции mod (в программе на школьном алгоритмическом языке) или с использованием операции определения остатка (в программах на других языках).

Вся программа имеет вид:

```
алг Пример_задачи_типа_1
нач цел n, кол, макс
  кол := 0  | Начальное значение переменной-счетчика
  нц для n от 792 до 6737  | Рассматриваем все числа диапазона
                           | в порядке возрастания
    | Проверяем условие
    если mod(n, 5) = 0 и mod(n, 11) > 0 и mod(n, 17) > 0
         и mod(n, 19) > 0 и mod(n, 23) > 0
      то
        | Увеличиваем значение переменной-счетчика
        кол := кол + 1
        | Принимаем текущее значение n в качестве максимального числа
        макс := n
    все
  кц
  | Выводим ответы
  вывод нс, кол, " ", макс
кон
```

Обратим внимание на то, что попытка сократить условие для подсчета подходящих чисел:

```
алг Пример_задачи_типа_1
нач цел n, кол, макс
  кол := 0  | Начальное значение переменной-счетчика
  нц для n от 792 до 6737  | Рассматриваем все числа диапазона
                           | в порядке возрастания
    | Проверяем условие
    если mod(n, 5) = 0 и mod(n, 11 * 17 * 19 * 23) > 0
      то
        | Увеличиваем значение переменной-счетчика
        кол := кол + 1
        | Принимаем текущее значение n в качестве максимального числа
        макс := n
    все
  кц
  | Выводим ответы
  вывод нс, кол, " ", макс
кон
```

– приведет к неправильному результату (почему – подумайте).

Программа решения примера задачи типа 2 аналогична; отличие только в том, что все числа диапазона рассматриваются от большего к меньшему:

```
алг Пример_задачи_типа_2
нач цел n, кол, мин
  кол := 0  | Начальное значение переменной-счетчика
  нц для n от 8537 до 1067 шаг -1  | Рассматриваем все числа диапазона
                                   | в порядке убывания
    | Проверяем условие
    если mod(n, 3) = 0 и mod(n, 13) > 0 и mod(n, 21) > 0
         и mod(n, 29) > 0 и mod(n, 31) > 0
      то
        | Увеличиваем значение переменной-счетчика
        кол := кол + 1
        | Принимаем текущее значение n в качестве минимального числа
        мин := n
    все
  кц
  | Выводим ответы
  вывод нс, кол, " ", мин
кон
```

Пример задачи типа 3. Рассматривается множество целых чисел, принадлежащих числовому отрезку [3294; 8199], которые делятся на 5 и не делятся на 11, 17, 19 и 23. Найдите наибольшее из таких чисел и их сумму. Гарантируется, что искомая сумма не превосходит 10^{10}.

Решение

Программа решения данной задачи отличается от программы решения задачи типа 1 тем, что вместо переменной-счетчика используется переменная-сумматор. Разработайте ее самостоятельно.

Пример задачи типа 4. Рассматривается множество целых чисел, принадлежащих отрезку [792; 6737], которые делятся на 5 и не делятся на 11, 17, 19 и 23. Определить среднее арифметическое таких чисел. Ответ записать с двумя значащими цифрами после запятой.

Решение

Так как для расчета искомого значения нужно знать не только количество указанных в условии чисел, но и их сумму, то общая схема программы решения задачи такая:

1. Присваивание переменной-счетчику и переменной-сумматору начальных значений

```
2. цикл для каждого числа из заданного диапазона в порядке
        возрастания
     если указанное в задаче условие соблюдается
       то
         Увеличить значение переменной-счетчика на 1
         Учесть текущее число в значении переменной-сумматора
     все
   конец цикла
3. Расчет среднего арифметического
4. Вывод ответа
```

где переменная-сумматор – переменная, соответствующая сумме учитываемых чисел.

Прежде чем представлять программы решения задачи, заметим, что в школьном алгоритмическом языке нет возможности так называемого форматированного вывода вещественных значений. Поэтому для вывода результата с двумя цифрами в дробной части можно применить такой прием: умножить его на 100, взять целую часть произведения (использовать функцию int()), а ее разделить на 100:

```
алг Пример_задачи_типа_4
нач цел n, кол, сумма, вещ сред_арифм
  кол := 0
  сумма := 0
  нц для n от 792 до 6737
    | Проверяем условие
    если mod(n, 5) = 0 и mod(n, 11) > 0 и mod(n, 17) > 0
         и mod(n, 19) > 0 и mod(n, 23) > 0
      то
        | Увеличиваем значение переменной-счетчика
        кол := кол + 1
        | Учитываем значение n в общей сумме
        сумма := сумма + n
    все
  кц
  | Рассчитываем и выводим ответ
  сред_арифм := сумма/кол
  вывод int(сред_арифм * 100)/100
кон
```

Пример задачи типа 5. Рассматривается множество целых чисел, принадлежащих отрезку [792; 6737], которые делятся на 5 и не делятся на 11, 17, 19 и 23. Определить разность между максимальным и минимальным из них.

Решение

Задачу можно рассматривать как «сумму» задач типа 1 и типа 2, т. е. решить ее так:

1) рассмотрев все числа отрезка в порядке возрастания, найти максимальное из соответствующих условию;
2) рассмотрев все числа в порядке убывания, найти минимальное;
3) определить разность найденных экстремумов.

Можно также найти искомое значение, так сказать, «за один проход» (при однократном рассмотрении всех чисел заданного отрезка) на основе следующих рассуждений.

1. Рассмотреть все числа отрезка в порядке возрастания.
2. Первое число, соответствующее условию, принять в качестве минимального.
3. Максимальное число найти, как в задаче типа 1 (это будет последнее из подходящих чисел отрезка).
4. Определить разность найденных экстремумов.

Зафиксировать факт нахождения первого числа, соответствующего условию, можно, используя переменную логического типа. В приведенной ниже программе ее имя – мин_найден. Остальные особенности программы описаны в комментариях.

```
алг Пример_задачи_типа_5
нач цел n, макс, мин, лог мин_найден
  мин_найден := нет          |
  нц для n от 792 до 6737    | Рассматриваем все числа диапазона
                             | в порядке возрастания
    | Проверяем условие
    если mod(n, 5) = 0 и mod(n, 11) > 0 и mod(n, 17) > 0
         и mod(n, 19) > 0 и mod(n, 23) > 0
      то  | Число подходит
        если не мин_найден | Если это первое подходящее число
          то
            | Принимаем его в качестве минимального числа
            мин := n
            | Фиксируем факт нахождения минимального числа
            мин_найден := да
          иначе | Это одно из остальных подходящих чисел
            | Принимаем его в качестве максимального числа
            макс := n
        все
    все
```

```
  кц
  | Выводим ответ
  вывод макс - мин
кон
```

В программе на языке Python удобно все подходящие по условию числа записывать в список (используя метод append), а затем определить искомую разность с помощью функций max и min:

```
mas = []
for n in range(792, 6738):
  if n % 5 == 0 and n % 11 > 0 and n % 17 > 0 and n % 19 > 0
        and n % 23 > 0:
    mas.append(n)
print(max(mas) - min(mas))
```

Пример задачи типа 6. Рассматривается множество целых чисел, принадлежащих отрезку [792; 6737], которые делятся на 5 и не делятся на 11, 17, 19 и 23. Определить медиану чисел этого множества.

Решение

Так как понадобятся все значения чисел, указанных в условии, то здесь без массива не обойтись. Поскольку количество чисел, удовлетворяющих условию, неизвестно, размер этого массива должен быть принят «с запасом». Можно принять размер массива равным, например, трети разности значений на границах отрезка. Прежде чем представлять программу решения задачи, обратим внимание на то, что числа в массив записываются в порядке возрастания, что облегчает поиск медианы.

```
алг Пример_задачи_типа_5
нач цел n, кол, вещ медиана, цел таб мас[1 : div(6737 - 792, 3)]
  кол := 0
  нц для n от 792 до 6737
    | Проверяем условие
    если mod(n, 5) = 0 и mod(n, 11) > 0 и mod(n, 17) > 0
          и mod(n, 19) > 0 и mod(n, 23) > 0
      то | Встретилось очередное подходящее число
        кол := кол + 1 | Его номер
        мас[кол] := n  | Записываем число в массив
    все
  кц
  если mod(кол, 2) = 1 | Размер массива - нечетный
    то
```

```
      медиана := мас[div(кол, 2) + 1]
    иначе
      медиана := (мас[div(кол, 2)] + мас[div(кол, 2) + 1])/2
  все
  | Выводим ответ
  вывод медиана
кон
```

Язык Python

```
mas = []
for n in range(792, 6738):
  if n % 5 == 0 and n % 11 > 0 and n % 17 > 0 and n % 19 > 0
       and n % 23 > 0:
    mas.append(n)
if len(mas) % 2 == 1:
  mediana = mas[len(mas)//2]
else:
  mediana = (mas[len(mas)//2 - 1] + mas[len(mas)//2])/2
print(mediana)
```

Задачи для самостоятельной работы

См. [16], файл kege4.doc.

Разработайте также программы, в которых применительно к условиям приведенных в [16] задач определяется:

- среднее арифметическое таких чисел (ответ должен быть записан с тремя значащими цифрами после запятой);
- разность между максимальным и минимальным из них;
- медиана указанного в условии множества чисел.

В заключение обратим внимание на то, что во всех перечисленных выше задачах в условии фигурировала кратность или «некратность» чисел. Возможны также более сложные условия. Приведем примеры и комментарии к решению соответствующих задач.

1. «... сумма цифр которых больше (или меньше) [*указывается значение*]».

Здесь возникает задача расчета суммы всех цифр числа (см. задачу 1.1.3) и сравнение ее с заданным в условии значением.

2. «... сумма цифр которых кратна [*указывается значение*]».

После нахождения суммы всех цифр числа проверяется остаток от ее деления на значение, заданное в условии.

3. «... произведение цифр которых больше (или меньше) [*указывается значение*]».

4. «... произведение цифр которых кратно [*указывается значение*]».

Отличие программ решения двух последних задач от задач с условием в примерах 1 и 2 в том, что рассчитывается не сумма, а произведение цифр числа (см. п. 1.1.4).

5. «... одна из цифр которых равна [*указывается значение*]».

Здесь для каждого числа при выделении его цифр следует проверять каждую цифру на равенство заданному значению. Факт встречи нужной цифры можно зафиксировать с помощью величины логического типа (или принимающей значение 0 и 1) и в зависимости от окончательного значения этой величины учитывать текущее число в количестве всех таких чисел, их сумме и т. п.

6. «... двоичная запись которых оканчивается (или не оканчивается) на 0 (или 00, или 000, или т. п.)» [12][1].

При решении таких задач нет необходимости переводить числа в двоичную систему счисления. Можно учесть, что двоичная запись десятичного числа оканчивается на 0, если это число четное, на 00 – если оно кратно четырем и т. д. Это значит, что достаточно проверять кратность или «некратность» проверяемых чисел.

7. «... запись которых в двоичной системе оканчивается на 11».

Анализ двоичных чисел, оканчивающихся на 11 (11, 111, 1011, 1111 и т. д.), показывает, что это десятичные числа 3, 7, 11, 15 и т. д., общее свойство которых такое: «остаток от их деления на 4 равен 3». Это свойство и следует использовать в программе в условии для учета исследуемых целых чисел.

8. «... запись которых в девятеричной (или другой) системе оканчивается на [*указывается цифра*]».

Здесь также нет необходимости переводить числа в девятеричную (или другую) систему счисления. Нужная последняя цифра равна остатку от деления проверяемого числа на 9 (или на основание другой системы) – вспомните метод перевода десятичных чисел в другие системы счисления методом деления числа на основание «новой» системы.

9. «... для которых запись в двоичной и четверичной системах счисления оканчивается одинаковой цифрой».

---

[1] Автор задач, аналогичных задачам примеров 5–10, – А. Куканова.

Для данного условия свойства соответствующих десятичных чисел можно получить на основе следующих рассуждений.

Возможная одинаковая цифра в условии – 0 или 1.

Двоичная запись десятичного числа оканчивается на 0, если это число четное, четверичная запись – если оно кратно четырем. Общий признак, дающий цифру 0, – число кратно четырем (значения 6, 10 и т. п. не подходят).

И двоичная, и четверичная записи десятичного числа оканчиваются на 1, если это число нечетное (убедитесь в этом!).

Это значит, что условие для учета соответствующих десятичных чисел можно сформулировать так: «число кратно четырем или нечетное».

10. «... запись которых в пятеричной системе имеет не менее 6 цифр и оканчивается на 21 или 23» (или с другими значениями).

Здесь рассмотрим два признака.

Во-первых, можно определить, какое минимальное десятичное число в пятеричной системе является шестизначным. Максимальное пятеричное пятизначное число – 44444. Соответствующее десятичное – 3124. Значит, уже число 3125 будет в пятеричной системе шестизначным. Именно с него и должен начинаться поиск подходящих чисел.

Далее, какие десятичные числа в пятеричной системе оканчиваются на 21? Ответ – у которых остаток от деления на 25 равен 11 (убедитесь в этом!). Аналогично для числа 23 – у которых остаток от деления на 25 равен 13. Эти два признака должны быть объединены с помощью дизъюнкции.

Возможны и комбинации перечисленных условий и условия кратности или «некратности» чисел из обрабатываемого диапазона.

Дополнение

Возможны также задачи, в которых требуется исследовать все цифры десятичных чисел. Обсудим методику их решения.

В общем виде смысл задачи можно сформулировать так[1]: «Определить количество ...-значных натуральных чисел, у которых *<условие, связанное со значениями цифр>*».

Пример 1. Найти количество шестизначных натуральных чисел, у которых суммы трех их первых и трех последних цифр равны.

---

[1] В условии могут фигурировать также «счастливые числа», «простые числа», «совершенные числа» и т. п.

Пример 2. Найти количество четырехзначных натуральных чисел, у которых суммы двух их первых и двух последних цифр отличаются не более чем на 2.

Пример 3. Назовем натуральное шестизначное число N счастливым, если сумма двух его первых цифр больше, чем сумма двух средних цифр, которая в свою очередь больше, чем сумма двух последних цифр. Найти количество таких чисел.

Пример 4. Назовем натуральное пятизначное число N счастливым, если его цифры записаны в порядке неубывания. Найти количество таких чисел.

Общая схема программы решения задач указанного типа следующая:

```
1. Присваивание переменной-счетчику начального значения
2. цикл для каждого числа из заданного диапазона
     Выделение цифр числа
     если указанное в задаче условие соблюдается
       Увеличить значение переменной-счетчика на 1
     все
   конец цикла
3. Вывод ответа
```

где переменная-счетчик – переменная, соответствующая искомому количеству.

Частная задача выделения отдельных цифр натурального числа была рассмотрена в п. 1.1.2.

В программе решения задачи примера 1 используем следующие переменные величины:

- n – отдельное исследуемое шестизначное число;
- цифра1, цифра2... цифра6 – отдельные цифры этого числа;
- кол – искомое значение количества чисел (переменная-счетчик).

Программа:

```
алг Задача_примера_1
нач цел n, цифра1, цифра2, цифра3, цифра4, цифра5, цифра6, кол
  кол := 0
  нц для n от 100000 до 999999  | Рассматриваем все 6-значные числа
    | Выделяем кажду цифру (см. п 1.1.1)
    цифра6 := mod(n, 10); n := div(n, 10)
    цифра5 := mod(n, 10); n := div(n, 10)
    цифра4 := mod(n, 10); n := div(n, 10)
```

```
    цифра3 := mod(n, 10); n := div(n, 10)
    цифра2 := mod(n, 10); n := div(n, 10)
    цифра1 := mod(n, 10)
    | Сравниваем суммы
    если цифра1 + цифра2 + цифра3 = цифра4 + цифра5 + цифра6
      то
        кол := кол + 1
    все
  кц
  вывод нс, кол
кон
```

Язык Python

```
kol = 0
for n in range(100000, 1000001):
  tsifra6 = n % 10; n = n//10
  tsifra5 = n % 10; n = n//10
  tsifra4 = n % 10; n = n//10
  tsifra3 = n % 10; n = n//10
  tsifra2 = n % 10; n = n//10
  tsifra1 = n % 10
  if tsifra1 + tsifra2 + tsifra3 == tsifra4 + tsifra5 + tsifra6:
    kol = kol + 1
 print(kol)
```

В программе на языке Паскаль в теле оператора цикла с параметром величину, являющуюся параметром цикла, изменять нельзя, поэтому приходится использовать переменную, хранящую копию значения числа n (ее имя в приведенной ниже программе – kopia_n).

```
var n, tsifra1, tsifra2, …, tsifra1, kopia_n, kol: longint;
BEGIN
  kol := 0;
  for n := 100000 to 999999 do
    begin
      kopia_n := n;
      tsifra6 := kopia_n mod 10; kopia_n := kopia_n div 10;
      tsifra5 := kopia_n mod 10; kopia_n := kopia_n div 10;
      …
      tsifra1 := kopia_n mod 10;
      if tsifra1 + tsifra2 + tsifra3 = tsifra4 + tsifra5 + tsifra6 then
        kol := kol + 1
```

```
    end;
  writeln(kol)
END.
```

Когда выделяемых цифр много, целесообразно предварительно записать их в массив (см. п. 1.1.6). Например, для задачи с восьмизначными числами: «Найти количество восьмизначных натуральных чисел, у которых суммы четырех их первых и четырех последних цифр равны» – программа будет иметь вид:

```
алг
нач цел n, k, кол, цел таб цифры[1 : 8]
  кол := 0
  нц для n от 10000000 до 99999999
    | Записываем цифры числа n в массив
    нц для k от 1 до 8
      цифры[k] := mod(n, 10)
      n := div(n, 10)
    кц
    если цифры[1] + цифры[2] + цифры[3] + цифры[4] =
         цифры[5] + цифры[6] + цифры[7] + цифры[8]
      то
        кол := кол + 1
    все
  кц
  вывод нс, кол
кон
```

В ней использован массив из восьми элементов. Как отмечалось в п. 1.1.6, в результате заполнения этого массива в первом элементе массива записана младшая (последняя) цифра. В данной задаче это значения не имеет (сравниваются суммы цифр в двух «половинах» числа).

Язык Python

```
kol = 0
for n in range(10000000, 100000000):
  tsifri = [0 for i in range(8)]
  for k in range(8):
    tsifri[k] =  n % 10
    n = n//10
  if tsifri[0] + tsifri[1] + tsifri[2] + tsifri[3] == _
        tsifri[4] + tsifri[5] + tsifri[6] + tsifri[7]:
```

```
    kol = kol + 1
print(kol)
```

Вариант программы с методом append:

```
kol = 0
for n in range(10000000, 100000000):
  tsifri = []  # Именно здесь!
  for k in range(8):
    tsifri.append(n % 10)
    n = n//10
  if tsifri[0] + tsifri[1] + tsifri[2] + tsifri[3] == _
      tsifri[4] + tsifri[5] + tsifri[6] + tsifri[7]:
    kol = kol + 1
print(kol)
```

Обратим внимание на важное обстоятельство. В задачах, аналогичных примеру 2, следует использовать абсолютную величину разности сумм, так как сумма первых двух цифр может быть как больше, так и меньше, чем сумма двух последних цифр:

```
если iabs((цифра1 + цифра2) - (цифра3 + цифра4)) <= 2
  то
    ...
```

где функция школьного алгоритмического языка iabs возвращает модуль (абсолютное значение) своего аргумента целого типа.

Заметим также, что в данном случае при использовании функции iabs в условии указывается нестрогое неравенство (в других задачах, возможно, это и не так).

Обсудим решение примера 4.

Как определить, что все цифры некоторого числа n записаны в порядке неубывания? Проще всего это сделать, подсчитав количество цифр, не меньше предыдущих, начиная со второй цифры. Если это количество равно 4, то число n – счастливое. В приведенной ниже программе использованы переменные величины-счетчики:

- кол_не_меньше – количество цифр числа, не меньше предыдущих;
- кол_счаст – искомое количество счастливых чисел.

```
алг Задача_примера_4
нач цел n, кол_счаст, цифра1, цифра2, цифра3, цифра4, цифра5,
        кол_не_меньше
  кол_счаст := 0
```

```
  нц для n от 10000 до 99999
    | Выделяем цифры
    цифра5 := mod(n, 10); n := div(n, 10)
    цифра4 := mod(n, 10); n := div(n, 10)
    …
    цифра1 := mod(n, 10)
    | Определяем значение переменной кол_не_меньше
    кол_не_меньше := 0   | Именно здесь!
    если цифра2 >= цифра1
      то
        кол_не_меньше := кол_не_меньше + 1
    все
    если цифра3 >= цифра2
      то
        кол_не_меньше := кол_не_меньше + 1
    все
    …
    если цифра5 >= цифра4
      то
        кол_не_меньше := кол_не_меньше + 1
    все
    | Проверяем на "счастливость"
    если кол_не_меньше = 4
      то
        кол_счаст := кол_счаст + 1
    все
  кц
  вывод нс, кол_счаст
кон
```

Конечно, при записи цифр в массив лучше использовать оператор цикла:

```
алг Задача_примера_4
нач цел n, кол_счаст, кол_не_меньше, инд, k, цел таб цифры[1 : 5]
  кол_счаст := 0
  нц для n от 10000 до 99999
    | Записываем цифры числа n в массив
    нц для k от 5 до 1 шаг -1
      цифры[k] := mod(n, 10)
      n := div(n, 10)
    кц
    | Определяем значение переменной кол_не_меньше
    кол_не_меньше := 0
    нц для инд от 2 до 5                | Начиная со 2-й цифры,
```

```
      если цифры[инд] >= цифры[инд - 1] | сравниваем с предыдущей
        то
          кол_не_меньше := кол_не_меньше + 1
      все
    кц
    | Проверяем на "счастливость"
    если кол_не_меньше = 4
      то
        кол_счаст := кол_счаст + 1
    все
  кц
  вывод нс, кол_счаст
кон
```

Обратим внимание на то, что в приведенной программе цифры записываются в массив в порядке их следования в числе.

Более сложным вариантом задач рассматриваемого типа являются задачи, в которых используются особенности цифр, представленных в недесятичной записи натурального числа.

Пример 5. Назовем натуральное число N ($10000_8 \le N \le 77777_8$) счастливым, если суммы двух первых и двух последних цифр его восьмеричной записи различаются не больше чем на 2. Найти количество таких чисел.

Решение

Отличие решения в том, что следует получить цифры каждого из натуральных чисел в их восьмеричной записи. Такая задача отличается от рассмотренной в п. 1.1.6 тем, что при выделении восьмеричных цифр нужно вместо числа 10 использовать 8 (основание новой системы) [15]. Кроме того, здесь важно определить диапазон проверяемых натуральных чисел. Для рассматриваемой задачи – это числа от 4096 до 32767 ($10000_8 = 4096_{10}$, $77777_8 = 32767_{10}$).

Вся программа имеет вид:

```
алг Задача_примера_5
нач цел n, к, кол, цел таб цифры[1 : 5]
  кол := 0
  нц для n от 4096 до 32767
    | Переводим число в восьмеричную систему,
    | записывая восьмеричные цифры числа в массив
    нц для к от 5 до 1 шаг -1
      цифры[к] := mod(n, 8)
      n := div(n, 8)
    кц
```

```
    | Проверяем (абсолютную величину разности сумм цифр)
    если iabs(цифры[1] + цифры[2] - цифры[4] - цифры[5]) <= 2
      то
        кол := кол + 1
    все
  кц
  вывод нс, кол
кон
```

Язык Паскаль

```
Var n, kol: longint; k: byte; tsifri: array[1..5] of byte;
BEGIN
  kol := 0;
  for n := 4096 to 32767 do
    begin
      for k := 5 downto 1 do
        begin
          tsifri[k] := n mod 8;
          n := n div 8
        end;
      if abs(tsifri[1] + tsifri[2] - tsifri[4] - tsifri[5]) <= 2 then
        kol = kol + 1;
    end;
  write(kol)
END.
```

Язык Python

```
tsifri = [0 for k in range(5)]
kol = 0
for n in range(4096, 32768):
  for k in range(4, -1, -1):
    tsifri[k] = n % 8
    n = n//8
  if abs(tsifri[0] + tsifri[1] - tsifri[3] - tsifri[4]) <= 2:
    kol = kol + 1
print(kol)
```

В программе на языке Python можно не переводить восьмеричные числа 10000 и 77777 в десятичную систему вручную, как это делалось при написании программ выше. Удобно использовать следующую особенность функции `int()`. Эта функция не толь-

ко возвращает целую часть своего аргумента-числа и переводит строковое представление в целые числа, но и переводит недесятичные числа в их десятичную запись! Для такого перевода надо указать в функции `int()` два аргумента:

1) переводимое число (как строку – в кавычках);
2) основание системы, в которой оно записано.

Например, в результате выполнения инструкции

```
print(int("A1 ", 16))
```

на экран будет выведено число 161 ($A1_{16} = 161_{10}$). Видно, что цифры, большие 9, записываются латинскими буквами (от A до Z). Основание (второй аргумент) может быть больше либо равно 2 и меньше либо равно 36. Например, в результате выполнения инструкции

```
print(int("ZF1 ", 36))
```

на экран будет выведено число 45901.

Учет указанной особенности на экзамене позволяет сэкономить время выполнения задания.

Например, в последней программе заголовок инструкции `for` может быть оформлен так:

```
for n in range(int("10000 ", 8), int("100000 ", 8)):
  ...
```

Обсудим также вариант задачи, в которой используются особенности цифр, представленных в недесятичной записи натурального числа, но количество цифр в этой записи или/и основание недесятичной системы счисления заранее неизвестно.

Пример 6. «Определить количество натуральных чисел из интервала от ... до ..., в записи которых в системе счисления с основанием ... суммы двух первых и двух последних цифр равны».

Решение

В таких программах нужно:

1) использовать оператор цикла с условием;
2) массив для хранения недесятичных цифр описать с запасом;
3) при записи цифр в массив вести подсчет их количества.

```
алг Задача_примера_6
нач цел n, основание, кол, цел таб цифры[1 : 100]
  ввод основание
```

```
  кол := 0
  нц для n от … до …
    кол_цифр := 0
    нц пока n > 0
      кол_цифр := кол_цифр + 1
      цифры[кол_цифр] := mod(n, основание)
      n := div(n, основание)
    кц
    если цифры[1] + цифры[2] = цифры[кол_цифр] + цифры[кол_цифр - 1]
      то
        кол := кол + 1
    все
  кц
  вывод нс, кол
кон
```

Язык Паскаль

```
Var n, kol: longint; k: byte; tsifri: array[1..100] of byte;
BEGIN
  readln(osnovanie);
  kol := 0;
  for n := … to … do
    begin
      kol_tsifr := 0;
      while n > 0: do
        begin
          kol_tsifr := kol_tsifr + 1;
            tsifri[kol_tsifr] := n mod osnovanie;
          n := n div osnovanie
        end;
      if tsifri[1] + tsifri[2] = tsifri[kol_tsifr - 1] +
              tsifri[kol_tsifr])
        then
          kol = kol + 1;
    end;
  write(kol)
END.
```

Язык Python

```
osnovanie = int(input())
tsifri = [0 for k in range(100)]
kol := 0
```

```
for n in range(…, …):
  kol_tsifr = 0
  while n > 0:
    kol_tsifr = kol_tsifr + 1
    tsifri[kol_tsifr - 1] = n % osnovanie
    n = n//osnovanie
  if tsifri[0] + tsifri[1] ==
            tsifri[kol_tsifr - 2] + tsifri[kol_tsifr - 1]:
    kol = kol + 1
print(kol)
```

При использовании в программе на языке Python метода append обратиться к двум последним элементам списка (индексы которых в данном случае неизвестны) так: `tsifri[-2]` и `tsifri[-1]`.

В заключение заметим, что возможны также варианты всех рассмотренных выше типов задач, в которых требуется определить не количество чисел, удовлетворяющих условию, а другие характеристики их множества, упоминавшиеся выше (медиану и др.).

Задачи для самостоятельной работы

См. [16], файл kege3.doc.

2.8. Задание 18

В Спецификации [13] применительно к заданию 18 в качестве проверяемого элемента содержания указано: «Умение обрабатывать вещественные выражения в электронных таблицах».

В демонстрационном варианте ЕГЭ по информатике 2021 года в задании 18 представлена такая задача: «Квадрат разлинован на N×N клеток (1 < N < 17). Исполнитель Робот может перемещаться по клеткам, выполняя за одно перемещение одну из двух команд: вправо или вниз. По команде вправо Робот перемещается в соседнюю правую клетку, по команде вниз – в соседнюю нижнюю. При попытке выхода за границу квадрата Робот разрушается. Перед каждым запуском Робота в каждой клетке квадрата лежит монета достоинством от 1 до 100. Посетив клетку, Робот забирает монету с собой; это также относится к начальной и конечной клетке маршрута Робота.

Определите максимальную и минимальную денежную сумму, которую может собрать Робот, пройдя из левой верхней клетки

в правую нижнюю. В ответе укажите два числа – сначала максимальную сумму, затем минимальную.

Исходные данные представляют собой электронную таблицу размером N×N, каждая ячейка которой соответствует клетке квадрата».

Задание выполняется с использованием прилагаемого файла.

Решение

Указанная задача является типичной задачей, решаемой методом так называемого динамического программирования – особого метода поиска оптимальных решений, специально приспособленного к так называемым многошаговым, или многоэтапным, операциям [18].

Основные положения динамического программирования изложены в приложении 1, с которым читатель должен предварительно ознакомиться. Здесь же отметим принципиальный момент, характерный для динамического программирования, – анализ начинается с конца маршрута Робота.

Опишем, как методом динамического программирования решается рассматриваемая задача применительно к поиску максимальной суммы.

Пусть квадрат, по которому перемещается Робот, состоит из 36 клеток. Квадрат будем моделировать в виде диапазона A1:F6 электронной таблицы, в ячейках которой записаны достоинства монет (см. рис. 2.8.1).

	A	B	C	D	E	F
1	13	8	20	9	20	7
2	18	9	3	10	17	14
3	8	10	16	5	2	19
4	11	9	10	4	2	5
5	15	13	13	2	7	6
6	7	4	13	5	2	3

Рис. 2.8.1

Понятно, из ячейки F5 к цели (в ячейку F6) можно попасть единственным способом – по команде вниз. При этом общая сумма, собранная Роботом, составит 6 + 3 = 9. Конечно, расчеты удобнее проводить с помощью электронной таблицы в диапазоне A8:F13 (рис. 2.8.2).

	A	B	C	D	E	F
1	13	8	20	9	20	7
...						
6	7	4	13	5	2	3
7						
8						
9						
10						
11						
12						
13						

Рис. 2.8.2

Запишем в ячейку F13 значение 3 (или, для общего случая, формулу =F6), а в ячейку F12 – формулу =F5 + F13. При этом, естественно, на листе появятся результаты расчета по формулам (см. рис. 2.8.3).

Аналогично из всех остальных клеток крайнего правого столбца квадрата имеется только один вариант перемещения. Это значит, что можем формулу в ячейке F12 распространить (скопировать) на остальные ячейки столбца F.

Для клеток последней строки квадрата также возможен единственный вариант движения Робота (вправо). Это означает, что в ячейку E13 можно ввести формулу =E6 + F13, которую затем распространить (скопировать) на ячейки A13:D13.

	A	B	C	D	E	F
...						
8						54
9						47
10						33
11						14
12						9
13	34	27	23	10	5	3

Рис. 2.8.3

Итак, мы знаем, какая сумма может быть собрана Роботом при движении из клеток, моделируемых на рис. 2.8.3 ячейками с уже рассчитанными значениями. В дальнейшем для удобства будем говорить об условном перемещении исполнителя по ячейкам электронной таблицы.

Сложнее с выбором решения о направлении перемещения из ячейки E12. Здесь возможные два варианта: вправо (в ячейку F12) и вниз (в ячейку E13). Какой из них лучше? Так как мы уже знаем, что из ячейки F12 до конца маршрута Робот может собрать 9, а из ячейки E13 – 5, то понятно, что следует идти вправо. В результате сумма при движении из E12 в F13 будет равна 16. В общем случае формула для определения максимальной суммы для ячейки E12 будет такой:

=ЕСЛИ(F12 > E13; E5 + F12; E5 + E13).

Эта формула может быть скопирована в остальные ячейки диапазона A8:E12. В результате фрагмент листа A8:F13 примет вид:

	A	B	C	D	E	F
...						
8	134	121	113	93	84	54
9	104	86	77	74	64	47
10	83	72	62	40	35	33
11	75	58	46	22	18	14
12	64	49	36	18	16	9
13	34	27	23	10	5	3

Рис. 2.8.4

Это значит, что искомая максимальная сумма равна 134.

Задания для самостоятельной работы

1. Определите, как должен перемещаться Робот, чтобы собрать максимальную сумму для рассмотренного случая.

2. Оформите лист электронной таблицы для нахождения минимальной суммы, собранной Роботом. Определите, как должен перемещаться Робот в этом случае.

3. Квадрат разлинован на 6×6 клеток. Исполнитель Робот может перемещаться по клеткам, выполняя за одно перемещение одну из двух команд: вправо или вверх. По команде вправо Робот перемещается в соседнюю правую клетку, по команде вверх – в со-

седнюю верхнюю. При попытке выхода за границу квадрата Робот разрушается. Перед каждым запуском Робота в каждой клетке квадрата лежит монета достоинством, указанным на рис. 2.8.1. Посетив клетку, Робот забирает монету с собой; это также относится к начальной и конечной клетке маршрута Робота.

Определите максимальную денежную сумму, которую может собрать Робот, пройдя из левой нижней клетки в правую верхнюю».

4. «Квадрат разлинован на 6×6 клеток. Исполнитель Робот может перемещаться по клеткам, выполняя за одно перемещение одну из трех команд: вправо, вниз или вправовниз. По команде вправо Робот перемещается в соседнюю правую клетку, по команде вниз – в соседнюю нижнюю, по команде вправовниз – по диагонали в соседнюю нижнюю правую. При попытке выхода за границу квадрата Робот разрушается. Перед каждым запуском Робота в каждой клетке квадрата лежит монета достоинством, как на рис. 2.8.1. Посетив клетку, Робот забирает монету с собой; это также относится к начальной и конечной клетке маршрута Робота.

Определите минимальную денежную сумму, которую может собрать Робот, пройдя из левой верхней клетки в правую нижнюю.

5. На рисунке изображен треугольник из чисел. Найдите максимальную сумму чисел, расположенных на пути, начинающемся в верхней точке треугольника и заканчивающемся на его основании:

```
                8
            5       7
        8       2       1
    2       7       5       5
7       10      1       9       7
```

Оформите также лист электронной таблицы для решения подобных задач, когда в основании числового треугольника записано 10 чисел.

2.9. Задание 22

В качестве проверяемого элемента содержания для задания 22 в Спецификации [13] указывается: «Анализ алгоритма, содержащего цикл и ветвление».

В данном разделе рассмотрим решения заданий из демонстрационных вариантов нескольких последних лет, аналогичных заданию 22 в [1].

Прежде чем представлять задания, напомним (см. раздел 1.1), что для заданного натурального числа x действия в цикле

```
нц пока x > 0
  ...
  x := div(x, 10)
кц
```

– это многократное «отбрасывание» последней цифры числа x, а операция

```
mod(x, 10)
```

определяет его последнюю цифру.

2.9.1. Задание из [6]

Условие

Ниже на четырех языках программирования записан алгоритм. Получив на вход натуральное число x, этот алгоритм печатает два числа: a и b. Укажите наименьшее из таких чисел x, при вводе которых алгоритм печатает сначала 2, а потом 15.

Алгоритмический язык

```
алг
нач цел x, a, b
  ввод x
  a := 0; b := 1
  нц пока x > 0
    a := a + 1
    b := b * mod(x, 10)
    x := div(x, 10)
  кц
  вывод a, нс, b
кон
```

Паскаль

```
var x, d, R: longint;
begin
  readln(x);
  a := 0; b := 1;
  while x > 0 do
    begin
      a := a + 1;
      b := b * (x mod 10);
      x := x div 10
    end;
  writeln(a); writeln(b);
end.
```

Python

```
x = int(input())
a = 0
b = 1
while x > 0:
  a = a + 1
  b = b * (x % 10)
  x = x//10
print(a)
print(b)
```

Си

```
#include<stdio.h>
int main(void)
{
  int x, a, b;
  scanf("%d", &x);
  a = 0;
  b = 1;
```

Python

Си

```
#
  while (x > 0){
    a = a + 1;
    b = b * (x % 10);
    x = x/10;
  }
  printf("%d\n%d", a, b);
}
```

Решение

Анализ показывает, что переменная a – это количество цифр в числе x (см. задачу 1.1.5), а переменная b – их произведение (см. задачу 1.1.4). Так как по условию a = 2, b = 15, то искомым числом (двузначным с произведением цифр, равным 15) является 35.

Ответ: 35.

2.9.2. Задание из [9]

Условие

Ниже на пяти языках программирования записан алгоритм. Получив на вход натуральное число x, этот алгоритм печатает число R. Укажите такое число x, при вводе которого алгоритм печатает двузначное число, сумма цифр которого равна 16. Если таких чисел x несколько, укажите наименьшее из них.

Алгоритмический язык

```
алг
нач цел x, d, R
  ввод x
  R := 0
  нц пока x > 0
    d := mod(x, 10)
    R := 10 * R + d
    x := div(x, 10)
  кц
  вывод R
кон
```

Паскаль

```
var x, d, R: longint;
begin
  readln(x);
  R := 0;
  while x > 0 do
    begin
      d := x mod 10;
      R := 10 * R + d;
      x := x div 10
    end;
  writeln(R)
end.
```

Python

```
x = int(input())
R = 0
while x > 0:
  d = x % 10
  R = 10 * R + d
  x = x//10
print(R)
```

Си

```
#include<stdio.h>
int main()
{
  long x,d,R;
  scanf("%ld", &x);
  R = 0;
  while (x > 0)
    {
      d = x % 10;
      R = 10 * R + d;
      x = x/10;
    }
  printf("%ld", R);
  return 0;
}
```

Решение

Здесь переменная R – это число, образованное цифрами заданного числа x, записанными справа налево (убедитесь в этом!). Так как по условию R – двузначное число с суммой цифр 16, то искомое значение x равно 79.

Ответ: 79.

2.9.3. Задание из [8]

Условие

Ниже на четырех языках записан алгоритм. Получив на вход натуральное 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.

Алгоритмический язык

```
алг
нач цел x, a, b, c
  ввод x
  a := 0; b := 10
  нц пока x > 0
    c := mod(x, 10)
    a := a + c
    если c < b
      то
        b := c
```

Паскаль

```
var x, d, R, c: longint;
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;
```

Алгоритмический язык

```
    все
    x := div(x, 10)
  кц
  вывод a, нс, b
кон
```

Паскаль

```
            x := x div 10
    end;
  writeln(a); writeln(b);
end.
```

Python

```
x = int(input())
a = 0
b = 10
while x > 0:
  c = x % 10
  a = a + c
  if c < b:
    b = c
  x = x//10
print(a)
print(b)
```

Си

```
#include<stdio.h>
void main()
{
  int x, a, b, c;
  scanf("%d", &x);
  a = 0; b = 10;
  while (x > 0) {
    c = x % 10;
    a = a + c;
    if (c < b)
      b = c;
    x = x/10;
  }
  printf("%d\n%d", a, b);
}
```

Решение

Анализ показывает, что в алгоритме переменная a – это сумма цифр заданного числа (см. задачу 1.1.3), а b – его минимальная цифра (см. задачу 1.1.8). Так как a = 13, а b = 5, то искомое число x равно 58.

Ответ: 58.

2.9.4. Задание из [7]

Условие

Получив на вход число x, этот алгоритм печатает два числа: L и M. Укажите наименьшее из таких чисел x, при вводе которых алгоритм печатает сначала 5, а потом 7.

Алгоритмический язык

```
алг
нач цел x, L, M
  ввод x
  L := 0
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := 0;
```

Алгоритмический язык

```
  M := 0
  нц пока x > 0
    M := M + 1
    если mod(x, 2) <> 0
      то
        L := L + 1
    все
    x := div(x, 2)
  кц
  вывод L, нс, M
кон
```

Паскаль

```
  M := 0;
  while x > 0 do
    begin
      M := M + 1;
      if x mod 2 <> 0 then
         L := L + 1;
      x := x div 2
    end;
  writeln(L); writeln(M);
end.
```

Python

```
x = int(input()
L = 0
M = 10
while X > 0:
  M = M + 1
  if X % 2 != 0:
    L = L + 1
  x = x//2
print(L)
print(M)
```

Си

```
#include<iostream>
using namespace std;
int main(){
  int x, L, M;
  cin >> x;
  L = 0;
  M = 0;
    while (x > 0) {
      M = M + 1;
        if(x % 2 != 0) {
          L = L + 1;
      }
      x = x/2;
  }
  cout << L << endl << M << endl;
  return 0;
}
```

Решение

Обсудим, что выполняет алгоритм в следующем фрагменте:

```
нц пока x > 0
  M := M + 1
  ...
  x := div(x, 2)
кц
```

Ответ – таким образом подсчитывается количество цифр двоичного числа, соответствующего заданному десятичному x. При этом в приведенных в задании фрагментах одновременно с по-

мощью условного оператора (команды **если**) определяется количество единиц (убедитесь в обоих утверждениях).

Итак, в двоичном представлении заданного числа x общее количество двоичных цифр равно 7, а количество единиц – 5. Минимальное из таких двоичных чисел – 1001111 (вариант 0011111 невозможен). Соответствующее десятичное значение равно 79.

Ответ: 79.

2.9.5. Задание из [5]

Условие

Ниже на четырех языках программирования записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: L и M. Укажите наименьшее число x, при вводе которого алгоритм печатает сначала 6, а потом 7.

Алгоритмический язык

```
алг
нач цел x, L, M
  ввод x
  L := 0
  M := 0
  нц пока x > 0
    M := M + 1
      если mod(x, 2) = 0
        то
          L := L + 1
      все
    x := div(x,2)
  кц
  вывод L, нс, M
кон
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := 0;
  M := 0;
  while x > 0 do
    begin
      M := M + 1;
        if x mod 2 = 0 then
          L := L + 1;
      x := x div 2;
    end;
  writeln(L);
  writeln(M)
end.
```

Python

```
x = int(input())
L = 0
M = 0
while x > 0:
  M = M + 1
  if x % 2 == 0:
    L = L + 1
  x = x//2
```

C++

```
#include <iostream>
using namespace std;
int main()
{
  int x, L, M;
  cin >> x;
  L = 0;
  M = 0;
```

```
print(L)
print(M)
```

```
while (x > 0){
  M = M + 1;
    if(x % 2 == 0){
      L = L + 1;
    }
    x = x/2;
  }
cout << L << endl << M << endl;
return 0;
}
```

Решение

Сравнение условия данного задания с рассмотренным в предыдущем пункте показывает, что принципиальное отличие заключается в условном операторе:

```
если mod(x, 2) = 0
  то
    L := L + 1
все
```

Это говорит о том, что, наряду с общим количеством двоичных цифр M, определяется количество нулей (а не единиц, как в предыдущем задании) L. Минимальное 7-значное двоичное число с шестью нулями: 1 000 000, что соответствует десятичному числу 64.

Ответ: 64.

2.9.6. Задание из [6]

Условие

Ниже на пяти языках программирования записан алгоритм. Получив на вход натуральное десятичное число x, этот алгоритм печатает два числа: L и M. Укажите наибольшее число x, при вводе которого алгоритм печатает сначала 21, а потом 3.

Алгоритмический язык

```
алг
нач цел x, L, M
  ввод x
  L := 1
  M := 0
  нц пока x > 0
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := 1;
  M := 0;
  while x > 0 do
```

Алгоритмический язык

```
    M := M + 1
    если mod(x, 2) <> 0
      то
        L := L * mod(x, 8)
    все
    x := div(x, 8)
  кц
  вывод L, нс, M
кон
```

Паскаль

```
    begin
      M := M + 1;
      if x mod 2 <> 0 then
        L := L * (x mod 8);
      x := x div 2
    end;
  writeln(L); writeln(M);
end.
```

Python

```
x = int(input()
L = 1
M = 10
while X > 0:
  M = M + 1
  if X % 2 != 0:
    L = L * (X % 8)
print(L)
print(M)
```

Си

```
#include<iostream>
using namespace std;
int main(){
  int x, L, M;
  cin >> x;
  L = 0;
  M = 0;
    while (x > 0) {
      M = M + 1;
        if(x % 2 != 0) {
          L = L * (x % 8);
      }
      x = x / 2;
  }
  cout << L << endl << M << endl;
  return 0;
}
```

Решение

На первый взгляд кажется, что задание отличается от задания, рассмотренного в п. 2.9.5, только основанием системы счисления (в п. 2.9.5 – 2, в данной задаче – 8). Однако это не так. Здесь переменная L не «счетчик» количества четных остатков.

Как же работает приведенный алгоритм? В теле оператора цикла происходит уменьшение заданного числа x путем его многократного деления нацело на 8. При этом ведется подсчет количества таких действий с помощью переменной-счетчика M. Кроме того, подсчитывается произведение L всех остатков от деления меняющегося x на 8 (только для нечетных x!).

Так как в результате значение M равно 3, то можем определить диапазон возможных значений числа x. В данном случае минимальное значение x равно $8^2 + 1 = 65$, максимальное – $8^3 - 1 = 511$.

Ответ: 511.

Задания для самостоятельной работы

1. Имеется алгоритм, аналогичный приведенному в пункте 2.9.1. Укажите:

1) наименьшее из таких чисел *x*, при вводе которых алгоритм печатает сначала 2, а потом 0;
2) такое число x, при вводе которого алгоритм печатает сначала 2, а потом 1;
3) такое число x, при вводе которого алгоритм печатает сначала 1, а потом 2;
4) такое число x, при вводе которого алгоритм печатает сначала 2, а потом 64;
5) наименьшее из таких чисел x, при вводе которых алгоритм печатает сначала 15, а потом 3.

2. Получив на вход число x, алгоритм печатает два числа: L и M. Укажите набольшее из таких чисел x, при вводе которых алгоритм печатает сначала 5, а потом 8.

Алгоритмический язык

```
алг
нач цел x, L, M
  ввод x
  L := 0
  M := 0
  нц пока x > 0
    M := M + 1
    если mod(x, 2) = 0
      то
        L := L + 1
    все
    x := div(x, 2)
  кц
  вывод L, нс, M
кон
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := 0;
  M := 0;
  while x > 0 do
    begin
      M := M + 1;
      if x mod 2 = 0 then
        L := L + 1;
      x := x div 2
    end;
  writeln(L); writeln(M);
end.
```

В заключение заметим, что в [10] задание существенно отличалось от рассмотренных.

Условие

Ниже на четырех языках программирования записан алгоритм. 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 26.

Алгоритмический язык

```
алг
нач цел x, L, M
  ввод x
  L := x
  M := 65
  если mod(L, 2) = 0
    то
      M := 52
  все
  нц пока L <> M
    если L > M
      то
        L := L - M
      иначе
        M := M - L
    все
  кц
  вывод M
кон
```

Паскаль

```
var x, L, M: integer;
begin
  readln(x);
  L := x;
  M := 65;
  if L mod 2 = 0 then
    M := 52;
  while L <> M do
    if L > M then
      L := L - M
    else
      M := M - L;
  writeln(M)
end.
```

Python

```
x = int(input())
L = x
M = 65
if L % 2 == 0:
  M = 52
while L != M:
  if L > M:
    L = L - M
  else:
    M = M - L
print(M)
```

Си

```
#include<stdio.h>
void main()
{
  int x, L, M;
  scanf("%d", &x);
  L = x;
  M = 65;
  if (L % 2 == 0)
    M = 52;
  while (L != M){
    if(L > M)
      L = L - M;
    else
      M = M - L;
  }
  printf("%d", M);
}
```

Решение

Анализ показывает, что в операторе цикла многократно происходят вычитания меньшего числа из большего, пока числа L и M не равны между собой. Такие действия выполняются при нахождении наибольшего общего делителя (НОД) двух чисел согласно так называемому алгоритму Евклида (см. приложение 2). Значит, переменная M в результате и есть НОД двух чисел L и M.

Если заданное число x – четное, то M = 52 и при этом L – четное число (L = x); в противном случае M = 65, и при этом L – четное число.

Так как 52 = 2 × 2 × 13, то наименьшим четным числом x, большим 100, при котором НОД(L, M) = 26, является число 130 (130 = 2 × 13 × 5).

Для нечетного x значений, для которых НОД(x, 65) = 26, нет (убедитесь в этом, учитывая, что 65 = 5 × 13).

Ответ: 130.

2.10. Задание 23

В качестве проверяемого элемента содержания для задания 23 в Спецификации [13] указывается: «Умение анализировать результат исполнения алгоритма».

Приведем несколько примеров заданий из [5–8].

Пример 1: «Исполнитель преобразует число на экране. У исполнителя есть две команды, которым присвоены номера.

1. Прибавить 1.
2. Умножить на 2.

Первая команда увеличивает число на экране на 1, вторая умножает его на 2.

Программа для исполнителя – это последовательность команд.

Сколько существует программ, для которых при исходном числе 1 результатом является число 22, и при этом траектория вычислений содержит число 11?

Траектория вычислений программы – это последовательность результатов выполнения всех команд программы. Например, для программы 121 при исходном числе 7 траектория будет состоять из чисел 8, 16, 17».

Пример 2: «Исполнитель Вычислитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера.

1. Прибавить 2.
2. Умножить на 2.
3. Прибавить 3.

Первая из них увеличивает число на экране на 2, вторая умножает его на 2, третья увеличивает его на 3.

Программа для Вычислителя – это последовательность команд.

Сколько существует таких программ, которые преобразуют исходное число 2 в число 22, и при этом траектория вычислений программы содержит число 11?

Траектория вычислений программы – это последовательность результатов выполнения всех команд программы. Например, для программы 123 при исходном числе 7 траектория будет состоять из чисел 9, 18, 21».

Пример 3: «Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера.

1. Прибавить 1.
2. Прибавить 2.
3. Умножить на 2.

Первая из них увеличивает число на экране на 1, вторая увеличивает его на 2, третья умножает его на 2.

Программа для исполнителя – это последовательность команд. Сколько существует таких программ, которые исходное число 3 преобразуют в число 12, и при этом траектория вычислений программы содержит число 10? Траектория вычислений программы – это последовательность результатов выполнения всех команд программы. Например, для программы 132 при исходном числе 7 траектория будет состоять из чисел 8, 16, 18».

Решение

Прежде всего заметим, что полный перебор всех возможных вариантов, начиная с исходного числа, является нерациональным. Оптимальное решение заключается в следующем.

Рассмотрим решение примера 1.

Во-первых, надо разбить задачу на две:

1) задачу *А* – определение количества программ для получения числа 10 при исходном числе 1;
2) задачу *Б* – определение количества программ для получения числа 20 при исходном числе 10.

Тогда искомое в основной задаче значение будет равно произведению двух найденных количеств.

Как решить задачу *А*?

Если исходное число равно 1, а нужно получить 2 (обозначим такую задачу 1 → 2), то возможны два варианта программы:

1) команда 1;
2) команда 2.

Пусть нужно получить 3. Так как 3 – нечетное число, то оно может быть получено только из числа 2 одним способом. Общее количество вариантов для 1 → 2 мы знаем (2). Значит, и для задачи 1 → 3 ответ равен 2.

Задача 1 → 4. Число 4 – четное, оно может быть получено двумя способами – через 2 и через 3. Количество вариантов для каждого случая мы уже знаем (2 и 2 соответственно), т. е. для задачи 1 → 4 ответ равен 2 + 2 = 4.

Искомые значения для больших результатов (и логика их получения) приведена в таблице:

Число	**Может быть получено через...**	**Общее количество вариантов**
2		2
3		2
4	2 и 3	2 + 2 = 4
5	4	4
6	3 и 5	2 + 4 = 6
7	6	6
8	4 и 7	4 + 6 =10
9	8	10
10	5 и 9	4 + 10 =14
11	10	14

Итак, ответом к задаче *А* является 14.

Задача *Б* (определение количества программ для получения числа 22 при исходном числе 11) решается проще. Так как 22 – четное число, то для него искомый результат может быть получен через 11 и 21. Для чисел 20, 19, ..., 12 также возможен только один результат, значит, общее число вариантов программы для задачи Б – 2, а общий ответ ко всему заданию 14 × 2 = 28.

Обратим внимание на то, что для четных чисел 20, 18 ...12 их «половины» рассматривать не следует.

Задания примеров 2 и 3 отличаются тем, что в них в системе команд исполнителя не две, а три команды. Однако принципиально это ничего не меняет, и они решаются аналогично.

Приведем начало решения примера 2.

Задача, названная выше *А*, решается следующим образом:

Число	Может быть получено через...	Общее количество вариантов
2		1 (так как 2 – исходное число)
3	–	0
4	2 и 2	1 + 1 = 2
5	2 и 3	1
6	3, 3 и 4	2
7	4 и 5	2 + 1 = 3
8	4, 5 и 6	2 + 1 + 2 = 5
9	6 и 7	2 + 3 = 5
10	5, 7 и 8	1 + 3 + 4 = 8
11	8 и 9	5 + 5 = 10

Для задачи, названной *Б*, решение такое:

Число	Может быть получено через...	Общее количество вариантов
11		1
12	–	0
13	11	1
14	11 и 12	1
15	12 и 13	1
16	13 и 14	1 + 1 = 2
17	14 и 15	1 + 1 = 2
18	15 и 16	1 + 2 = 3
19	16 и 17	2 + 2 = 4
20	17 и 18	2 + 3 = 5
21	18 и 19	3 + 4 = 7
22	11, 19 и 20	1 + 4 + 5 = 10

Общий ответ: 100.

Задание примера 3 выполните самостоятельно.

В заключение обсудим еще один пример.

Пример 4: «Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера.

1. Прибавить 1.
2. Прибавить 2.
3. Умножить на 3.

Первая из них увеличивает число на экране на 1, вторая увеличивает его на 2, третья умножает на 3.

Программа для исполнителя – это последовательность команд.

Сколько существует таких программ, которые преобразуют исходное число 2 в число 12, и при этом траектория вычислений программы содержит числа 8 и 10? Траектория должна содержать оба указанных числа.

Траектория вычислений программы – это последовательность результатов выполнения всех команд программы. Например, для программы 132 при исходном числе 7 траектория будет состоять из чисел 8, 24, 26».

Решение

Здесь отличие в том, что в условии заданы два обязательных «промежуточных» числа (8 и 10). Поэтому задача разбивается не на две, а на три частные задачи, и общее количество искомых вариантов программы равно произведению трех частных результатов. Найдите это количество самостоятельно.

2.11. Использование файлов

2.11.1. Общие вопросы

Вам, конечно, знакомо понятие «файл». Файлом является документ текстового редактора Microsoft Word, изображение, созданное в графическом редакторе, презентация Microsoft PowerPoint и т. п. Из множества файлов состоит система программирования языка Python или любого другого. То есть так называют некоторую информацию, записанную на носителе (диске, флешке или др.). С другой стороны, файл – это область на носителе с этой информацией, которой присвоено имя.

А зачем файлы нужны при разработке программ? Известно, что удобным средством хранения информации являются мас-

сивы. Но при таком хранении она доступна только в данной, отдельной программе. Когда же нужно использовать одну и ту же информацию (перечень товаров и их характеристики, список фамилий и т. п.) в разных программах, лучше сохранить ее в отдельном файле и при необходимости подключать этот файл к той или иной программе.

С точки зрения программиста, файлы бывают двух типов:

1) текстовые, которые содержат текст, разбитый на строки (в конце каждой строки записан символ, означающий «конец строки»). Сам файл заканчивается меткой конца файла. Доступ к каждой строке возможен лишь последовательно, начиная с первой;
2) двоичные, в которых могут содержаться любые данные и любые коды без ограничений; в двоичных файлах хранятся рисунки, звуки, видеофильмы и т. д.

Мы будем обсуждать только работу с текстовыми файлами.

Работа с текстовым файлом в программе включает три основных этапа.

Сначала надо открыть файл, т. е. сделать его доступным для программы. Если файл не открыт, то программа не может к нему обращаться. Когда файл открыт (доступен), программа выполняет все необходимые операции с ним – читает и обрабатывает информацию из файла, записывает в него новую информацию. После этого нужно закрыть файл, т. е. освободить его, разорвать связь с программой. Именно при закрытии все последние изменения, сделанные программой в файле, записываются на диск или другой носитель информации.

Опишем правила открытия и закрытия файла в программах на языках Python и Паскаль.

Введем важное понятие – «файловая переменная». Так называют переменную, с помощью которой программа осуществляет связь с файлом.

Язык Python

В программе на этом языке открытие файла проводится с помощью инструкции open. Ее формат:

```
<имя файловой переменной> = open(<имя_файла>, <режим открытия>)
```

где *<имя файловой переменной>* – имя файловой переменной; *<режим открытия>* указывает, с какой целью открывается файл; *<имя файла>* – имя используемого файла в кавычках. Этот файл

должен находиться в папке, в которой расположена использующая его программа. Если это не так, то в параметре *<имя файла>* указывается полный или относительный путь к файлу.

На экзамене режим открытия файла – это его чтение. Обозначение режима чтения – "r". Поэтому для чтения инструкция open оформляется следующим образом:

```
f = open("мойфайл.txt", "r")
```

Можно режим "r" не указывать (в операционной системе Windows это значение является значением по умолчанию):

```
f = open("мойфайл1.txt")   # Использован режим открытия по умолчанию
```

После открытия файла и его обработки следует закрыть его. Для этого предназначен метод close. Примеры:

```
f.close()
input.close()
```

где f и input – имена файловых переменных.

Язык Паскаль

В программе на этом языке для работы с файлом файловая переменная (см. выше), как и любая переменная величина, должна быть объявлена (описана) в разделе описаний. Файловая переменная для связи с текстовым файлом описывается так:

```
<имя файловой переменной>: text;
```

Далее, для работы с конкретным файлом нужно созданную файловая переменную связать с этим файлом. Это делается в разделе операторов программы с помощью процедуры Assign. Ее общий вид:

```
Assign(<имя файловой переменной>, (<имя_файла>);
```

Требования к имени используемого файла такие же, как применительно к файлам в программах на языке Python.

После связи файловой переменной с файлом в программе нужно указать режим работы с ним. Для чтения файл открывается с помощью процедуры Reset. Ее формат:

```
Reset(<имя файловой переменной>);
```

После использования файла он должен быть закрыт с помощью процедуры Close:

```
Close(<имя файловой переменной>)
```

2.11.2. Чтение информации из файла

Чтение первой (или единственной) строки файла

Пример задачи: «В текстовом файле *Пример1.txt* записана строка символов. Найти длину самой длинной последовательности подряд идущих символов "Д"».

Алгоритм ее решения состоит из трех основных этапов:

```
1. Прочитать строку из файла и запомнить ее в переменной, например
с именем stroka.
2. Применительно к строке и к символу "Д" решить задачу,
рассмотренную в разделе 2.12.
3. Вывести ответ.
```

Опишем методику выполнения этапа 1 в программе на языке Python.

Как указывается в п. 2.11.1, для чтения информации из файла его нужно открыть:

```
f = open("Пример1.txt", "r")
```

или (используя режим открытия по умолчанию):

```
f = open("Пример1.txt")
```

где `f` – файловая переменная.

После открытия файла его первую строку можно прочитать с помощью метода `readline()`:

```
stroka = f.readline()
```

Конечно, после решения задачи следует файл закрыть (см. выше).

Удобно совместить операции открытия файла, чтения информации из его первой строки и закрытия файла. Это можно сделать следующим образом:

```
with open(<имя файла>, "r") as <имя файловой переменной>:
  stroka = <имя файловой переменной>.readline()
```

В результате выполнения указанной конструкция в переменную `stroka` будет записана первая (или единственная) строка соответствующего файла.

Если в программе будет происходить чтение и других (или всех) строк файла, то последний вариант применять нельзя.

В программе на языке Паскаль этап 1 выполняется так:

```
Var f: text; stroka: string;              256!
BEGIN
  Assign(f, "Пример1.txt");
  Reset(f);
  { После открытия файла его первую строку можно прочитать с помощью
  процедуры Readln }
  Readln(f, stroka);
  …
```

Приведем некоторые из возможных задач по обработке первой (или единственной) строки файла:

1) определить количество символов, равных заданному или обладающих некоторыми свойствами;
2) определить позицию (номер) первого вхождения некоторого символа;
3) определить, имеется ли в строке некоторый символ;
4) определить количество цифр в строке;
5) определить, имеются ли в строке цифры (знаки препинания или т. п.);
6) определить количество вхождений некоторой подстроки;
7) определить сумму цифр в строке;
8) определить количество слов в строке;
9) определить среднюю длину слова;
10) вывести на экран слово с заданным номером;
11) вывести на экран слово максимальной/минимальной длины (если таких слов несколько – любое из них);
12) вывести на экран длину самого большого или/и самого маленького слова;
13) определить длину самой длинной подстроки (цепочки) из заданного символа;
14) определить длину самой длинной подстроки (цепочки) из одинаковых символов и символ в этой подстроке;
15) получить строку, которая получается из исходной после удаления из нее всех вхождений заданной подстроки;
16) получить строку, которая получается из исходной после замены всех вхождений заданной подстроки на некоторую заданную подстроку;
17) известно, что в строке имеется несколько подстрок из трех цифр. Определить:
    а) их количество;
    б) сумму всех чисел, образованных подстроками из цифр;

в) максимальное/минимальное из соответствующих чисел;
г) порядковый номер подстроки, число в которой является максимальным (если таких подстрок несколько – номер последней из них);
д) подстроку, соответствующее число в которой содержит минимальную сумму цифр (если таких подстрок несколько – номер первой из них);

18) известно, что в строке имеется несколько подстрок из цифр. Определить:
а) их количество;
б) сумму всех чисел, образованных подстроками из цифр;
в) минимальное из соответствующих чисел;
г) количество цифр в самой длинной из них;
д) порядковый номер самой короткой из них (если таких подстрок несколько – номер первой из них);
е) порядковый номер подстроки, число в которой содержит максимальную сумму цифр (если таких подстрок несколько – номер последней из них).

Обратим внимание на то, что:
1) задание на обработку строк представлено в проекте демонстрационного варианта ЕГЭ 2021 года (см. раздел 3.12);
2) большинство перечисленных выше и далее задач представлено в Кодификаторе элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2021 году единого государственного экзамена по информатике и ИКТ [4];
3) методика решения задач описана в разделах 1.5 и 2.12.

Чтение двух первых строк файла

Прочитать две первые строки файла и сохранить их в переменных, соответственно, можно с следующим образом:
– в программе на языке Python:

```
# Открываем файл на чтение
f = open(…,"r")
# Читаем первую строку файла и запоминаем ее в переменной stroka1
stroka1 = f.readline()
# То же, вторую строку
stroka2 = f.readline()
```

– в программе на языке Паскаль:

```
Var f: text; stroka1, stroka2 : string;
BEGIN
```

```
Assign(f, "Пример1.txt");
Reset(f);
Readln(f, stroka1);
Readln(f, stroka2);
...
```

Приведем некоторые из возможных задач по обработке двух прочитанных строк файла:

1) определить общее количество (здесь и далее подразумеваются обе строки):
   а) символов;
   б) конкретных символов;
   в) слов;
   г) слов с определенными свойствами (начинающимися на некоторую букву, заданной длины и т. п.);
2) определить среднюю длину слова;
3) определить количество символов в строке максимальной/минимальной длины;
4) определить номер строки максимальной/минимальной длины;
5) определить, имеется ли в прочитанных строках некоторый символ, некоторая подстрока или некоторое слово;
6) определить общее количество цифр;
7) определить, имеются ли в прочитанных строках цифры (знаки препинания или т. п.);
8) определить общее количество вхождений некоторой подстроки;
9) определить сумму цифр;
10) вывести на экран слово максимальной/минимальной длины (если таких слов несколько – любое из них);
11) вывести на экран длину самого большого или/и самого маленького слова;
12) определить номер строки со словом максимальной/минимальной длины (если таких строк две – номер любой строки с соответствующим словом);
13) определить длину самой длинной подстроки (цепочки) из заданного символа;
14) определить номер строки с самой длинной подстрокой (цепочкой) из заданного символа (если таких строк две – номер любой из них);
15) определить длину самой длинной подстроки (цепочки) из одинаковых символов и символ в этой подстроке;

16) определить номер строки с самой длинной подстрокой (цепочкой) из одинаковых символов (если таких строк две – номер любой из них);
17) определить количество символов в слове, больше которого только в самом длинном слове;
18) известно, что в строках имеется несколько подстрок из трех цифр. Определить:
    а) их количество;
    б) сумму всех чисел, образованных подстроками из цифр;
    в) максимальное/минимальное из соответствующих чисел;
    г) номер строки, число в которой является максимальным (если таких строк две – номер любой из них);
    д) номер строки, соответствующее число в которой содержит минимальную сумму цифр (если таких строк две – номер любой из них);
19) известно, что в строках имеется несколько подстрок из цифр. Определить:

а) их количество;
б) сумму всех чисел, образованных подстроками из цифр;
в) минимальное из соответствующих чисел;
г) количество цифр в самой длинной из них;
д) номер строки, число в которой является максимальным (если таких строк две – номер любой из них);
е) номер строки, соответствующее число в которой содержит максимальную сумму цифр (если таких строк две – номер любой из них).

Чтение всех строк файла

Сначала примем, что количество строк в файле известно и равно `n` (значение `n` вводится с клавиатуры).

В программе на языке Python для чтения всех строк можно использовать оператор цикла с параметром:

```
n = input()                    # Ввод значения n
f = open(…,"r")
for k in range(n):             # Для всех строк читаемого файла
  stroka = f.readline()        # Значение очередной строки
  …                            # Обработка очередной строки
```

в программе на языке Паскаль:

```
Var f: text; stroka : string; k, n: integer;
Begin
```

```
Readln(n);                    { Ввод значения n }
Assign(f, …);
Reset(f);
for k := 1 to n do
  begin
  Readln(f, stroka);          { Значение очередной строки }
  …                           { Обработка очередной строки }
```

Приведем некоторые из возможных задач обработки всех прочитанных строк файла:

1) определить общее количество (здесь и далее подразумеваются все строки файла):
   - символов;
   - конкретных символов;
   - слов;
   - слов с определенными свойствами (начинающихся на некоторую букву, заданной длины и т. п.);
2) определить количество символов в строке максимальной/минимальной длины;
3) определить номер строки максимальной/минимальной длины;
4) определить, имеется ли в файле некоторый символ, некоторая подстрока или некоторое слово;
5) вывести на экран слово максимальной/минимальной длины (если таких слов несколько – любое из них);
6) вывести на экран длину самого большого или/и самого маленького слова;
7) определить номер строки с словом максимальной/минимальной длины (если таких строк несколько – номер любой строки с соответствующим словом);
8) определить номер строки, в которой встречается первое слово максимальной/минимальной длины;
9) вывести на экран строку/строки с заданным номером/номерами;
10) определить номер строки, в которой впервые встречается заданный символ или заданная подстрока (принять, что такой символ/подстрока единственные);
11) определить, имеется ли в файле заданный символ или заданная подстрока;
12) определить длину самой длинной подстроки (цепочки) из заданного символа;

13) определить номер строки с самой длинной подстрокой (цепочкой) из заданного символа (если таких строк несколько – номер любой из них);
14) определить длину самой длинной подстроки (цепочки) из одинаковых символов и символ в этой подстроке;
15) определить номер строки с самой длинной подстрокой (цепочкой) из одинаковых символов (если таких строк несколько – номер любой из них);
16) определить количество символов в слове, больше которого только в самом длинном слове;
17) известно, что в каждой строке файла записано целое число. Определить:
    а) сумму всех чисел;
    б) среднее арифметическое всех чисел;
    в) сумму чисел, записанных на 2-й, 4-й, 6-й... строках;
    г) количество четных чисел;
    д) среднее арифметическое отрицательных чисел;
    е) максимальное число в файле;
    ж) номер строки, в которой записано первое минимальное число файла;
    з) имеется ли в файле заданное число;
18) известно, что в конце некоторых строк файла записана точка. Определить:
    а) количество таких строк;
    б) максимальное число слов в отдельной строке;
    в) общее количество всех слов в таких строках.
19. Известно, что в строках файла записаны через пробел название государства и его столица. Определить:
    а) столицу заданного государства;
    б) столицей какого государства является заданный город;
20) известно, что в строках имеется несколько подстрок из трех цифр. Определить:
    а) их количество;
    б) сумму всех чисел, образованных подстроками из цифр;
    в) максимальное/минимальное из соответствующих чисел;
    г) номер строки файла, число в которой является максимальным (если таких строк более одной – номер любой из них);
    д) номер строки файла, соответствующее число в которой содержит минимальную сумму цифр (если таких строк более одной – номер любой из них);

21. Известно, что в строках имеется несколько подстрок из цифр. Определить:
    а) их количество;
    б) сумму всех чисел, образованных подстроками из цифр;
    в) минимальное из соответствующих чисел;
    г) количество цифр в самой длинной из них;
    д) номер строки файла, число в которой является максимальным (если таких строк несколько – номер первой из них);
    е) номер строки файла, соответствующее число в которой содержит максимальную сумму цифр (если таких строк несколько – номер последней из них).

Перечисленные задачи можно решить двумя способами:

1) с обработкой каждой строки файла сразу после ее прочтения;
2) с обработкой строк файла после записи их в массив.

В языке программирования Python имеются оригинальные возможности записи всех строк текстового файла в список. В частности, это можно сделать с помощью методов `readlines()` и `strip()`. Например, для получения списка `mas`, состоящего из всех строк файла, связанного в программе с файловой переменной `f`, необходимо записать:

```
mas = f.readlines()
```

или

```
mas = [line.strip() for line in f]
```

Еще один оригинальный способ – одновременное использование методов `read()`и `split()`:

```
mas = f.read().split()
```

Первый метод читает все строки файла и представляет их как одну строку, а второй – разделяет ее на исходные части и записывает их по отдельности в список.

Можно также записать в список все слова одной строки текстового файла. Для этого надо:

1) прочитать эту строку:

```
stroka = f.readline()
```

2) получить список `slova` ее слов:

```
slova = stroka.split()
```

Отдельно рассмотрим случай, когда в первой строке текстового файла записано количество остальных строк в нем (именно так заданы исходные данные в заданиях 26 и 27 [1]).

Ясно, что в этом случае сначала надо прочитать первую строку файла, после ее обработки определить количество остальных строк и затем прочитать и обработать эти строки.

При этом обязательно следует учесть, что прочитанное значение имеет строковый тип, а нужное количество – это число. Как указывалось в дополнении к разделу 1.5, для преобразования строковой величины в число предусмотрены специальные функции.

Соответствующие фрагменты программ:

Язык Паскаль

```
Var f: text; stroka, ks : string; k, kol_strok: integer;
BEGIN
  Assign(f, …);
  Reset(f);
      { Читаем первую строку файла со значением}
  Readln(f, ks);      { ks - количество остальных строк
                        в виде величины строкового типа}
  kol_strok := StrToInt(ks)    { Количество остальных строк }
  for k := 1 to kol_strok do
    begin
        Readln(f, stroka);   { Значение очередной строки }
        …                    { Обработка очередной строки }
```

Язык Python

```
f = open(…,"r")
# Читаем первую строку файла
# и сразу преобразовываем ее в целое число
kol_strok = int(f.readline())  # Количество остальных строк
for k in range(kol_strok):
  stroka = f.readline()        # Значение очередной строки
  …                            # Обработка очередной строки
```

В заключение заметим следующее. Если вы будете выполнять задания для самостоятельной работы или другие тренировочные задания на обработку файлов, то вам понадобятся файлы с исходной информацией. Для этого следует подготовить такие файлы. С правилами записи информации в текстовый файл ознакомьтесь самостоятельно (см. также [17]).

2.12. Задание 24

В качестве проверяемого элемента содержания для задания 24 в Спецификации [13] указывается: «Умение создавать собственные программы (10–20 строк) для обработки символьной информации».

В данном разделе рассмотрим ряд задач, связанных с подстроками[1], обладающими некоторыми свойствами (удовлетворяющими некоторому условию).

2.12.1. Нахождение максимальной длины подстроки

Хотя в заданиях экзамена говорится о том, что обрабатываемые символы читаются из файла, обсудим алгоритмы решения задач, в которых исходная строка вводится с клавиатуры.

Пример задачи. «Дана строка символов, среди которых есть несколько подстрок из букв "я". Определить количество букв в самой длинной из них».

Решение

Для решения задачи надо определить максимальное из чисел – длин каждой подстроки. Длина некоторой подстроки рассчитывалась в п. 1.5.3, а максимум среди ряда чисел – в пп. 1.1.7 и 1.4.1.

Пусть заданная строка имеет вид:

```
ппяяярооояяяяялллояяяяяяпряяя
```

Как решает аналогичную задачу человек? Эти действия можно описать такой схемой:

```
1. Он просматривает каждую отдельную букву (слева направо):
   если это буква «я»,
     то (цепочка букв «я» началась или продолжается)
        увеличивает длину текущей цепочки на 1,
     иначе (текущая цепочка букв «я» закончилась)
              или  не начиналась,
        если закончилась,
          то
             сравнивает длину закончившейся цепочки
                со «старым» значением максимальной длины;
             если длина закончившейся цепочки больше «старого»
                  значения максимальной длины,
```

[1] Напомним (см. п.1.5.3), что подстрокой называют часть величины строкового типа (используется также понятие «цепочка символов»).

то
в качестве значения максимальной длины
запоминает длину закончившейся цепочки,
все
Готовится в будущем определять длину новой цепочки
(пока ее длина равна 0),
все
все

2. После просмотра, если в конце строки также имеется цепочка из букв «я», он сравнивает ее длину с максимальной длиной и при необходимости запоминает эту длину в качестве искомой максимальной.

Разработаем соответствующую программу. Используем в ней следующие основные величины:

- строка – заданная строка;
- ном – номер символа в ней;
- тек_длина – длина текущей цепочки букв «я»;
- макс_длина – искомая максимальная длина.

Прежде чем представлять программу, обсудим вопрос: «Как определить, что очередная цепочка букв "я" закончилась?» Ответ – если встретилась буква, отличная от буквы «я», а предыдущая буква – «я». Правда, этот признак не распространяется на первую букву строки.

Вся программа:

```
алг
нач лит строка, цел тек_длина, макс_длина, ном
  ввод строка
  тек_длина := 0            | Начальные
  макс_длина := 0           | значения
  нц для ном от 1 до длин(строка)
    если строка[ном] = "я"
      то
        | Цепочка началась или продолжается
        | Увеличиваем ее длину
        тек_длина := тек_длина + 1
      иначе | Цепочка закончилась или не начиналась
        если ном > 1  и строка[ном - 1] = "я"
          то | Цепочка закончилась
             | Сравниваем ее длину со значеним макс_длина
             если тек_длина > макс_длина
```

```
            то
              | Запоминаем новое значение
              макс_длина := тек_длина
          все
          | Готовимся к определению длины очередной цепочки
          тек_длина := 0
      все
    все
  кц
  | Проверяем, не закончилась ли строка цепочкой букв "я"
  если строка[длин(строка)] = "я"
    то
      | Проверяем длину последней цепочки букв "я"
      если тек_длина > макс_длина
        то
          макс_длина := тек_длина
      все
  все
  | Выводим ответ
  вывод макс_длина
кон
```

Обратим внимание на то, что после окончания текущей цепочки величине тек_длина присваивается нулевое значение независимо от результата сравнения величин тек_длина и макс_длина.

Приведенная программа является несколько громоздкой. Ее можно упростить, учитывая следующие рассуждения.

В конце программы можно не проверять последний символ, а сравнить длину последней возможной цепочки без условия:

```
  ...
  кц
  | Проверяем последнюю возможную цепочку букв "я"
  если тек_длина > макс_длина
    то
      макс_длина := тек_длина
  все
  | Выводим ответ
  ...
```

При этом, если в конце строки нет нужных символов, то будет проводиться сравнение значения величины макс_длина и нуля.

Аналогично в теле оператора цикла можно не проверять, закончилась ли очередная цепочка:

```
нц для ном от 1 до длин(строка)
  если строка[ном] = "я"
    то
      ...
    иначе | Цепочка закончилась или не начиналась
      | Сравниваем значения тек_длина и макс_длина
      если тек_длина > макс_длина
        то
          макс_длина := тек_длина
      все
      тек_длина := 0
  все
кц
```

Здесь также будет иногда проводиться сравнение значения величины макс_длина и нуля.

Язык Паскаль

```
Var stroka: string;
    tek_dlina, max_dlina, nom:
integer;
BEGIN
 readln(stroka);
 tek_dlina := 0;
 max_dlina := 0;
 for nom := 1 to length(stroka) do
   if stroka[nom] = 'я' then
     tek_dlina := tek_dlina + 1
   else
     begin
       if tek_dlina > max_dlina then
          max_dlina := tek_dlina;
       tek_dlina := 0
     end;
 if tek_dlina > max_dlina then
     max_dlina := tek_dlina;
 writeln(max_dlina)
END.
```

Язык Python

```
stroka = input()
tek_dlina = 0
max_dlina = 0
for nom in
range(len(stroka)):
  if stroka[nom] == "я":
    tek_dlina = tek_dlina + 1
  else:
    if tek_dlina > max_
dlina:
      max_dlina = tek_dlina
    tek_dlina = 0
if tek_dlina > max_dlina:
  max_dlina = tek_dlina
print(max_dlina)
```

В программе на языке Python можно после окончания каждой цепочки записать ее длину в список[1], а затем определить максимальное значение в списке с помощью функции max():

[1] Массив можно использовать и в программах на школьном алгоритмическом языке и языке Паскаль, но его придется описать с «запасом».

```
spisok = []
stroka = input()
tek_dlina = 0
for nom in range(len(stroka)):
  if stroka[i] == "я":
    tek_dlina = tek_dlina + 1
  else: # Цепочка закончилась или не начиналась
    # Добавляем ее длину или 0 в список
    spisok.append(tek_dlina)
    tek_dlina = 0
# Добавляем в список последнее значение tek_dlina
spisok.append(tek_dlina)
max_dlina = max(spisok)
print(max_dlina)
```

Заметим, что в [16] приведен такой компактный вариант алгоритма решения обсуждаемой задачи:

```
1. Начальные значения длины текущей цепочки и искомой максимальной
длины принимаются равными нулю
2. Цикл для каждого символа строки
   если он совпадает с заданным символом
      то
           1. Длина текущей цепочки соответствующих символов
              увеличивается на 1
           2. Полученная длина текущей цепочки
              сравнивается с максимальной длиной
           если Полученная длина больше максимальной
             то
               Длина текущей цепочки принимается в качестве максимальной
          все
      иначе
           Начальное значение текущей длины очередной цепочки
           принимается равной нулю
   все
конец цикла
```

Соответствующая программа (применительно к задаче, связанной с буквой «я») оформляется так:

```
алг
нач лит строка, цел тек_длина, макс_длина, ном
  ввод строка
  тек_длина := 0          | Начальные значения
  макс_длина := 0         | переменных
  нц для ном от 1 до длин(строка)
```

```
    если строка[ном] = "я"
      то | Цепочка букв "я" началась или продолжается
         | Увеличиваем ее длину
        тек_длина := тек_длина + 1
        | Сравниваем
        если тек_длина > макс_длина
          то
            | Запоминаем новое значение
            макс_длина := тек_длина
        все
      иначе | Цепочка букв "я" закончилась или не начиналась
            | Готовимся к определению длины очередной цепочки букв "я"
        тек_длина := 0
    все
  кц
  вывод макс_длина
кон
```

Язык Паскаль

```
Var stroka: string;
    tek_dlina, max_dlina, nom:
integer;
BEGIN
  readln(stroka);
  tek_dlina := 0;
  max_dlina := 0;
  for nom := 1 to length(stroka) do
    if stroka[i] = 'я' then
      begin
        tek_dlina := tek_dlina + 1;
        if tek_dlina > max_dlina then
           max_dlina := tek_dlina;
      end
    else
      tek_dlina := 0;
  writeln(max_dlina)
END.
```

Язык Python

```
stroka = input()
tek_dlina = 0
max_dlina = 0
for nom in
range(len(stroka)):
  if stroka[nom] == "я":
    tek_dlina = tek_dlina + 1
    if tek_dlina > max_dlina:
      max_dlina = tek_dlina
  else:
    tek_dlina = 0
print(max_dlina)
```

Особенность приведенного алгоритма в том, что проверка на максимум и при необходимости его изменение проводятся после «встречи» каждого требуемого символа, а не только после окончания цепочки нужных символов.

Предлагаем читателям сравнить его с приведенным выше «естественным» алгоритмом решения рассматриваемой задачи, аналогичным действию человека, который решает ее.

Заметим также, что в [16] в описании алгоритма и в программах ошибочно указано, что в цикле присваивание текущей длине нулевого значения происходит, когда цепочка символов кончилась (на самом деле это имеет место и тогда, когда очередная цепочка еще не начиналась).

И еще важное замечание. Ниже (см. п 2.12.6) будет рассмотрена задача нахождения номера подстроки максимальной длины. Решать эту задачу на основе приведенного «нашего» краткого варианта программы и варианта, предложенного в [16], нельзя.

Задача нахождения минимальной длины подстроки решается аналогично (естественно, с учетом того, что ищется не максимум, а минимум). В качестве начального значения искомой длины можно принять общее количество символов в строке.

Задание для самостоятельной работы

Дана строка символов, в которой есть несколько цепочек цифр. Определить количество цифр в самой короткой цепочке.

Из приведенных *выше* примеров задач видно, что в них речь идет о подстроках одних и тех же символов или символов, обладающих одними и теми же свойствами (например, символов-цифр). Возможны и другие варианты задач на нахождение максимальной/минимальной длины подстроки.

2.12.2. Нахождение максимальной длины подстроки. Второй вариант задачи

Пример задачи: «Дана строка символов, среди которых есть цепочки из букв *а*, цепочки из букв *б* и цепочки из букв *в* (в каждом случае – одна или несколько). Определить количество букв в самой длинной из них (известно, что такая цепочка единственная) и букву в соответствующей цепочке. Учесть, что в строке могут быть и другие символы».

Видно, что отличием задачи является то, что в строке учитываются подстроки не одного вида, а более.

Решение

Так как на экзамене потребуется только ввести ответ (в данном случае – два значения), то программу решения данной задачи

можно разработать на основе программы решения соответствующей задачи в предыдущем разделе, внеся в нее незначительное изменение. В частности, вместо конкретной буквы следует использовать переменную величину (пусть ее имя – *буква*).

Полный вариант программы:

```
алг
нач лит строка, цел тек_длина, макс_длина, ном, сим буква
  ввод строка
  ввод буква
  тек_длина := 0
  макс_длина := 0
  нц для ном от 1 до длин(строка)
    если строка[ном] = буква
      …
```

Запустив эту программу трижды и введя каждый раз ту или иную букву, можно получить три значения максимальной длины, большее из которого указать на экзамене в качестве ответа, а также установить второй ответ – букву в цепочке максимальной длины (которую также следует ввести в качестве ответа).

Можно также использовать упрощенный вариант программы и вариант, названный в п. 2.12.1 компактным.

Программа, в которой рассматривается три возможных значения каждого символа строки, является более громоздкой.

2.12.3. Нахождение максимальной длины подстроки. Третий вариант задачи

Пример задачи: «Дана строка символов. Определить максимальное количество идущих подряд одинаковых символов».

Решение

Здесь признак окончания подстроки одинаковых символов такой – очередной символ не совпадает с предыдущим. Еще одно отличие программы в том, что рассмотрение символов строки начинается со второго символа, а начальное значение переменной тек_длина принимается равным 1 (см. задачу 1 в п. 1.5.3).

Программа

```
алг
нач лит строка, цел тек_длина, макс_длина, ном
  ввод строка
  тек_длина := 1
```

```
  макс_длина := 0
  нц для ном от 2 до длин(строка)
    если строка[ном] = строка[ном - 1]
      то
        тек_длина := тек_длина + 1
      иначе
        если тек_длина > макс_длина
          то
            макс_длина := тек_длина
        все
        тек_длина := 1  | Началась новая подстрока
                        | возможных одинаковых соседних символов
    все
  кц
  если тек_длина > макс_длина
    то
      макс_длина := тек_длина
  все
  вывод нс, макс_длина
кон
```

Язык Паскаль

```
Var stroka: string;
    tek_dlina, max_dlina, nom:
integer;
BEGIN
 readln(stroka);
 tek_dlina := 1;
 max_dlina := 0;
 for nom := 2 to length(stroka) do
   if stroka[nom] = stroka[nom - 1]
 then
     tek_dlina := tek_dlina + 1
   else
     begin
       if tek_dlina > max_dlina then
          max_dlina := tek_dlina;
       tek_dlina := 1
     end;
 if tek_dlina > max_dlina then
     max_dlina := tek_dlina;
 writeln(max_dlina)
END.
```

Язык Python

```
stroka = input()
tek_dlina = 1
max_dlina = 0
for nom in range(1,
len(stroka)):
  if stroka[nom] ==
        stroka[nom - 1]:
    tek_dlina = tek_dlina + 1
  else:
    if tek_dlina > max_dlina:
      max_dlina = tek_dlina
    tek_dlina = 1
if tek_dlina > max_dlina:
  max_dlina = tek_dlina
print(max_dlina)
```

2.12.4. Нахождение максимальной длины подстроки. Четвертый вариант задач

Пример задачи: «Дана строка символов. Определить количество символов в самой длинной подстроке из одинаковых символов и символ в соответствующей подстроке. Известно, что такая подстрока – единственная».

Видно, что особенность задач в том, что значения символов в исследуемых подстроках неизвестны.

Здесь идея решения такая:

1. Рассматривается каждый символ строки.

Если очередной символ совпадает с предыдущим,
то
(подстрока таких символов продолжается)
ее длина увеличивается на 1,
иначе
(очередная подстрока закончилась, и началась новая.
1. Длина закончившейся подстроки сравнивается с максимальной длиной.
Если она больше максимальной,
то
1) в качестве максимальной длины принимается текущая длина;
2) в качестве второго искомого значения принимается буква в закончившейся подстроке,
все.
2. Новая подстрока пока состоит из одного символа (текущего), а ее длина пока равна 1 ,
все.
2. Длина последней подстроки сравнивается с максимальной длиной.
Если она больше максимальной,
то
1) в качестве максимальной длины принимается длина последней подстроки;
2) в качестве второго искомого значения принимается буква в последней подстроке,
все.

Понятно, что в цикле рассматриваются символы строки, начиная со второго, а первый элемент рассматривается отдельно.

В программе решения задачи используем две новые (по сравнению с программой решения задачи примера 3, рассмотренной в предыдущем пункте) переменные величины:

- *симв* – текущий символ строки;
- *симв_макс* – символ в подстроке максимальной длины.

Программа:

```
алг
нач лит строка, цел тек_длина, макс_длина, ном, сим симв, симв_макс
  ввод строка
  макс_длина := 0          | Начальное значение
  | Запоминаем первый символ строки
  симв := строка[1]
  тек_длина := 1   | Длина начальной подстроки пока равна 1
  | Рассматриваем остальные символы
  нц для ном от 2 до длин(строка)
    если строка[ном] = симв | Или если строка[ном] = строка[ном - 1]
      то
        | Подстрока продолжается
        | Увеличиваем ее длину
        тек_длина := тек_длина + 1
      иначе | 1. Подстрока закончилась
        | Сравниваем ее длину со значением макс_длина
        если тек_длина > макс_длина
          то
            | Запоминаем новые значения
            макс_длина := тек_длина
            симв_макс := симв
        все
        | 2. Длина новой подстроки
        тек_длина := 1
        | Символ новой подстроки
        симв := строка[ном]
    все
  кц
  | Проверяем длину последней подстроки
  если тек_длина > макс_длина
    то
      | Запоминаем последние значения
      макс_длина := тек_длина
      симв_макс := симв
  все
  | Выводим ответы
  вывод макс_длина
  вывод нс, симв_макс
кон
```

Язык Паскаль

```
Var stroka: string;
    tek_dlina, max_dlina, nom:
integer;
    simv, simv_max: char;
BEGIN
 readln(stroka);
 max_dlina := 0;
 simv := stroka;
 tek_dlina := 1;
 for nom := 2 to length(stroka) do
   if stroka[nom] = simv then
     tek_dlina := tek_dlina + 1
   else
     begin
       if tek_dlina > max_dlina then
         begin
          max_dlina := tek_dlina;
          simv_max := simv
         end;
       tek_dlina := 1;
       simv := stroka[nom]
     end;
 if tek_dlina > max_dlina then
   begin
     max_dlina := tek_dlina;
     simv_max := simv
   end;
 writeln(max_dlina);
 writeln(simv_max)
END.
```

Язык Python

```
stroka = input()
max_dlina = 0
simv = stroka[0]
tek_dlina = 1
for nom in range(1,
len(stroka)):
  if stroka[nom] == simv:
    tek_dlina = tek_dlina + 1
  else:
    if tek_dlina > max_
dlina:
      max_dlina = tek_dlina
      simv_max = simv
    tek_dlina = 1
    simv = stroka[nom]
if tek_dlina > max_dlina:
  max_dlina = tek_dlina
  simv_max = simv
print(max_dlina)
print(simv_max)
```

В [16] представлен компактный вариант алгоритма (аналогичный приведенному в п. 2.12.1):

```
1. Запоминается первый символ строки, а длина текущей цепочки
   принимается равной 1
   Начальное значения искомой максимальной длины принимается равным 1
2. Цикл для каждого из остальных символов строки
   если он совпадает с предыдущим символом
     то
       1. Длина текущей цепочки символов увеличивается на 1
       2. Полученная длина текущей цепочки сравнивается
       с максимальной длиной
```

```
      если Полученная длина больше максимальной
        то
           Длина текущей цепочки принимается в качестве максимальной
           Символ текущей цепочки запоминается
           в качестве второго искомого значения
      все
    иначе
      Запоминается текущий символ
      Начальное значение текущей длины очередной цепочки
      принимается равной 1
  все
  конец цикла
```

Описанный подход к решению задачи нахождению порядкового номера подстроки максимальной/минимальной длины не подходит (см. задачу 1.12.6).

2.12.5. Нахождение максимальной длины цепочки подстрок

Пример задачи: «Дана строка символов. Определить максимальное количество подряд идущих подстрок “САН”».

Решение

Здесь, как и при решении задачи 2 в п. 1.5.3, надо рассматривать не по одному символу, а по три (использовать срез – см. п. 1.5.1). При этом оператор цикла с параметром применять нельзя.

Логика решения изложена в комментариях к программе.

```
алг
нач лит строка, цел тек_длина, макс_длина, ном
  ввод строка
  тек_длина := 0
  макс_длина := 0
  ном := 1
  нц пока ном <= длин(строка) - 2
    если строка[ном : ном + 2] = "САН"
      то  | Цепочка подстрок "САН" началась или продолжается
        тек_длина := тек_длина + 1
        ном := ном + 3 | Пропускаем подстроку "САН"
      иначе  | Цепочка подстрок "САН" закончилась
             | Сравниваем ее длину со значением макс_длина
        если тек_длина > макс_длина
          то
            макс_длина := тек_длина
        все
```

```
        | Переходим к следующему символу строки
        ном := ном + 1
        тек_длина := 0
    все
  кц
  | Проверяем длину последней возможной цепочки подстрок "САН"
  если тек_длина > макс_длина
    то
      макс_длина := тек_длина
  все
  вывод макс_длина
кон
```

Вариант программы, названный в п. 2.12.1 компактным:

```
алг
нач лит строка, цел тек_длина, макс_длина, ном
  ввод строка
  тек_длина := 0
  макс_длина := 0
  ном := 1
  нц пока ном <= длин(строка) - 2
    если строка[ном : ном + 2] = "САН"
      то  | Цепочка подстрок "САН" началась или продолжается
        тек_длина := тек_длина + 1
        | Сравниваем ее длину со значением макс_длина
        если тек_длина > макс_длина
          то
            макс_длина := тек_длина
        все
        ном := ном + 3 | Пропускаем подстроку "САН"
      иначе  | Цепочка подстрок "САН" закончилась
        | Переходим к следующему символу строки
        ном := ном + 1
        тек_длина := 0
    все
  кц
  вывод макс_длина
кон
```

Язык Паскаль

```
Var stroka: string;
    tek_dlina, max_dlina, nom: integer;
BEGIN
  readln(stroka);
```

```
  tek_dlina := 0;
  max_dlina := 0;
  nom := 1;
  while nom <= length(stroka) - 2  do
    if stroka[ном : ном + 2] = 'САН' then
      begin
        tek_dlina := tek_dlina + 1;
        if tek_dlina > max_dlina then
           max_dlina := tek_dlina;
         nom := nom + 3
      end
    else
      begin
        nom := nom + 1;
        tek_dlina := 0
      end;
  writeln(max_dlina)
END.
```

Язык Python

```
stroka = input()
tek_dlina = 0
max_dlina = 0
nom = 0
while nom <= len(stroka) - 3:
  if stroka[nom : nom + 2] == "САН":
    tek_dlina = tek_dlina + 1
    if tek_dlina > max_dlina:
      max_dlina = tek_dlina
    nom = nom + 3
  else:
    nom = nom + 1
    tek_dlina = 0
print(max_dlina)
```

2.12.6. Нахождение порядкового номера подстроки максимальной длины

Обсудим программу применительно к задаче, аналогичной рассмотренной в п. 2.12.1: «Дана строка символов, среди которых есть несколько цепочек букв “я”. Определить порядковый номер цепочки максимальной длины среди таких цепочек. Если в строке

имеется несколько цепочек максимальной длины – определить номер первой из них».

Сразу же заметим, что здесь упрощенный вариант программы (см. задачу 2.12.1) применять нельзя (обоснование приведено в комментариях к приведенной ниже программе). Это же уточнение касается решения данной задачи на основе компактного варианта решения задачи 1.12.1, предложенного в [16].

В приведенной ниже программе, по сравнению с программой решения задачи 1.12.1, использованы две дополнительные переменные величины:

1) *ном_подстр* – порядковый номер очередной подстроки;
2) *ном_макс* – искомый номер подстроки максимальной длины.

```
алг
нач лит строка, цел тек_длина, макс_длина, ном_подстр, ном_макс, ном
  тек_длина := 0
  макс_длина := 0
  ном_подстр := 0
  нц для ном от 1 до длин(строка)
    если строка[ном] = "я"
      то
        тек_длина := тек_длина + 1
      иначе
        если ном > 1  и строка[ном - 1] = "я"
          то | Цепочка закончилась (только в этом случае)
            | Определяем номер закончившейся цепочки
            ном_подстр := ном_подстр + 1
            | Проверяем длину последней цепочки букв "я"
            если тек_длина > макс_длина
              то | Запоминаем значение двух величин
                макс_длина := тек_длина
                ном_макс := ном_подстр
            все
            тек_длина := 0
        все
    все
  кц
  | Проверяем, не закончилась ли строка цепочкой букв "я"
  если строка[длин(строка)] = "я" | Только в этом случае
    то
      ном_посл := ном_посл + 1     | увеличиваем номер цепочки
      | Проверяем длину последней цепочки букв "я"
      если тек_длина > макс_длина
```

```
        то
          макс_длина := тек_длина  | Значение макс_длина
                                   | можно не запоминать
          ном_макс := ном_подстр
      все
  все
  вывод ном_макс
кон
```

Язык Паскаль

```
BEGIN
  readln(stroka);
  tek_dlina := 0;
  max_dlina := 0;
  nom_podstr := 0;
  for nom := 1 to length(stroka) do
    if stroka[nom] = 'я' then
      tek_dlina := tek_dlina + 1
    else
      if nom > 1 and stroka[nom - 1] = 'я' then
        begin
          nom_podstr := nom_podstr + 1;
          if tek_dlina > max_dlina then
            begin
              max_dlina := tek_dlina;
              nom_max := nom_podstr
            end;
          tek_dlina := 0;
  if stroka[length(stroka)] = 'я' then
    begin
      nom_podstr := nom_podstr + 1;
      if tek_dlina > max_dlina then
        nom_max := nom_podstr
    end;
  writeln(nom_max)
END.
```

Язык Python

```
stroka = input()
tek_dlina = 0
max_dlina = 0
```

```
nom_podstr = 0
for nom in range(len(stroka)):
  if stroka[nom] == "я":
    tek_dlina = tek_dlina + 1
  else:
    if nom > 0 and stroka[nom - 1] == "я":
      nom_podstr = nom_podstr + 1
      if tek_dlina > max_dlina:
        max_dlina = tek_dlina
        nom_max = nom_podstr
      tek_dlina = 0
if stroka[len(stroka) - 1] == "я":
  nom_podstr = nom_podstr + 1
  if tek_dlina > max_dlina:
    nom_max = nom_podstr
print(nom_max)
```

2.12.7. Нахождение порядкового номера подстроки максимальной длины. Второй вариант задачи

Пример задачи: «Дана строка символов. Определить:

а) количество символов в самой длинной подстроке из одинаковых символов;

б) символ в соответствующей подстроке;

в) порядковый номер соответствующей подстроки среди всех подстрок одинаковых символов (в том числе состоящих из одного символа).

Известно, что такая подстрока – единственная. Каждое из искомых значений вывести на отдельной строке.

Например, для массива

```
ааббаввиаааааии
```

на экран должно быть выведено:

```
5
а
6
```

Программу решения указанной задачи разработайте самостоятельно.

Задача нахождения порядкового номера подстроки минимальной длины решается аналогично двум вариантам, рассмотренным в пп. 2.12.6 и 2.12.7.

2.13. Задание 25

В качестве проверяемого элемента содержания для задания 25 в Спецификации [13] указывается: «Умение создавать собственные программы (10–20 строк) для обработки целочисленной информации».

В общем виде один из вариантов задания формулируется так [1]: «Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [...; ...], числа, имеющие ровно ... различных делителей. Вывести эти делители для каждого найденного числа в порядке возрастания».

Пример 1. Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [103446; 103495], числа, имеющие ровно 4 различных делителя. Выведите эти делители для каждого найденного числа в порядке возрастания.

Общая схема программы решения задач указанного типа следующая:

```
цикл для каждого числа из заданного диапазона
   Определить количество его делителей и записать их в массив
   если указанное в задаче условие соблюдается
     то
        Вывести требуемое количество делителей текущего числа
   все
конец цикла
```

Задача определения количества делителей числа и записи их в массив была рассмотрена в п. 1.1.10. С ее учетом программа решения задачи примера 1 оформляется следующим образом:

```
алг Задача_примера_1
нач цел n, возм_дел, кол_дел, i, цел таб делители[1 : 2000]
  нц для n от 103446 до 103495
    кол_дел := 0
    | Проверяем числа до половины значения n
    нц для возм_дел от 1 до div(n, 2)
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
          делители[кол_дел] := возм_дел
      все
    кц
    | Учитываем в качестве делителя число n
    кол_дел := кол_дел + 1
    делители[кол_дел] := n
```

```
    если кол_дел = 4  | Только в этом случае
      то
        | Выводим делители текущего числа из массива
        вывод нс
        нц для i от 1 до 4
          вывод делители[i], " "
        кц
    все
  кц
кон
```

Обратим внимание на то, что для каждого проверяемого числа обновлять («обнулять») массив необходимости нет.

Язык Паскаль

```
Var n, vozm_del, kol_del: longint; i: integer;
    deliteli: array [1..2000] of longint;
BEGIN
  for n := 103446 to 103495  do
    begin
      kol_del := 0;
      for vozm_del := 1 to n div 2 do
        if n mod vozm_del = 0 then
          begin
            kol_del := kol_del + 1;
            deliteli[kol_del] := vozm_del
          end;
      kol_del := kol_del + 1;
      deliteli[kol_del] := n
      if kol_del = 4 then
         begin
           writeln;
           for i := 1 to 4 do
             write(deliteli[i], ' ')
         end
  end
END.
```

Язык Python

В программе на этом языке для заполнения массива deliteli удобно использовать метод append. При этом для каждого проверяемого числа обновлять массив (делать его «пустым») нужно:

```
for n in range(103446, 103496):
  deliteli = []
  for vozm_del in range(1, n//2 + 1):
     if n % vozm_del == 0:
       kol_del = kol_del + 1
       deliteli.append(vozm_del)
  # Учитываем в качестве делителя число n
  deliteli.append(n)
  if len(deliteli) == 4:
    print(deliteli)   # Можно вывести делители именно так
```

Можно величину kol_del не использовать, а после заполнения списка проверять количество его элементов с помощью функции len:

```
for n in range(103446, 103496):
  deliteli = []
  for vozm_del in range(1, n\\2 + 1):
     if n % vozm_del == 0:
       deliteli.append(vozm_del)
  # Учитываем в качестве делителя число n
  deliteli.append(n)
  if len(deliteli) == 4:
    print(deliteli)
```

Недостатками описанной программы являются необходимость использования массива с «запасом» (в программах на школьном алгоритмическом языке, языке Паскаль и ряде других), а также получение всех делителей проверяемых чисел. Можно записывать в массив только 4 делителя (или меньше, если их должно быть столько), а после нахождения пятого делителя прекратить проверку. Это позволит использовать массив из четырех элементов. Прежде чем представлять программы, обратим внимание на то, что в таком варианте надо проверять в качестве возможных делителей все значения:

```
алг Задача_примера_1_вариант_2
нач цел n, возм_дел, кол_дел, i, цел таб делители[1 : 4]
  нц для n от 103446 до 103495
    кол_дел := 0
    нц для возм_дел от 1 до n
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
        | Записываем только не более четырех делителей
```

```
          если кол_дел <= 4
            то
              делители[кол_дел] := возм_дел
          иначе     | Встретился 5-й делитель
            выход   | Выходим из цикла поиска делителей
          все
      все
    кц
    если кол_дел = 4
      то
        вывод нс
        нц для i от 1 до 4
          вывод делители[i], " "
        кц
    все
  кц
кон
```

Язык Паскаль

```
Var n, vozm_del, kol_del: longint; i: integer;
    deliteli: array [1..4] of longint;
BEGIN
  for n := 103446 to 103495  do
    begin
      kol_del := 0;
      for vozm_del := 1 to n do
        if n mod  vozm_del = 0 then
          begin
            kol_del := kol_del + 1;
            if kol_del <= 4 then
              deliteli[kol_del] := vozm_del
            else
              break
          end;
      if kol_del = 4 then
         begin
           writeln;
           for i := 1 to 4 do
             write(deliteli[i], ' ')
         end
  end
END.
```

Язык Python

```
for n in range(103446, 103496):
  deliteli = []
  for vozm_del in range(1, n + 1):
    if n % vozm_del == 0:
      if len(deliteli) <= 4:
        deliteli.append(vozm_del)
      else:
          break
  if len(deliteli) == 4:
    print(deliteli)
```

Ответ

```
1 3 34483 103449
1 7 14779 103453
1 307 337 103459
1 3 34487 103461
1 157 659 103463
1 5 20693 103465
1 107 967 103469
1 7 14783 103481
1 239 433 103487
1 37 2797 103489
```

Возможны также варианты задач, в которых требуется указанное количество делителей вывести в порядке их (делителей) убывания. Пример: «Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [190201; 190230], числа, имеющие ровно 4 различных делителя. Выведите эти четыре делителя для каждого найденного числа в порядке убывания» ([16], автор задачи А. Н. Носкин).

Подобные задачи можно решить разными способами.

Во-первых, можно в программе Задача_примера_1 рассматривать возможные делители, начиная с числа n:

```
алг
нач цел n, возм_дел, кол_дел, i, цел таб делители[1 : 2000]
  нц для n от 103446 до 103495
  кол_дел := 0
  нц для возм_дел от n до 1 шаг -1
    если mod(n, возм_дел) = 0
      то
```

```
        кол_дел := кол_дел + 1
        делители[кол_дел] := возм_дел
    все
  кц
  ...
```

Язык Паскаль

```
Var n, vozm_del, kol_del: longint; i: integer;
    deliteli: array [1..2000] of longint;
BEGIN
  for n := 103446 to 103495  do
    begin
      kol_del := 0;
      for vozm_del := n downto 1 do
        ...
```

Язык Python

```
for n in range(103446, 103496):
  deliteli = []
  for vozm_del in range(n, 0, -1):
    ...
```

Можно также при приведенной в программе Задача_примера_1 очередности проверки возможных делителей изменить порядок вывода делителей из массива:

```
...
если кол_дел = 4
  то
    | Выводим делители текущего числа
    вывод нс
    нц для i от 4 до 1 шаг -1 | начиная с последнего элемента массива
      вывод делители[i], " "
    кц
все
```

В обоих приведенных вариантах решения обсуждаемой задачи также можно провести оптимизацию, связанную с записью в массив не более чем четырех элементов.

Отдельно рассмотрим задачи, в которых идет речь о количестве делителей, удовлетворяющих некоторому условию (четных и т. п.).

Пример 2. Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [103446; 103495], числа, имеющие ровно 4 различных четных делителя. Выведите эти делители для каждого найденного числа в порядке возрастания.

Решение

Здесь в программах следует уточнить, что учитываются только четные делители:

```
алг Задача_примера_2
  …
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0 | Встретился делитель числа n
      то | Проверяем, является ли он четным
        если mod(возм_дел, 2) = 0 | Только в этом случае
          | записываем его в массив
          кол_дел := кол_дел + 1
          делители[кол_дел] := возм_дел
        все
    все
кц
…
```

Язык Паскаль

```
…
for n := 103446 to 103495  do
    begin
      kol_del := 0;
      for vozm_del := 1 to n do
        if (n mod vozm_del = 0) and (vozm_del mod 2 = 0) then
          { Использовано сложное условие }
          begin
            …
```

Язык Python

```
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0 and vozm_del % 2 == 0:
    …
```

Вариант с проверкой до половины значения n:

```
алг Задача_примера_2_вариант_2
  …
  кол_дел := 0
  нц для возм_дел от 2 до div(n, 2) | Число 1 не учитываем
    если mod(n, возм_дел) = 0 и mod(возм_дел, 2)
      …
  кц
    | Если максимальное число заданного диапазона является четным,
    | то учитываем и его
    …
    если кол_дел = 4
      то
        …
```

Язык Паскаль

```
…
for n := 103446 to 103495  do
  begin
    kol_del := 0;
    for vozm_del := 2 to n div 2 do
      if n mod vozm_del = 0 then
        begin
          …
        end;
    { Если максимальное число заданного диапазона является четным,
    то учитываем и его }
    …
        if kol_del = 4 then
      …
```

Язык Python

```
for n in range(103446, 103496):
  deliteli = []
  for vozm_del in range(1, n//2 + 1):
    …
  # Если максимальное число заданного диапазона является четным,
  # то учитываем и его
  …
  if len(deliteli) == 4:
    …
```

Вариант с ограничением количества:

```
алг Задача_примера_2_вариант_3
  …
  кол_дел := 0
  нц для возм_дел от 1 до n
    если mod(n, возм_дел) = 0 и mod(возм_дел, 2)
      то
        кол_дел := кол_дел + 1
        если кол_дел <= 4
          то
            делители[кол_дел] := возм_дел
          иначе
            выход
       все
    все
  кц
…
```

Язык Паскаль

```
…
for n := 103446 to 103495  do
  begin
    kol_del := 0;
    for vozm_del := 1 to n do
      if (n mod vozm_del = 0) and (vozm_del mod 2 = 0) then
        { Использовано сложное условие }
        begin
          kol_del := kol_del + 1;
          if kol_del <= 4 then
         deliteli[kol_del] := vozm_del
       else
         break
     end;
 if kol_del = 4 then
   …
```

Язык Python

```
for n in range(103446, 103496):
  deliteli = []
```

```
for vozm_del in range(1, n + 1):
  if n % vozm_del == 0 and vozm_del % 2 == 0:
    if len(deliteli) <= 4:
      deliteli.append(vozm_del)
if len(deliteli) == 4:
  print(deliteli)
```

Конечно, и в таких задачах (с делителями, удовлетворяющими некоторому условию) возможно требование о выводе делителей в порядке убывания (см. выше).

Задачи для самостоятельной работы

См. [16], файл kege5.doc.

Разработайте также программы, в которых применительно к условиям приведенных в [16] задач определяется:

- среднее арифметическое таких чисел (ответ должен быть записан с одной значащей цифрой после запятой);
- разность между максимальным и минимальным из них;
- медиана указанного в условии множества чисел.

Дополнение

Возможны также задания, связанные с нахождением так называемых простых чисел – чисел, не имеющих других делителей, кроме самого себя и единицы.

В общем виде задача формулируется так: «Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [...; ...], простые числа. Выведите все найденные простые числа в порядке возрастания, слева от каждого числа выведите его порядковый номер».

Общая схема простейшего варианта программы решения задач указанного типа такая:

```
1. Присваивание начального значения переменной - порядковому номеру
   простого числа
2. цикл для каждого числа из заданного диапазона
   2.1. Определение количества его делителей
   2.2. Проверка
   если это количество равно 2
    то    | Найдено очередное простое число
      Определить его порядковый номер (увеличить текущее значение на 1)
      Вывести порядковый номер и значение простого числа
   все
конец цикла
```

Соответствующая программа, учитывающая методику подсчета количества делителей натурального числа (см. п 1.1.9), имеет вид:

```
алг
нач цел n, возм_дел, кол_дел, номер
  номер := 0 | Порядковый номер найденного простого числа
             | (начальное значение)
  нц для n от … до …          | Числа заданного в условии отрезка
    нц для возм_дел от 1 до n
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
      все
    кц
    если кол_дел = 2        | Встретилось очередное простое число
      то
        номер := номер + 1 | Его порядковый номер
        | Выводим требуемые значения
        вывод нс, номер, " ", n
    все
  кц
кон
```

Язык Паскаль

```
Var n, vozm_del: longint; kol_del, nomer: integer;
BEGIN
  nomer := 0;
  for n := … to … do
    begin
      for vozm_del := 1 to n do
        if n mod vozm_del = 0 then
          kol_del := kol_del + 1;
      if kol_del = 2 then
        begin
          nomer := nomer + 1;
          writeln(nomer, ' ', n)
        end
  end
END.
```

Язык Python

```
nomer = 0
for n in range(…, …):
  kol_del = 0
  for vozm_del in range(1, n + 1):
     if n % vozm_del == 0:
       kol_del = kol_del + 1
  if kol_del == 2:
    nomer = nomer + 1
    print(nomer, n)
```

Обратим внимание на то, что задачах обсуждаемого типа могут использоваться достаточно большие значения числа n, что может привести к значительному времени выполнения программы. С целью ускорения работы программы можно применять усовершенствованные методы, описанные в п. 1.1.9. Приведем пример с прекращением подсчета количества делителей, когда это количество превысит 2:

```
алг
нач цел n, возм_дел, кол_дел, номер
  номер := 0
  нц для n от … до …
    кол_дел := 0
    нц для возм_дел от 1 до n
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
          если кол_дел = 3
            то
              выход
          все
      все
    кц
    если кол_дел = 2
      то
        номер := номер + 1
        вывод нс, номер, " ", n
    все
  кц
кон
```

Язык Паскаль

```
Var n, vozm_del: longint; kol_del, nomer: integer;
BEGIN
  nomer := 0;
  for n := … to … do
    begin
      for vozm_del := 1 to n do
        if n mod vozm_del = 0 then
          begin
            kol_del := kol_del + 1;
            if kol_del = 3 then break
          end;
      if kol_del = 2 then
        begin
          nomer := nomer + 1;
          writeln(nomer, ' ', n)
        end
  end
END.
```

Язык Python

```
nomer = 0
for n in range(…, …):
  kol_del = 0
  for vozm_del in range(1, n + 1):
     if n % vozm_del == 0:
       kol_del = kol_del + 1
       if kol_del == 3:
         break
  if kol_del == 2:
    nomer = nomer + 1
    print(nomer, n)
```

В заключение заметим, что возможны также варианты задачи рассмотренного типа, в которых требуется определить не все простые числа, а их максимальное/минимальное значение, или их сумму, или среднее арифметическое, или медиану их набора. Такие варианты задач решаются аналогично рассмотренным в разделе 2.7. Возможно также требование о выводе простых чисел в порядке убывания.

Задачи для самостоятельной работы

См. [16], файл kege6.doc.

2.14. Задание 26

В качестве проверяемого элемента содержания для задания 26 в Спецификации [13] указывается: «Умение обрабатывать целочисленную информацию с использованием сортировки».

Обратим внимание на то, что задание на использование сортировки как отдельное включено в демонстрационный вариант ЕГЭ впервые (см. раздел 3.13), и других заданий пока нет. Обсудим задачу, предложенную автором[1].

Условие

Роботу по конвейеру поступают корзины с монетами. В каждой корзине может быть от 1 до 99 монет. Робот складывает монеты в ящики. Задача робота заполнить как можно большее количество ящиков монетами в количестве 100 штук.

Известно, что робот может высыпать в каждый ящик один раз содержимое не более двух корзин.

Необходимо написать программу, которая определяет две самые «дорогие» корзины, содержимым которых можно заполнить ящики – 100 монетами (чем больше монет в корзине, тем она «дороже»).

Входные данные представлены в файле следующим образом. В первой строке записано число $1 < N < 10000$ – количество корзин, в каждой из последующих N строк целое число $0 < K < 100$ – количество монет в каждой корзине.

В качестве ответа дать два числа – количество монет в двух самых «дорогих» корзинах, которыми можно заполнить ящики 100 монетами (в порядке возрастания этих количеств).

Пример организации исходных данных во входном файле:

```
7
10
44
66
90
65
47
34
```

[1] Основа задачи заимствована из [12].

При таких исходных данных можно заполнить только два ящика по 100 монет: из корзин с 10 и 90 монетами и из корзин с 66 и 34 монетами.

Две корзины с самым большим количеством монет – 66 и 90.

Решение

Идея решения такая – записать все значения количества монет в каждой корзине в массив, после чего рассмотреть все пары элементов массива и найти такие значения, сумма которых равно 100. Для исходных данных, приведенных в условии, это будут числа 10, 90, 66 и 34.

Записав найденные значения в массив, после его сортировки в порядке возрастания вывести на экран два максимальных значения массива[1].

Используем в программе следующие основные переменные величины (кроме количества корзин N):

кол_монет_в_корзине – массив с количеством монет в каждой корзине;

кол_корзин – общее число корзин, монетами которых можно заполнить ящики ровно 100 монетами;

подходят – массив со значениями количества монет в подходящих корзинах.

Задача рассмотрения всех пар элементов массива рассматривалась в п. 1.6.5:

```
нц для i от 1 до N - 1
  нц для j от 2 до N
    ...
  кц
кц
```

Однако мы не можем для некоторой пары корзин с индексами i и j сказать, что если

```
кол_монет_в_корзине[i] + кол_монет_в_корзине[j] = 100
```

то i-я и j-я корзина подходят, потому что корзина с индексом j могла ранее подойти к другой i-й корзине.

Чтобы учесть это обстоятельство, введем массив подходят с данными логического типа (или принимающими значения 0

[1] Можно также определить два максимальных значения массива без его сортировки (см. п. 1.4.9).

и 1), в котором будем фиксировать факт использования корзины в некоторой паре.

Прежде чем представлять программу, заметим, что для сортировки массива подходят использован метод «сортировки» (метод обмена)[1] – см. приложение 3.

```
алг
нач цел N, цел таб кол_монет_в_корзине[1:9999]
    кол_корзин, i, j, подходят[1:9999]
    лог таб использ[1:9999]
    цел номер_прохода, лев, всп
  | Чтение из файла значения N
  …
  | Чтение из файла значений количества монет в корзинах
  | и запись их в массив кол_монет_в_корзине
  …
  | Заполнение массива использ начальными значениями
  нц для i от 1 до N
    использ[i] := нет  | Сначала ни одна корзина не использована
  кц
  кол_корзин := 0
  нц для i от 1 до N - 1
    нц для j от 2 до N
      если кол_монет_в_корзине[i] + кол_монет_в_корзине[j ] = 100 и
           использ[i] = нет и использ[j] = нет
        то
          | Записываем 2 найденных значения в массив подходят
          кол_корзин := кол_корзин + 1
          подходят[кол_корзин] := кол_монет_в_корзине[i]
          кол_корзин := кол_корзин + 1
          подходят[кол_корзин] := кол_монет_в_корзине[j]
          | Учитываем, что эти корзины уже использованы
          использ[i] := да
          использ[j] := да
      все
    кц
  кц
  | Сортируем массив подходят
  нц для номер_прохода от 1 до кол_корзин - 1
    нц для лев от 1 до кол_корзин - номер_прохода
      если подходят[лев] > подходят[лев + 1]
        то | Проводим обмен
```

[1] Можно применить любой другой способ сортировки.

```
            всп := подходят[лев]
            подходят[лев] := подходят[лев + 1]
            подходят[лев + 1] := всп
        все
      кц
    кц
    | Выводим искомые значения
    вывод нс, подходят[кол_корзин - 1], " ", подходят[кол_корзин]
кон
```

Примечание

Оператор использ[i] := да можно не записывать, так как i-я корзина не может быть рассмотрена более одного раза (убедитесь в этом!).

Язык Паскаль

```
Var
  N, kol_korzin, i, j, nomer_prohoda, lev, vsp: integer;
  kol_monet_v_korzine, podhodit: array [1..9999] of integer;
  ispolz: array [1..9999] of boolean;
BEGIN
  { Чтение из файла значения N }
  … (см. раздел 2.11)
  { Чтение из файла значений количества монет в корзинах
  и запись их в массив kol_monet_v_korzine }
  … (см. раздел 2.11)
  { Заполнение массива ispolz начальными значениями }
  for i := 1 to N do
    ispolz[i] := false; { Сначала ни одна корзина не использована }
  kol_korzin := 0;
  for i := 1 to  N - 1 do
    for j := 2 to N do
      if (kol_monet_v_korzine[i] + kol_monet_v_korzine[j] = 100) and
              ispolz[i] = false и ispolz[j] = false then
        begin
          { Записываем 2 найденных значения в массив podhodit }
          kol_korzin := kol_korzin + 1;
          podhodit[kol_korzin] := kol_monet_v_korzine[i];
          kol_korzin := kol_korzin + 1;
          podhodit[kol_korzin] := kol_monet_v_korzine[j];
          { Учитываем, что эти корзины уже использованы }
          ispolz[i] := True ;
          ispolz[j] := True
        end;
```

```
  { Сортируем массив podhodit }
  for nomer_prohoda := 1 to kol_korzin - 1 do
    for lev := 1 to kol_korzin - nomer_prohoda do
      if podhodit[lev] > podhodit[lev + 1] then
        begin
          { Проводим обмен }
          vsp := podhodit[lev];
          podhodit[lev] := podhodit[lev + 1];
          podhodit[lev + 1] := vsp
        end;
  { Выводим искомые значения }
  writeln(podhodit[kol_korzin - 1], " ", podhodit[kol_korzin])
END.
```

Язык Python

В программе на этом языке заполнение массива podhodit можно провести, используя метод append, а сортировку массива – с помощью метода sort:

```
# Чтение из файла значения N
… (см. раздел 2.11)
# Чтение из файла значений количества монет в корзинах
и запись их в массив kol_monet_v_korzine
… (см. раздел 2.11)
# Заполнение массива ispolz начальными значениями
ispolz = [False for i in range(N)] # Сначала ни одна корзина
                                   # не использована
kol_korzin = 0
for i in range(N - 1):  # Нумерация элементов начинается с нуля
  for j in range(1, N):
    if kol_monet_v_korzine[i] + kol_monet_v_korzine[j] == 100 and
        ispolz[i] == False и ispolz[j] == False:
      {Записываем 2 найденных значения в массив podhodit
      podhodit.append(kol_monet_v_korzine[i])
      podhodit.append(kol_monet_v_korzine[j])
      # Увеличиваем значение kol_korzin на 2
      kol_korzin = kol_korzin + 2
      # Учитываем, что эти корзины уже использованы
      ispolz[i] = True
      ispolz[j] = True
podhodit.sort()
# Выводим искомые значения
print(podhodit[kol_korzin - 2], podhodit[kol_korzin - 1])
```

2.15. Задание 27

В качестве проверяемого элемента содержания для задания 27 в Спецификации [13] указывается: «Умение создавать собственные программы (20–40 строк) для анализа числовых последовательностей».

Анализ заданий 27 из вариантов экзамена в некомпьютерной форме нескольких последних лет показывает, что для них трудно выделить ряд типовых задач. Обсудим задания из [5–7]. Методика выполнения других заданий 27 описана в [21].

2.15.1. Задание из [5]

Условие

На вход программы поступает последовательность из n целых положительных чисел. Рассматриваются все пары элементов последовательности a_i и a_j, такие что i < j и $a_i > a_j$ (первый элемент пары больше второго; i и j – порядковые номера чисел в последовательности входных данных). Среди пар, удовлетворяющих этому условию, необходимо найти и напечатать пару с максимальной суммой элементов, которая делится на m = 120. Если среди найденных пар максимальную сумму имеет несколько, то можно напечатать любую из них.

Описание входных и выходных данных

В первой строке входных данных задается количество чисел n ($2 \leqslant n \leqslant 12\,000$). В каждой из последующих n строк записано одно целое положительное число, не превышающее 10 000.

В качестве результата программа должна напечатать элементы искомой пары. Если таких пар несколько, можно вывести любую из них.

Гарантируется, что хотя бы одна такая пара в последовательности есть.

Пример входных данных:

```
6
60
140
61
100
300
59
```

Пример выходных данных для приведенного выше примера входных данных:

```
140 100
```

Пояснение. Из шести заданных чисел можно составить три пары, сумма элементов которых делится на m = 120: 60 + 300, 140 + 100 и 61 + 59. Во второй и третьей из этих пар первый элемент больше второго, но во второй паре сумма больше.

Решение

Задача может быть решена так.

Обозначим пару искомых значений:

1) лев – расположенное в последовательности раньше;

2) прав – расположенное в последовательности позже.

Запишем все n чисел последовательности в массив (имя массива – мас), после чего сравним все пары элементов мас[i] и мас[j], таких, что i < j. Если мас[i] > мас[j], сумма этих значений делится нацело на m = 120 и больше суммы «старых» значений величин лев и прав, то значения мас[i] и мас[j] принимаются в качестве новых значений, соответственно, величин лев и прав.

Соответствующая программа на школьном алгоритмическом языке:

```
цел m
m := 120
алг
нач цел n, лев, прав, i, j, цел таб мас[1 : 1200]
  ввод n
  | Ввод чисел последовательности и запись их в массив
  нц для i от 1 до n
    ввод мас[i]
  кц
  | Начальное присваивание значений искомым величинам
  лев := 0
  прав := 0
  | Перебор и проверка всех пар значений элементов массива
  нц для i от 1 до n - 1
    нц для j от i + 1 до n
      если мас[i] > мас[j] и mod(мас[i] + мас[j], m) = 0
        то | Сравниваем суммы
          если мас[i] + мас[j] > лев + прав
            то | Встретилась новая пара искомых значений
              | Запоминаем их
```

```
            лев := мас[i]
            прав := мас[j]
        все
    все
  кц
кц
| Вывод ответа
вывод лев, " ", прав
кон
```

Опишем также более эффективный вариант программы.

Прежде всего сделаем важное напоминание, которое будет лежать в основе решения задачи: «Сумма двух чисел делится на m, если сумма остатков этих чисел от деления на m равна m или 0».

Как и ранее, обозначим пару искомых значений:

1) лев – расположенный в последовательности раньше;
2) прав – расположенный в последовательности позже.

Допустим, что первые 11 чисел последовательности такие:

114 0 120 255 240 76 234 720 56 240 7

Пусть очередное число (его имя – очер) равно 120. Какие числа среди рассмотренных десяти могут, так сказать, составить пару этому числу с точки зрения решения задачи? Так как остаток от деления числа 120 на m равен нулю, то ответ такой: 0, 120, 240, 720 и 240 (поскольку у них соответствующий остаток также равен нулю). Но среди них нас интересует *максимальное* из чисел, больших очередного очер (мы ищем пару чисел с максимальной суммой элементов). Таким числом является 720. Означает ли это, что встретилась новая пара искомых чисел лев и прав? Для ответа на этот вопрос следует сравнить сумму чисел 720 и 120 с суммой значений лев и прав для первых 11 чисел последовательности. Анализ показывает, что нет (было лев = 720 и прав = 240, т. е. сумма была больше).

Если бы очередное число было равно, например, 360, то тогда бы значения лев и прав изменились бы на 720 и 360 соответственно.

Заметим также, что если очередное число равно, например, 960, то, возможно, оно когда-то станет значением переменной лев.

Во всех случаях, когда значение очер кратно 120, следует знать максимальное из рассмотренных ранее чисел, также кратных 120. Рассмотрим теперь случай, когда значение очер не кратно 120, например, равно 6. При этом остаток от деления на 120 равен 6, и,

согласно отмеченному выше правилу кратности суммы двух чисел, следует рассмотреть предшествующие числа, остаток от деления которых на 120 равен 120 – 6 = 114. Таких чисел два – 114 и 234. Оба они больше 6; из них максимальным является 234. Появилась ли новая пара чисел лев и прав? Сравним суммы. Сумма 234 + 6 = 240 меньше суммы значений «старой» пары (720 + 240 = 960). Значит, текущие значения искомых величин лев и прав не меняются. А если бы среди первых 10 чисел было, например, число 1334 (остаток от его деления на 120 равен 114):

1334 0 120 255 240 76 234 720 56 240 7

– тогда сумма 1334 + 6 = 1440 была бы больше 960, т. е. встретилась бы пара новых значений лев = 1334 и прав = 6.

Итак, можем сделать вывод, что для решения задачи для каждого очередного числа последовательности очер, имеющего остаток от деления на m (120), равный остат, надо знать:

если остат равен 0,
то
максимальное из уже рассмотренных чисел, остаток от деления которых на m также равен 0,
иначе
максимальное из уже рассмотренных чисел, остаток от деления которых на m равен m – остат
все.

Всего нужно знать 120 значений. Для их хранения следует использовать массив из 120 элементов с индексами от 0 до 119 (от 0 до m – 1). В приведенной ниже программе решения задачи на школьном алгоритмическом языке имя этого массива – макс_числа_с_остатком. Это значит, что, например, в элементе массива с индексом 13 будет храниться максимальное из уже рассмотренных чисел, остаток от деления которых на m равен 13 и т. п.

Пример массива в какой-то момент обработки чисел последовательности:

720	0	0	123		1334		0	1319
0	1	2	3	...	114	...	118	119

Прежде чем представлять программу, заметим, что после обработки очередного числа его, возможно, следует записать в со-

ответствующий элемент массива макс_числа_с_остатком (если оно больше записанного там значения).

```
цел m
m := 120
алг
нач цел n, очер, лев, прав, остат, i
    цел таб макс_числа_с_остатком[0 : m - 1]
  ввод n
  | Обнуление элементов массива
  нц для i от 0 до m - 1
    макс_числа_с_остатком[i] := 0
  кц
  | Начальное присваивание значений искомым величинам
  лев := 0
  прав := 0
  | Ввод и обработка чисел последовательности
  нц для i от 1 до n
    ввод очер
    остат := mod(очер, m)
    если остат = 0
      то | Сравниваем значение очер
         | с соответствующим элементом массива
        если очер < макс_числа_с_остатком[0] | Только в этом случае
          то | Сравниваем суммы
            если макс_числа_с_остатком[0] + очер > лев + прав
              то | Встретилась новая пара искомых значений
                 | Запоминаем их
                 лев := макс_числа_с_остатком[0]
                 прав := очер
            все
        все
      иначе | Остаток не равен 0
        если очер < макс_числа_с_остатком[m - остат]
             | Только в этом случае
          то | Сравниваем суммы
            если макс_числа_с_остатком[m - остат] + очер > лев + прав
              то | Встретилась новая пара искомых значений
                 | Запоминаем их
                 лев := макс_числа_с_остатком[m - остат]
                 прав := очер
            все
        все
    все
```

```
    | После обработки очередного числа может измениться
    | значение элемента массива с индексом остат
    если очер > макс_числа_с_остатком[остат]
      то
        макс_числа_с_остатком[остат] := очер
    все
  кц
  | Вывод ответа
  вывод лев, " ", прав
кон
```

Примечания

1. Вместо вложенных условных операторов можно использовать сложные условия:

```
остат = 0 и очер < макс_числа_с_остатком[0] и макс_числа_с_
остатком[0] + очер > лев + прав
```

и

```
очер < макс_числа_с_остатком[m - остат] и макс_числа_с_остатком[m -
остат] + очер > лев + прав
```

2. Можно при обработке чисел последовательности не рассматривать два варианта значения остат (не использовать полный условный оператор), а объединить их в общий случай. Это можно сделать двумя способами:

1) заменить значение остат, равное нулю, на значение m и при сравнении использовать элемент массива с индексом m - остат:

```
...
| Ввод и обработка чисел последовательности
нц для i от 1 до n
  ввод очер
  остат := mod(очер, m)
  если остат = 0
    то | Меняем
      остат := m
  все

  | Сравниваем
  если очер < макс_числа_с_остатком[m - остат]
    | Только в этом случае
    то
```

```
      если макс_числа_с_остатком[m - остат] + очер > лев + прав
        то
          лев := макс_числа_с_остатком[m - остат]
          прав := очер
      все
  все
кц
```

В этом случае при возможном изменении массива макс_числа_с_остатком следует рассмотреть два случая:

```
если остат = m
  то
    | Сравниваем с элементов с индексом 0
    если очер > макс_числа_с_остатком[0]
      то
        макс_числа_с_остатком[0] := очер
    все
  иначе
    если очер > макс_числа_с_остатком[m - остат]
      то
        макс_числа_с_остатком[m - остат] := очер
    все
все
```

2) использовать в массиве элемент с индексом mod(m - остат, m). Нетрудно убедиться, что этот индекс обобщает два индекса:

0 при остат = 0

и

m - остат в противном случае.

Если этот индекс обозначить общ_инд, то фрагмент программы, связанный с вводом и обработкой чисел последовательности, примет вид:

```
нц для i от 1 до n

  ввод очер
  остат := mod(очер, m)
  общ_инд := mod(m - остат,  m)
  | Сравниваем
  если очер < макс_числа_с_остатком[общ_инд]
    | Только в этом случае
    то
      если макс_числа_с_остатком[общ_инд] + очер > лев + прав
```

```
        то
          лев := макс_числа_с_остатком[общ_инд]
          прав := очер
      все
  все
кц
```

В этом случае фрагмент, связанный с возможным изменением массива, не меняется:

```
если очер > макс_числа_с_остатком[остат]
  то
    макс_числа_с_остатком[остат] := очер
все
```

2.15.2. Задание из [7]

Условие

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 26.

Описание входных и выходных данных

В первой строке входных данных задается 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 26.

Пример входных данных:

```
4
2
6
13
39
```

Пример выходных данных для приведенного выше примера входных данных:

```
140 100
```

Пояснение. Из четырех заданных чисел можно составить 6 попарных произведений: 2 × 6, 2 × 13, 2 × 39, 6 × 13, 6 × 39, 13 × 39 (результаты: 12, 26, 78, 78, 234, 507). Из них на 26 делятся 4 произведения (2 × 13 = 26; 2 × 39 = 78; 6 × 13 = 78; 6 × 39 = 234).

Решение

Задача может быть решена так. Записываем все входные значения a в массив m, после чего рассматриваем все пары элементов – если произведение их значений кратно 26, то увеличиваем счетчик искомых значений к26 на 1. Соответствующая программа:

```
алг
нач цел N, a, i, j, к26, цел таб мас[1:1000]
  ввод N
  | Ввод значений и запись их в массив
  нц для i от 1 до N
    ввод a
    мас[i] := a
  кц
  к26 := 0
  нц для i от 1 до N - 1
    нц для j от i + 1 до N
      если mod(мас[i] * мас[j], 26) = 0
        то
          к26 := к26 + 1
      все
    кц
  кц
  вывод нс, к26
кон
```

Опишем также решение без записи вводимых значений a в массив.

Прежде всего можем утверждать, что из всех N чисел заданной последовательности нас интересуют 3 группы чисел:

1) кратные 26;
2) кратные 13;
3) кратные 2.

Допустим, что мы знаем количество чисел первой группы (к26). Сколько пар различных чисел они дадут?

Во-первых, каждое число из к26 даст нужное произведение с каждым из остальных (N - к26) чисел, т. е. искомых пар будет: к26 * (N - к26)

Во-вторых, нужное произведение эти числа дадут, так сказать, между собой в количестве к26 * (к26 - 1)/2 пар.

Кроме того, на 26 будут делиться также пары чисел, одно из которых кратно 13, а другое кратно 2. Если количество первых равно к13, а вторых – к2, то искомое число пар равно:

```
к13 * к2
```

Правда, при подсчете значений к13 и к2 не должны быть учтены числа, уже учтенные раньше (кратные 26). В программе это можно сделать с помощью такой вложенной команды **если** (вложенного условного оператора):

```
если mod(a, 26) = 0  | Число кратно 26
  то | Увеличиваем значение к26
    к26 := к26 + 1
  иначе
    если mod(a, 13) = 0  | Число кратно 13
      то | Увеличиваем значение к13
        к13 := к13 + 1
    все
    если mod(a, 2) = 0  | Число кратно 2
      то | Увеличиваем значение к2
        к2 := к2 + 1
    все
все
```

или такой:

```
если mod(a, 26) = 0
  то
    к26 := к26 + 1
  иначе
    если mod(a, 13) = 0
      то
        к13 := к13 + 1
      иначе
        если mod(a, 2) = 0
          то
            к2 := к2 + 1
        все
    все
все
```

После определения значений к26, к13 и к2 общее искомое значение количества пар чисел к26 будет равно:

```
к26 := к26 * (N - к26) + div(к26 * (к26 - 1), 2) + к13 * к2
```

В заключение заметим, что последний вариант программы применим для решения аналогичных задач, в которых требуется определить количество пар, где произведение элементов кратно числу, являющемуся произведением двух простых чисел, например 22, 34, 39, 77 и т. п.

2.15.3. Задание из [6]

Условие

На вход программы поступает последовательность из N целых положительных чисел, все числа в последовательности различны. Рассматриваются все пары различных элементов последовательности, находящихся на расстоянии не меньше чем 4 (разница в индексах элементов пары должна быть 4 или более, порядок элементов в паре не важен). Необходимо определить количество таких пар, для которых произведение элементов делится на 29.

Описание входных и выходных данных

В первой строке входных данных задается количество чисел N ($4 \leqslant N \leqslant 1000$). В каждой из последующих N строк записано одно целое положительное число, не превышающее 10 000.

В качестве результата программа должна вывести одно число: количество пар элементов, находящихся на расстоянии не меньше чем 4, в которых произведение элементов кратно 29.

Пример входных данных:

```
7
58
2
3
5
4
1
29
```

Пример выходных данных для приведенного выше примера входных данных:

```
5
```

Пояснение. Из семи заданных элементов с учетом допустимых расстояний между ними можно составить 6 произведений: 58 × 4, 58 × 1, 58 × 29, 2 × 1, 2 × 29, 3 × 39. Из них на 29 делятся 5 произведений.

Решение

Данная задача, как и предыдущая, может быть решена путем полного перебора всех пар элементов (в данном случае – отстоящих друг от друга не менее чем на 4 элемента). Записываем все входные значения a в массив m[1], после чего рассматриваем все пары элементов – если произведение их значений кратно 29, то увеличиваем счетчик искомых значений (обозначим его к29) на 1. Соответствующая программа:

```
алг
нач цел N, a, i, j, к29, цел таб мас[1:1000]
  ввод N
  | Ввод значений и запись их в массив
  нц для i от 1 до N
    ввод a
    мас[i] := a
  кц
  к29 := 0
  нц для i от 1 до N - 4
    нц для j от i + 4 до N
      если mod(мас[i] * мас[j], 29) = 0
        то
          к29 := к29 + 1
      все
    кц
  кц
  вывод нс, к29
кон
```

Можно также обрабатывать числа по мере их ввода. Будем учитывать, что произведение двух чисел делится на 29, если хотя бы одно из этих чисел делится на 29.

Рассмотрим некоторый очередной, например 10-й, элемент массива.

Если его значение кратно 29:

4	87	8	13	58	29	116	56	23	**145**
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

то искомое значение k29 увеличится на общее количество чисел, отстоящих от 10-го на 4 и более элементов. Количество таких чисел равно 6 (10 – 4).

[1] Можно переменную a отдельно не вводить, а сразу записывать вводимые числа в массив.

Если его значение не кратно 29:

4	87	8	13	58	29	116	56	23	**146**
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

то искомое значение k29 увеличится на количество чисел, кратных 29, среди элементов, отстоящих от 10-го на 4 и более элементов. В приведенном примере таких чисел – 3 (87, 58 и 29). Обратим внимание на то, что среди них должен быть учтен и элемент, отстоящий от 10-го на 4 позиции (29). Это значит, что 6-й элемент следует проверить до проверки 10-го. Итак, мы пришли к необходимости подсчета количества элементов, кратных 29, находящихся от очередного на 4 и более элементов. Обозначим соответствующую переменную k29_ранее.

Соответствующий фрагмент для некоторого очередного i-го элемента:

```
...
| Ввод значения очередного элемента массива
ввод mас[i]
| Проверка (i - 4)-го элемента
если mod(mас[i - 4], 29) = 0
  то
    k29_ранее := k29_ранее + 1
все
| Проверка очередного элемента
если mod(a[i], 29) = 0
  то
    k29 := k29 + (i - 4)
  иначе
    k29 := k29 + k29_ранее
все
```

Прежде чем представлять всю программу решения задачи, заметим, что так как должны обрабатываться все числа, начиная с 5-го:

```
нц для i от 5 до N
  | Ввод и обработка очередного числа
  ввод очер
  ...
```

то до этого первые 4 числа должны быть уже записаны в массив m до этого.

Итак, программа:

```
алг
нач цел таб мас[1:1000], цел N, к29, к29_ранее, i
  | Ввод общего количества чисел
  ввод N
  | Ввод первых четырех чисел
  нц для i от 1 до 4
    ввод мас[i]
  кц
  | Начальные значения используемых величин
  к29_ранее := 0
     к29 := 0
  | Ввод и обработка остальных чисел
  нц для i от 5 до N
    ввод мас[i]
    если mod(мас[i - 4], 29) = 0
      то
        к29_ранее := к29_ранее + 1
    все
    | Проверка очередного элемента
    если mod(мас[i], 29) = 0
      то
        к29 := к29 + (i - 4)
      иначе
        к29 := к29 + к29_ранее
    все
  кц
  | Вывод ответа
  вывод нс, к29
кон
```

В приведенной программе используется массив, размер которого зависит от N. Указанный недостаток можно устранить, учитывая, что при обработке очередного числа проверяется число, отстоящее от очередного на 4 «позиции». Значит, достаточно хранить в массиве только 4 последних числа. Если имя этого массива принять также мас, а очередное число обозначить – очер, то фрагмент программы, связанный с вводом и обработкой очередного, i-го, числа, оформляется так:

```
...
| Ввод очередного числа
ввод очер
| Проверка 1-го элемента массива (отстоящего от очередного на 4 "позиции" )
```

```
если mod(мас[1], 29) = 0
  то
    к29_ранее := к29_ранее + 1
все
| Проверка очередного числа
если mod(очер, 29) = 0
  то
    к29 := к29 + (i - 4)
  иначе
    к29 := к29 + к29_ранее
все
```

Понятно, что первые 4 числа также должны быть предварительно записаны в массив m.

После ввода и обработки каждого очередного числа этот массив должен быть изменен. Как? Если, например, массив выглядит следующим образом:

8	13	58	29
1	2	3	4

а очередной элемент равен 116, то массив должен стать таким:

13	58	29	116
1	2	3	4

Такая задача решалась в п. 1.6.6.

Вся программа имеет вид:

```
алг
нач цел таб мас[1:4], цел N, очер, к29, к29_ранее, i, j
  | Ввод общего количества чисел
  ввод N
  | Ввод первых четырех чисел
  нц для i от 1 до 4
    ввод мас[i]
  кц
  | Начальные значения используемых величин
  к29_ранее := 0
     к29 := 0
  | Ввод и обработка остальных чисел
  нц для i от 5 до N
    ввод очер
    если mod(мас[1], 29) = 0
      то
```

```
        к29_ранее := к29_ранее + 1
    все
    | Проверка очередного числа
    если mod(очер, 29) = 0
      то
        к29 := к29 + (i - 4)
      иначе
        к29 := к29 + к29_ранее
    все
    | Сдвиг элементов массива влево
     нц для j от 1 до 3 | Индекс i использовать нельзя
      мас[j] := мас[j + 1]
    кц
    | Запись нового (текущего) числа в конец массива
    мас[4] := очер
  кц
  | Вывод ответа
  вывод нс, к29
кон
```

Примечание. Можно вместо использования массива из четырех элементов хранить 4 числа в четырех «обычных» переменных, также меняя их значения аналогично описанному. Конечно, когда величина «расстояния» между учитываемыми числами большая, целесообразно использовать массив.

В заключение заметим, что все описанные варианты программы применимы для решения аналогичных задач – когда требуется определить количество таких пар чисел, отстоящих в последовательности на заданное «расстояние», у которых произведение элементов кратно некоторому числу, являющемуся простым.

Глава 3

Методика выполнения заданий из [1]

3.1. Задание 5

Условие

На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом.

1. Строится двоичная запись числа N.
2. К этой записи дописываются справа еще два разряда по следующему правилу:
   а) складываются все цифры двоичной записи числа N, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;
   б) над этой записью производятся те же действия – справа дописывается остаток от деления суммы ее цифр на 2.

Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью искомого числа R.

Укажите такое наименьшее число N, для которого результат работы данного алгоритма больше числа 77. В ответе это число запишите в десятичной системе счисления.

Решение

Обратим внимание на то, что данная задача отличается от задач, рассмотренных в пп. 2.1.2–2.1.4, тем, что следует найти не число, являющееся результатом работы алгоритма, а входное число алгоритма.

Проанализируем условие. В нем фигурируют три десятичных числа:

1) исходное число N;
2) результат работы алгоритма R;
3) некоторое целое число (в данном случае – 77), которое назовем граничным и обозначим G.

Согласно условию нужно найти такое минимальное значение N, которое в результате выполнение алгоритма даст значение R, большее G.

Если соответствующие двоичные записи перечисленных чисел обозначим $N2$, $R2$ и $G2$, то решение задачи сводится к двум этапам:

1) нахождение такого минимального значения $N2$, которое в результате выполнение алгоритма даст значение $R2$, большее $G2$;
2) определение десятичного эквивалента найденного значения $N2$.

Обсудим первый этап. Здесь также можно использовать утверждение, сделанное в разделе 2.1, – длина (количество цифр) значения *R*2, соответствующего искомому *N*2, должна быть не менее длины числа *G*2. Это значит, что поиск значения *N*2 можно начинать с числа, состоящего из (*k* – 2) цифр значения *R*2.

Начнем решение.

Переведем число 77 в двоичную систему – 1001101. Согласно утверждению, сделанному чуть выше, минимально возможное значение *N*2 равно 10011. Исследуем это значение:

***N*2**	**Кол-во единиц**	**Двоичная запись результата (*R*2)**	**Значение *R*[3]**
10011	Нечетное	1001110	78

Итак, значение 10011 дает нужный результат, т. е. ответ: $N = 19$.

Обратим внимание на то, что в рассмотренном случае найденное число *N*2 имеет вид 10011, что совпадает с укороченной на две цифры двоичной записью заданного граничного числа 77. Однако это не означает, что значение *N*2 (а следовательно, и *N*) можно получить, отбросив две последние цифры двоичной записи заданного в условии числа *G*! Подтвердим сказанное на примере.

Пусть нужно определить такое наименьшее число *N*, для которого результат работы описанного в условии алгоритма больше числа 90.

$90_{10} = 1011010_2$.

Отбросив последние цифры, получим 10110. При этом, согласно алгоритму, результат *R* будет равен 90 ($1011010_2 = 90_{10}$), который не подходит по условию.

Проверка следующего числа дает искомый результат:

***N*2**	**Кол-во единиц**	***R*2**	***R***
10111	Четное	1011100	92

– т. е. правильный ответ – 23 (10111_2).

Итак, алгоритм решения обсуждаемой задачи можно сформулировать следующим образом:

1) по заданному граничному числу *G* получить *G*2;

[1] Это значение можно не определять, а сравнивать *R2* и *G2*.

2) отбросить в $G2$ две последние цифры;
3) принимая новое и последующие за ним двоичные числа в качестве числа $N2$, приписывать к каждому две новые цифры по правилу:

если количество единиц в числе $N2$ нечетное,
то
к нему приписываются цифры 1 и 0,
иначе
к нему приписываются цифры 0 и 0,
все.

Первое число $N2$, дающее результат $R2 > G2$, определяет искомое значение N.

Задания для самостоятельной работы

1. Для условия, приведенного в начале раздела, определите такое наименьшее число N, для которого результат работы данного алгоритма больше числа:

а) 25;
б) 41;
в) 88.

2. На вход алгоритма подается натуральное число N. Алгоритм строит по нему новое число R следующим образом:

1) строится двоичная запись числа N;
2) к этой записи дописываются справа еще два разряда по следующему правилу:
   а) если остаток от деления суммы всех цифр двоичной записи числа N на 2 равен нулю, то в конец числа (справа) дописывается 1, в противном случае дописывается 0. Например, запись 11000 преобразуется в запись 110001;
   б) над новой записью производятся те же действия.

Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N) является двоичной записью искомого числа R.

Укажите такое наименьшее число N, для которого результат работы данного алгоритма больше числа:

а) 21;
б) 56;
в) 83.

3.2. Задание 6

Условие

Определите, при каком наименьшем введенном значении переменной s программа выведет число 64. Для вашего удобства программа представлена на четырех языках программирования.

Алгоритмический язык

```
алг
нач
  цел n, s
  ввод s
  n := 1
  нц пока s < 51
    s := s + 5
    n := n * 2
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  readln (s);
  n := 1;
  while s < 51 do
    begin
      s := s + 5;
      n := n * 2
    end;
  writeln(n)
END.
```

Python

```
s = int(input())
n = 1
while s < 51:
  s = s + 5
  n = n * 2
print(n)
```

C++

```
# include <iostream>
using namespace std;
int main()
{ int s, n;
  cin >> s;
  n = 1 ;
  while (s < 51) { s = s + 5; n = n * 2; }
  cout << n << endl;
  return 0;
}
```

Решение

Анализ программы показывает, что в теле цикла с предусловием происходит многократное умножение на 2, т. е. выводится значение n, являющееся степенью двойки. Для того чтобы было выведено число 64, умножение в цикле должно проводиться 6 раз ($64 = 2^6$). В свою очередь это означает, что увеличение числа s на 5 также должно проходить 6 раз. Наименьшее начальное значение переменной s, при котором тело оператора цикла будет выполняться (условие для продолжения работы оператора – s < 51), равно:

$51 - 5 \times 6 = 21$.

Ответ: 21.

Обобщение метода выполнения задания

Если вместо числа 51 в условии указано некоторое число A, увеличение переменной s происходит с шагом B, а должно быть выведено некоторое число C, являющееся степенью двойки, то искомое значение может быть определено по формуле:

$A - B \times \log_2 C$.

Аналогично, если число C представляет собой степень числа D, то формула принимает вид:

$A - B \times \log_D C$.

Задания для самостоятельной работы

1. Определите, при каком наименьшем введенном значении переменной s программа выведет число 256.

Алгоритмический язык

```
алг
нач цел n, s
  ввод s
  n := 1
  нц пока s < 78
    s := s + 6
    n := n * 2
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  readln (s);
  n := 1;
  while s < 78 do
    begin
      s := s + 6;
      n := n * 2
    end;
  writeln(n)
END.
```

Python

```
s = int(input())
n = 1
while s < 78:
  s = s + 6
  n = n * 2
print(n)
```

2. Определите, при каком наименьшем введенном значении переменной s программа выведет число 128.

Алгоритмический язык

```
алг
нач цел n, s
  ввод s
  n := 1
  нц пока s > 16
    s := s - 4
    n := n * 2
  кц
  вывод n
кон
```

Паскаль

```
var s, n: integer;
BEGIN
  readln (s);
  n := 1;
  while s > 16 do
    begin
      s := s - 4;
      n := n * 2
    end;
  writeln(n)
END.
```

Python

```
s = int(input())
n = 1
while s > 16:
  s = s - 4
  n = n * 2
print(n)
```

4. Установите общую формулу для расчета искомого числа для случая, когда в программах, аналогичных приведенной в предыдущем задании, значение переменной s уменьшается и должно быть выведено некоторое число C, являющееся степенью двойки.

В заключение заметим, что поскольку экзамен будет проходить в компьютерной форме, то результат можно проверить, скопировав программу в соответствующую систему программирования.

3.3. Задание 10

Условие

С помощью текстового редактора определите, сколько раз, не считая сносок, встречается слово «долг» или «Долг» в тексте романа в стихах А. С. Пушкина «Евгений Онегин». Другие формы слова «долг», такие как «долги», «долгами» и т. д., учитывать не следует. В ответе укажите только число.

Задание выполняется с использованием прилагаемого файла.

Решение

Для выполнения следует воспользоваться средствами поиска текстового редактора Microsoft Word. Для этого нужно вызвать окно **Найти и заменить** с активной вкладкой **Найти:** (см. рис. 2.3.5 и 2.3.6). Напомним (см. раздел 2.3), что это можно сделать двумя способами – с помощью панели **Навигация** и кнопки **Заменить** в правой части вкладки **Главная**.

Искомое значение можно получить, просуммировав результаты двух поисков в документе:

1) с регулярным выражением <долг>;
2) с регулярным выражением <Долг>.

В обоих случаях переключатель с надписью **Подстановочные знаки** должен быть включен, при этом во втором случае переключатель с надписью **Учитывать регистр** будет недоступен.

Ответ: 1.

3.4. Задание 12

Условие

Исполнитель Редактор получает на вход строку цифр и преобразовывает ее. Редактор может выполнять две команды, в обеих командах v и w обозначают цепочки цифр.

А) заменить(v, w).

Эта команда заменяет в строке первое слева вхождение цепочки v на цепочку w. Например, выполнение команды

```
заменить(111, 27)
```

преобразует строку 05111150 в строку 0527150.

Если в строке нет вхождений цепочки v, то выполнение команды заменить(v, w) не меняет эту строку.

Б) нашлось(v).

Эта команда проверяет, встречается ли цепочка v в строке исполнителя Редактор. Если она встречается, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Строка исполнителя при этом не изменяется.

Цикл

```
ПОКА условие
  последовательность команд
КОНЕЦ ПОКА
```

выполняется, пока условие истинно.

В конструкции

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

выполняется *команда*1 (если условие истинно) или *команда*2 (если условие ложно).

Какая строка получится в результате применения приведенной ниже программы к строке, состоящей из 70 идущих подряд цифр 8? В ответе запишите полученную строку.

```
НАЧАЛО
  ПОКА нашлось(2222) ИЛИ нашлось(8888)
    ЕСЛИ нашлось(2222)
      ТО заменить(2222, 88)
    ИНАЧЕ заменить(8888, 22)
      КОНЕЦ ЕСЛИ
  КОНЕЦ ПОКА
КОНЕЦ
```

Решение

Таблица, аналогичная использовавшимся в разделе 2.4, для начальных шагов замены будет следующей:

Цифра в начале строки	Меняется (количество цифр)	Не меняется (количество цифр)	Последние цифры строки
8	68	2	88
2	32	2	2288
8	16		
2	8		
8	4		
2	2		

То есть строка примет вид: 222288. После этого замены продолжатся: 222288 → 8888 → 22.

Ответ: 22.

Так как экзамен будет проводиться в компьютерной форме, то для нахождения ответа можно использовать уже упоминавшуюся возможность текстового редактора Microsoft Word поиска и замены текста (см. панель **Найти и заменить** на рис. 2.3.6).

Сначала надо записать в документ исходную строку. Так как цифры 8 в ней повторяются, то можно записать, например, 10 цифр, а затем скопировать ее еще 6 раз. При этом надо проследить, чтобы в полученном тексте был один (!) абзац и не было пробелов:

```
8888888888888888888888888888888888888888888888888888888888888888888888888888
```

Вызывается панель **Найти и заменить** (см. выше) и многократно проводится замена текста.

В поле **Найти** следует записать 8888, в поле **Заменить на** – 22 (см. рис. 3.4.1). При этом активной станет кнопка **Заменить все**, по щелчку на которую произойдет замена всех цепочек 8888 на 22.

После этого, не закрывая панель **Найти и заменить**, провести замены 2222 на 88 или 8888 на 22 (в зависимости от того, имеется ли в текущем документе цепочка 2222) до тех пор, пока в документе не останется цепочек 8888 и 2222. Обратим внимание на то, что тексты, которые уже заменялись, и заменяющие тексты можно повторно использовать, щелкнув по треугольничку в правой части полей **Найти** и **Заменить на** и выбрав нужный текст (см. рис. 3.4.2); это ускоряет процесс замены.

Оставшаяся после всех замен строка будет ответом к заданию.

Рис. 3.4.1

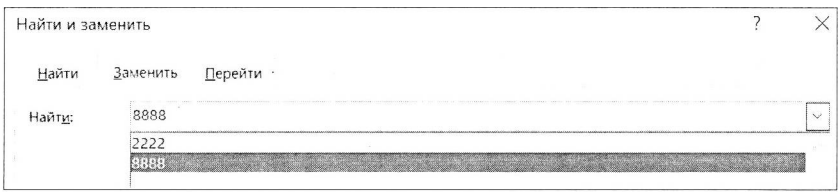
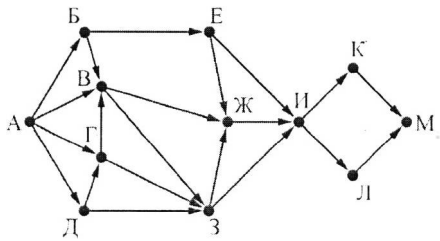


Рис. 3.4.2

3.5. Задание 13

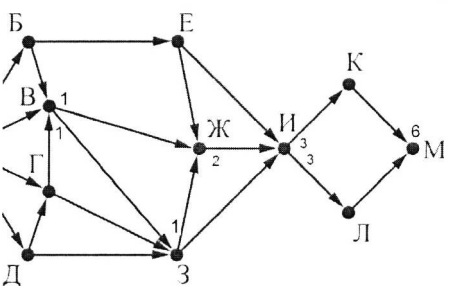
Условие

На рисунке представлена схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, З, И, К, Л, М. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город М, проходящих через город В?



Решение

Как указывалось в разделе 2.5, такая задача разбивается на две подзадачи – подсчет количества путей из города А в город В и из города В в город М. Первое количество можно определить в уме (оно равно 4). Для нахождения второго количества применим «метод Чеширского Кота». Решение показано на рисунке ниже.



Ответ: 4 × 6 = 24.

Примечание. Можно также решить задачу, рассмотрев три участка:

1) от города А до города В;
2) от города В до города И;
3) от города И до города М.

При этом общий результат будет получен так: 4 × 3 × 2 = 24.

3.6. Задание 16

Условие

Алгоритм вычисления значения функции F(n), где n – натуральное число, задан следующими соотношениями:

F(n) = 1 при n = 1;

F(n) = n + F(n – 1), если n – четно,

F(n) = 2 × F(n – 2), если n > 1 и при этом n – нечетно.

Чему равно значение функции F(26)?

Решение

Так как экзамен проводится в компьютерной форме, то для нахождения искомого значения можно разработать программу.

Видно, что функция F – рекурсивная (см. раздел 2.6). На школьном алгоритмическом языке она может быть оформлена в виде:

```
алг цел F (арг цел n)
нач
  если n = 1
    то
      знач := 1
    иначе
      если mod(n, 2) = 0
        то
          знач := n + F(n - 1)
        иначе
          знач := 2 * F(n - 2)
      все
  все
кон
```

Обратим внимание на то, что третий вариант, когда n > 1 и при этом n – нечетно, с записью такого условия в функции рассматривать не нужно (этот вариант войдет в последнюю ветвь со служебным словом **иначе**).

Искомый результат можно получить, так оформив главную часть программы:

```
алг
нач
  вывод F(26)
кон
```

Ответ: 4122.

3.7. Задание 17

Условие

Рассматривается множество целых чисел, принадлежащих числовому отрезку [1016; 7937], которые делятся на 3 и не делятся на 7, 17, 19, 27. Найдите количество таких чисел и максимальное из них. В ответе запишите два целых числа: сначала количество, затем максимальное число.

Для выполнения этого задания можно написать программу или воспользоваться редактором электронных таблиц.

Решение (применительно к программе)

Данная задача – это задача типа 1, рассмотренная в разделе 2.7. Как отмечалось в указанном разделе, такой вариант программы:

```
алг
нач цел n, кол, макс
  кол := 0
  макс := 0
  нц для n от 1016 до 7937
    если mod(n, 3) = 0 и mod(n, 7) <> 0 и mod(n, 17) <> 0 и
         mod(n, 19) <> 0 и mod(n, 27) <> 0
      то
        кол := кол + 1
        если n > макс
          то
            макс := n
        все
    все
  кц
  вывод кол, " ", n
кон
```

можно не использовать. Ее можно оформить короче:

```
алг
нач цел n, кол, макс
  кол := 0
  нц для n от 1016 до 7937
    если mod(n, 3) = 0 и mod(n, 7) <> 0 и mod(n, 17) <> 0 и
         mod(n, 19) <> 0 и mod(n, 27) <> 0
      то
        кол := кол + 1
        макс := n
      все
  кц
  вывод кол, " ", макс
кон
```

Язык Паскаль

```
Var n, kol, max: longint;
BEGIN
  kol := 0;
  for n := 1016 to 7937 do
    if (n mod 3 = 0) and (n mod 7 > 0) and (n mod 17 > 0)
        and (n mod 19 > 0) and (n mod 27 > 0) then
      begin
        kol := kol + 1;
        max := n
      end;
  write(kol, ' ', max)
END.
```

Язык Python

```
kol = 0
for n in range(1016, 7938):
  if n % 3 == 0 and n % 7 > 0 and n % 17 > 0 and n % 19 > 0
      and n % 27 > 0:
    kol = kol + 1
    max = n
print(kol, max)
```

Ответы: 1568 7935.

3.8. Задание 18 (задание выполняется с использованием прилагаемых файлов)

Условие

Квадрат разлинован на N×N клеток (1 < N < 17). Исполнитель Робот может перемещаться по клеткам, выполняя за одно перемещение одну из двух команд: вправо или вниз. По команде вправо Робот перемещается в соседнюю правую клетку, по команде вниз – в соседнюю нижнюю. При попытке выхода за границу квадрата Робот разрушается. Перед каждым запуском Робота в каждой клетке квадрата лежит монета достоинством от 1 до 100. Посетив клетку, Робот забирает монету с собой; это также относится к начальной и конечной клетке маршрута Робота.

Определите максимальную и минимальную денежную сумму, которую может собрать Робот, пройдя из левой верхней клетки

в правую нижнюю. В ответе укажите два числа – сначала максимальную сумму, затем минимальную.

Решение

В [1] исходные данные представляют собой электронную таблицу размером 10×10, каждая ячейка которой соответствует клетке квадрата, в ячейках которого записаны достоинства монет (см. рис. 3.8.1).

	A	B	C	D	E	F	G	H	I	J
1	51	21	93	48	45	100	67	39	18	29
2	57	43	97	51	92	10	93	32	19	58
3	63	16	31	16	78	88	90	72	37	67
4	10	57	64	25	96	50	81	65	91	69
5	99	43	95	7	40	76	18	34	5	65
6	35	19	71	77	64	38	62	56	10	2
7	100	57	27	26	51	33	100	11	53	1
8	11	79	49	46	37	69	80	31	25	39
9	22	71	20	23	11	12	39	16	64	34
10	4	25	87	84	30	48	77	13	40	33

Рис. 3.8.1

Методика выполнения задания описана в разделе 2.8. Если расчеты по нахождению максимального значения провести в диапазоне ячеек A12:J21 (рис. 3.8.2), то ключевые формулы будут такими, как на рис. 3.8.2, а результаты расчетов – как на рис. 3.8.3.

	A	B	C	D	E	F	G	H	I	J
...										
12										
13										
...										
20									=ЕСЛИ(J20>I21;I9+J20;I9+I21)	=J21+J9
21									=J21+I10	=J10

Рис. 3.8.2

	A	B	C	D	E	F	G	H	I	J
...										
12	1204	1153	1132	990	938	893	793	542	415	397
13	1139	1082	1039	942	891	736	726	503	390	368
14	1004	926	884	815	799	721	633	471	371	310
15	941	910	853	729	704	608	543	399	334	243
16	931	832	789	630	598	558	462	316	230	174
17	779	713	694	623	546	482	444	282	225	109
18	744	644	519	492	466	415	382	226	215	107
19	598	587	483	434	388	351	282	193	162	106
20	530	508	432	348	252	223	202	153	137	67
21	441	437	412	325	241	211	163	86	73	33

Рис. 3.8.3

Ответ (максимальное значение): 1204.

Для определения минимального значения можно использовать тот же диапазон ячеек, изменив формулу в ячейке I20 (см. рис. 3.8.4) и скопировав ее в остальные ячейки диапазона A12:I20:

	A	B	C	D	E	F	G	H	I	J
12										
13										
...										
20									=ЕСЛИ(J20<I21;I9+J20;I9+I21)	=J21+J9
21									=J21+I10	=J10

Рис. 3.8.4

Ответ (минимальное значение): 502.

3.9. Задание 22

Условие

Ниже на четырех языках программирования записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: L и M. Укажите наибольшее число x, при вводе которого алгоритм печатает сначала 4, а потом 5.

Алгоритмический язык

```
алг
нач цел x, L, M, Q
  ввод x
  Q := 9
  L := 0
  нц пока x >= Q
    L := L + 1
    x := x - Q
  кц
  M := x
  если M < L
    то
      M := L
      L := x
  все
  вывод L, нс, M
кон
```

Паскаль

```
var x, L, M, Q: integer;
BEGIN
  readln(x);
  Q := 9;
  L := 0;
  while x >= Q do
    begin
      L := L + 1;
      x := x - Q
    end;
  M := x;
  if M < L then
    begin
      M := L;
      L := x
    end;
  writeln(L);
  writeln(M);
END.
```

Python

```
x = int(input())
Q = 9
L = 0
while x >= Q:
  L = L + 1
  x = x - Q
M = x
if M < L:
  M = L
  L = x
print(L)
print(M)
```

C++

```
#include <iostream>
using namespace std;
int main()
{
int x, L, M, Q;
cin >> x;
Q = 9;
L = 0;
while (x >= Q){
L = L + 1;
x = x - Q;
}
M = x;
if (M < L){
M = L;
L = x;
}
cout <<L <<endl <<M <<endl;
return 0;
}
```

Решение

Анализ алгоритма показывает, что во фрагменте с оператором цикла с условием определяются две величины:

1) L – целая часть от деления заданного числа X на 9 (назовем ее «целая часть»);
2) M – остаток от деления X на 9 (остаток).

Действия в следующих фрагмента алгоритма опишем с использованием указанных величин:

`M := x`	Величине M присваивается значение остатка
если M < L **то** M := L L := X **все**	**если** остаток < целой части, **то** величине M присваивается значение целой части, величине L присваивается значение остатка, **иначе** | остаток >= целой части, | величины M и L не меняются, **все**
вывод L, **нс**, M	Выводятся значения L и M

Можно сказать, что вывод значений происходит в зависимости от X по правилу:

если остаток < целой части,
то
выводятся остаток и целая часть,
иначе
выводятся целая часть и остаток,
все.

Нас интересуют значения X, для которых целая часть частного от деления на 9 равна 4, а остаток – 5 либо наоборот.

Такими числами являются 41 и 49. Для 41 имеем M = 4, L = 5, для 49 – M = 5, L = 4. Это значит, что подходит число 49.

Ответ: 49.

3.10. Задание 23

Условие

Исполнитель преобразует число на экране.

У исполнителя есть две команды, которым присвоены номера:

```
1 Прибавить 1
2 Умножить на 2
```

Первая команда увеличивает число на экране на 1, вторая умножает его на 2.

Программа для исполнителя – это последовательность команд.

Сколько существует программ, для которых при исходном числе 1 результатом является число 20, и при этом траектория вычислений содержит число 10?

Траектория вычислений программы – это последовательность результатов выполнения всех команд программы. Например, для программы 121 при исходном числе 7 траектория будет состоять из чисел 8, 16, 17.

Решение

Как указывалось в разделе 2.10, надо разбить задачу на две:

1) задачу *А* – определение количества программ для получения числа 10 при исходном числе 1;
2) задачу *Б* – определение количества программ для получения числа 20 при исходном числе 10.

Тогда искомое в основной задаче значение будет равно произведению двух найденных количеств.

Таблица для решения задачи *А* будет следующей:

Число	Может быть получено через...	Общее количество вариантов
2		2
3		2
4	2 и 3	2 + 2 = 4
5	4	4
6	3 и 5	2 + 4 = 6
7	6	6
8	4 и 7	4 + 6 =10
9	8	10
10	5 и 9	4 + 10 =14

Итак, ответом к задаче *А* является 14.

Задача *Б* (определение количества программ для получения числа 20 при исходном числе 10) решается проще. Так как 20 – четное число, то для него искомый результат может быть получен через 10 и 19. Для чисел 19, 18... 11 также возможен только один результат, значит, общее число вариантов программы для задачи *Б* – 2, а общий ответ ко всему заданию 14 × 2 = 28.

Обратим внимание на то, что для четных чисел 18, 16... 12 их «половины» рассматривать не следует.

Общий ответ: 14 × 2 = 28.

3.11. Задание 24 (задание выполняется с использованием прилагаемых файлов)

Условие

Текстовый файл состоит не более чем из 10^6 символов X, Y и Z. Определите максимальное количество идущих подряд символов, среди которых каждые два соседних различны.

Для выполнения этого задания следует написать программу.

Решение

Внимательный читатель, конечно, обнаружил, что данная задача аналогична задаче, рассмотренной в п. 2.12.3, с той разницей, что в данном случае учитывается подстрока, в которой каждые два соседних символа различны, а не равны.

Еще одно отличие в том, что обрабатываемые символы читаются из файла, в котором может быть большое количество символов. Это означает, что в программах на школьном алгоритмическом языке и на языке Паскаль нельзя прочитать весь файл и присвоить его содержимое переменной величине строкового типа, после чего обработать эту величину, как это делалось во всех рассмотренных в разделе 2.12 задачах[1].

Покажем, как может быть решена указанная проблема в программе на языке Паскаль. В этом языке предусмотрена возможность считывания не всей строки целиком, а по одному символу. Для этого надо описать переменную типа `char`, например

[1] В программе на языке Python это сделать можно, поскольку в системах программирования, использующих этот язык, значения переменных величин практически не ограничены.

с именем ocherednoi_simvol. И вместо процедуры Readln (см. раздел 2.11) использовать в программе процедуру Read:

```
Read(f, ocherednoi_simvol);
```

В результате произойдет чтение первого символа файла, и его значение будет присвоено переменной ocherednoi_simvol. Как показано в п. 2.12.3, это значение следует учесть отдельно.

Для чтения следующего символа также используется процедура Read и т. д. Сколько раз следует повторить выполнение этой процедуры? Если бы количество символов в файле было известно, то можно было бы применить в программе оператор цикла с параметром. Но в данном случае такое количество неизвестно. Можно учесть отмеченную в разделе 1.5 особенность, заключающуюся в том, что в конце файла имеется специальная метка конца файла. Это позволяет использовать в программе оператор цикла с условием вида **not** Eof(f):

```
Var f: text; ocherednoi_simvol: char;
BEGIN
  Assign(f, '24.txt');
  Reset(f);
  { Читаем первый символ файла }
  Read (f, ocherednoi_simvol);
    tek_dlina := 1
    max_dlina := 1
  { Остальные символы файла }
  while not Eof(f) do { Пока не достигнут конец файла }
    begin
      { Читаем очередной символ }
      Read (f, ocherednoi_simvol);
      { Обрабатываем его }
      …
```

Еще одна особенность, связанная с чтением одного символа файла, – необходимость сравнения очередного символа с предыдущим. Это значит, что предыдущий символ следует хранить.

В остальном программа решения обсуждаемой задачи аналогична рассмотренным в п. 2.12.3. Обратим внимание на то, что первый символ строки учитывается отдельно.

Приведем программу на школьном алгоритмическом языке с условным учетом чтения файла по одному символу:

```
алг
нач сим очередной_символ, пред_символ, цел макс_длина, тек_длина
  | Открытие файла на чтение
  …
  | Читаем первый символ файла
  очередной_символ := …
  | и учитываем его
  тек_длина := 1
  макс_длина := 0
  | Запоминаем его в качестве предыдущего для следующего символа
  пред_символ := очередной_символ
  | Рассматриваем остальные символы
  нц пока не конец_файла
    | Читаем очередной символ файла
    очередной_символ := …
    если очередной_символ <> пред_символ
      то
        тек_длина := тек_длина + 1
        | Запоминаем его в качестве предыдущего для следующего символа
        пред_символ := очередной_символ
      иначе | Очередной символ равен предыдущему
        если тек_длина > макс_длина
          то
            макс_длина := тек_длина
        все
        тек_длина := 1
        | Запоминаем очередной символ
        | в качестве предыдущего для следующего символа
        пред_символ := очередной_символ
    все
  кц
  | Проверяем последнюю подстроку
  если тек_длина > макс_длина
    то
      макс_длина := тек_длина
  все
  | Закрытие файла
  …
  вывод нс, макс_длина
кон
```

Язык Паскаль

```
Var stroka: string; tek_dlina, max_dlina: longint;
    ocherednoi_simvol, pred_simvol: char;
    f: text;
BEGIN
  Assign(f, '24.txt');
  Reset(f);
  Read(f, ocherednoi_simvol);
   tek_dlina := 1;
   max_dlina := 0;
   pred_simvol := ocherednoi_simvol
   while not eof() do
    begin
     Read(f, ocherednoi_simvol);
      if ocherednoi_simvol <> pred_simvol then
        begin
          tek_dlina := tek_dlina + 1;
          pred_simvol := ocherednoi_simvol
        end
      else
        begin
          if tek_dlina > max_dlina then
             max_dlina := tek_dlina;
          tek_dlina := 1;
          pred_simvol := ocherednoi_simvol
        end;
    end;
    if tek_dlina > max_dlina then
      max_dlina := tek_dlina;
    writeln(max_dlina);
    Close(f)
END.
```

Язык Python

```
f = open("24.txt", "r")
# Чтение всего файла и запись значения в переменную stroka
stroka = f.readln()
pred_simvol = stroka[0]
tek_dlina = 1
max_dlina = 0
for nom in range(1, len(stroka)):
```

```
  if stroka[nom] != pred_simvol:
    tek_dlina = tek_dlina + 1
  else:
    if tek_dlina > max_dlina:
      max_dlina = tek_dlina
    pred_simvol = stroka[nom]
    tek_dlina = 1
if tek_dlina > max_dlina:
  max_dlina = tek_dlina
print(max_dlina)
f.close
```

Вариант программы, названный в разделе 2.12 компактным [16]:

```
алг
нач сим очередной_символ, пред_символ, цел макс_длина, тек_длина, i
  | Открытие файла на чтение
  …
  | Читаем первый символ файла
  очередной_символ := …
  | и учитываем его
  тек_длина := 1
  макс_длина := 0
  | Запоминаем его в качестве предыдущего для следующего символа
  пред_символ := очередной_символ
  пока не конец_файла
    | Читаем очередной символ файла
    очередной_символ := …
    если очередной_символ <> пред_символ
      то
        тек_длина := тек_длина + 1
        если тек_длина > макс_длина
          то
            макс_длина := тек_длина
        все
        пред_символ := очередной_символ
      иначе
        тек_длина := 1
        пред_символ := очередной_символ
    все
  кц
  | Закрытие файла
  …
  вывод нс, макс_длина
кон
```

Язык Паскаль

```
Var stroka: string; tek_dlina, max_dlina: longint;
    ocherednoi_simvol, pred_simvol: char;
    f: text;
BEGIN
  Assign(f, '24.txt');
  Reset(f);
  Read(f, ocherednoi_simvol);
  tek_dlina := 1;
  max_dlina := 0;
  pred_simvol := ocherednoi_simvol;
  while not eof() do
     begin
      Read(f, ocherednoi_simvol);
       if ocherednoi_simvol <> pred_simvol then
         begin
           tek_dlina := tek_dlina + 1;
           if tek_dlina > max_dlina then
              max_dlina := tek_dlina;
           pred_simvol := ocherednoi_simvol
         end
       else
         begin
           pred_simvol := ocherednoi_simvol;
           tek_dlina := 1
         end
     end;
   writeln(max_dlina);
   Close(f)
END.
```

Язык Python

```
f = open("24.txt", "r")
stroka = f.readln()
pred_simvol = stroka[0]
tek_dlina = 1
max_dlina = 0
for nom in range(1, len(stroka)):
  if stroka[nom] != pred_simvol:
    tek_dlina = tek_dlina + 1
    if tek_dlina > max_dlina:
```

```
      max_dlina = tek_dlina
    pred_simvol = stroka[nom]
  else:
    pred_simvol = stroka[nom]
    tek_dlina = 1
print(max_dlina)
f.close
```

3.12. Задание 25

Условие

Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку [174457; 174505], числа, имеющие ровно два различных натуральных делителя, не считая единицы и самого числа. Для каждого найденного числа запишите эти два делителя в таблицу на экране с новой строки в порядке возрастания произведения этих двух делителей. Делители в строке таблицы также должны следовать в порядке возрастания.

Например, в диапазоне [5; 9] ровно два целых различных натуральных делителя имеют числа 6 и 8, поэтому для этого диапазона таблица на экране должна содержать следующие значения:

```
2 3
2 4.
```

Решение

Задача решается аналогично рассмотренным в разделе 2.13. Особенность – при подсчете количества делителей числа 1 и n учитываться не должны.

Требование о порядке возрастания произведения делителей будет учтено автоматически, если рассматривать числа n от меньшего к большему, а требование о порядке следования делителей – также автоматически, если значения возм_дел рассматривать от меньшего к большему (как это делалось во всех вариантах программ в п. 1.1.9 и 1.1.10).

Программа:

```
алг
нач цел n, возм_дел, кол_дел, i, цел таб делители[1:10000]
  нц для n от 174457 до 174505
    кол_дел := 0
```

```
    нц для возм_дел от 2 до div(n, 2)
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
          делители[кол_дел] := возм_дел
      все
    кц
    если кол_дел = 2
      то
        вывод нс
        нц для i от 1 до 2  | Можно выводить 2 делителя
                            | без использования оператора цикла
          вывод делители[i], " "
        кц
    все
  кц
кон
```

Язык Паскаль

```
Var n, vozm_del, kol_del: longint; i: integer;
    deliteli: array [1..10000] of longint;
BEGIN
  for n := 174457 to 174505  do
    begin
      kol_del := 0;
      for vozm_del := 2 to n div 2 do
        if n mod  vozm_del = 0 then
          begin
            kol_del := kol_del + 1;
            deliteli[kol_del] := vozm_del
          end;
      if kol_del = 2 then
         begin
           writeln;
           for i := 1 to 2 do
             write(deliteli[i], ' ');
         end
  end
END.
```

Язык Python

```
for n in range(174457, 174506):
  deliteli = []
  kol_del = 0
  for vozm_del in range(2, n//2 + 1):
    if n % vozm_del == 0:
      kol_del = kol_del + 1
      deliteli.append(vozm_del)
  if kol_del == 2:
    print(deliteli[0], deliteli[1])
```

Как указывалось в разделе 2.13, переменную kol_del можно не использовать:

```
for n in range(174457, 174506):
  deliteli = []
  for vozm_del in range(2, n//2 + 1):
    if n % vozm_del == 0:
      deliteli.append(vozm_del)
  if len(deliteli) == 2:
    print(deliteli[0], deliteli[1])
```

Вариант программы с оптимизацией по размеру массива:

```
алг
нач цел n, возм_дел, кол_дел, i, цел таб делители[1:2]
  нц для n от 174457 до 174505
    кол_дел := 0
    нц для возм_дел от 2 до div(n, 2)
      если mod(n, возм_дел) = 0
        то
          кол_дел := кол_дел + 1
          если кол_дел <= 2
            то
              делители[кол_дел] := возм_дел
            иначе
              выход
          все

      все
    кц
    если кол_дел = 2
      то
        вывод нс
```

```
        нц для i от 1 до 2
          вывод делители[i], " "
        кц
    все
  кц
кон
```

Язык Паскаль

```
Var n, vozm_del, kol_del: longint; i: integer;
    deliteli: array [1..2] of longint;
BEGIN
  for n := 174457 to 174505  do
    begin
      kol_del := 0;
      for vozm_del := 2 to n div 2 do
        if n mod  vozm_del = 0 then
          begin
            kol_del := kol_del + 1;
            if kol_del <= 2 then
               deliteli[kol_del] := vozm_del
            else break;
          end;
      if kol_del = 2 then
         begin
          writeln;
          for i := 1 to 2 do
            write(deliteli[i], ' ');
         end
  end
END.
```

Язык Python

```
for n in range(174457, 174506):
  deliteli = []
  for vozm_del in range(2, n//2 + 1):
    if n % vozm_del == 0:
      if len(deliteli) <= 2:
        deliteli.append(vozm_del)
  if len(deliteli) == 2:
    print(deliteli[0], deliteli[1])
```

Ответ:

```
3 58153
7 24923
59 2957
13 13421
149 1171
5 34897
211 827
2 87251
```

3.13. Задание 26 (задание выполняется с использованием прилагаемых файлов)

Условие

Системный администратор раз в неделю создает архив пользовательских файлов. Однако объем диска, куда он помещает архив, может быть меньше, чем суммарный объем архивируемых файлов.

Известно, какой объем занимает файл каждого пользователя.

По заданной информации об объеме файлов пользователей и свободном объеме на архивном диске определите максимальное число пользователей, чьи файлы можно сохранить в архиве, а также максимальный размер имеющегося файла, который может быть сохранен в архиве, при условии, что сохранены файлы максимально возможного числа пользователей.

Входные данные

В первой строке входного файла находятся два числа: `S` – размер свободного места на диске (натуральное число, не превышающее 10 000) и `N` – количество пользователей (натуральное число, не превышающее 1000). В следующих `N` строках находятся значения объемов файлов каждого пользователя (все числа натуральные, не превышающие 100), каждое в отдельной строке.

Запишите в ответе два числа: сначала наибольшее число пользователей, чьи файлы могут быть помещены в архив, затем максимальный размер имеющегося файла, который может быть сохранен в архиве, при условии, что сохранены файлы максимально возможного числа пользователей.

Пример входного файла:

```
100 4
80
30
```

```
50
40
```

При таких исходных данных можно сохранить файлы максимум двух пользователей. Возможные объемы этих двух файлов 30 и 40, 30 и 50 или 40 и 50. Наибольший объем файла из перечисленных пар – 50, поэтому ответ для приведенного примера:

```
2 50
```

Решение

Полный перебор всех возможных вариантов размещения файлов невозможен по следующей причине.

Все варианты такие:

1) файл одного из пользователей (N вариантов);
2) файлы двух любых пользователей;

...

N) файлы всех N пользователей (один вариант).

Но так как N может составлять 999, то рассмотрение всех вариантов в общем случае будет невозможным.

Рациональный вариант решения задачи основан на следующих рассуждениях.

Пусть N = 14, S = 100, а значения размеров файлов пользователей такие:

95 11 91 52 65 57 54 88 87 58 6 10 50 55.

Для размещения максимального количества файлов в первую очередь следует размещать на диске файлы небольших размеров. Отсортировав указанные размеры в порядке неубывания, получим:

6 10 11 50 52 54 55 57 58 65 87 88 91 95.

Далее, суммируя последовательно все начальные (меньшие) размеры, можно найти такой максимальный размер файла, для которого сумма всех размеров не превышает предельное значение 100. Для приведенного примера это четвертое число (50). Значит, на диске можно сохранить файлы максимум четырех пользователей.

Означает ли, что значение 50 является максимальным для найденного максимального количества файлов? Ответ – нет. Это может быть и большее значение, при котором сумма размеров всех файлов не превышает S. Для приведенного примера таким максимальным значением является число 65.

Прежде чем рассказывать о том, как найти окончательное максимальное значение, приведем фрагмент программы на школьном алгоритмическом языке, в котором находится одно из искомых значений – максимальное количество размещаемых файлов. В ней, кроме переменных N и S, использованы следующие основные величины:

- м – массив для хранения заданных размеров файлов пользователей;
- сумма – сумма размеров файлов;
- к – искомое максимальное количество размещаемых на диске файлов;
- номер_прохода, лев, всп – величины, необходимые для сортировки массива методом «пузырьковой сортировки» (методом обмена) – см. приложение 3[1].

Итак, начало программы:

```
алг
нач цел N, S, сумма, к, номер_прохода, лев, всп, i, цел таб м[1 :
1000]
  | Определение значений S и N, записанных во входном файле
  … (см. ниже)
  | Чтение из входного файла значений размеров файлов пользователей
  | и запись их в массив м
  …
  | Сортируем массив м методом обмена
  нц для номер_прохода от 1 до N - 1
    нц для лев от 1 до N - номер_прохода
      если м[лев] > м[лев + 1]
        то | Проводим обмен
          всп := м[лев]
          м[лев] := м[лев + 1]
          м[лев + 1] := всп
      все
    кц
  кц
  | Ищем первое искомое значение, используя оператор цикла с условием
  i := 0 | Условно для проверки первого элемента массива
  сумма := 0
  нц пока сумма + м[i + 1] <= S    | Проверяем следующий элемент
```

[1] Можно применить любой другой способ сортировки.

```
    | Учитываем его
    сумма := сумма + м[i + 1]
    | и переходим к нему
    i := i + 1
  кц
  | Первое искомое значение - максимальное количество файлов
к найдено
  | (оно равно последнему значению i)
  к := i
  | Сумма для найденного количества равна сумма
```

Теперь о том, как определить второе искомое значение. Покажем это на примере данных, приведенных нами выше.

Можно исключить из общей суммы размеров файлов значение 50 и попытаться найти среди остальных (бóльших) размеров максимальное значение, при котором общая сумма размеров не превышает значение S. Это можно сделать с использованием оператора цикла с условием:

```
сумма_без = сумма - м[к]  | Смысл величины сумма_без понятен из ее имени
i := к | Начинаем с найденного до этого к-го элемента
нц пока сумма_без + м[i + 1] <= S  | Подходит следующий элемент
  | Переходим на него
  i := i + 1
кц
```

После этого возможны два варианта:

1) большее значение найдено (если окажется i > к);

2) большее значение не найдено (значение i = к не изменилось).

В обоих случаях искомым максимальным значением будет значение м[i]. Например, для приведенных значений размеров это будет, как уже отмечалось, 65 (i = 10), а для таких размеров файлов:

6 10 11 50 87 88 91 95

– ответом будет все то же значение 50 (i останется равным 4). Такой же ответ будет для следующего случая:

6 10 11 50 50 87 88 91 94 95

(i = 5).

С учетом сказанного завершающая часть программы оформляется следующим образом:

```
  ...
  | Поиск второго искомого значения
  ...
  | Вывод ответов
  вывод к, " ", м[i]
  | Закрытие файла
  ...
кон
```

Обсудим теперь вопрос о чтении входных данных из файла. Согласно условию, в его первой строке записаны два числа (S и N). Их можно получить, прочитав и запомнив всю строку, после чего выделить в ней первое и второе «слово» (см. задачу 2 в п.1.5.4) и преобразовать их в числа (см. дополнение к разделу 1.5).

После этого, зная количество оставшихся строк (оно равно N), можно прочитать каждую и значение в ней, преобразовав его в число, записать в массив м.

Соответствующие программы:

Язык Паскаль

```
Var
N, S, summa, k, nomer_prohoda, lev, vsp, i: integer;
m: array [1..1000] of integer;
  f: text; stroka, slovo1, slovo2: string;
BEGIN
  { Определение значений S и N, записанных во входном файле }
Assign(f, '26.txt');
Reset(f);
{ Читаем первую строку файла
и записываем все ее значение в переменную stroka }
Readln(f, stroka);
{ Выделяем первое слово строки }
  slovo1  := '';
  i := 1;
  while stroka[i] <> ' ' do
    begin
      slovo1 := slovo1 + stroka[i];
      i := i + 1
    end;
  { Преобразовываем первое слово в число }
  S := StrToInt(slovo1};
{ Выделяем второе слово строки }
```

```
i := i + 1;
slovo2 := '';
while i <= length(stroka) do
  begin
    slovo2 := slovo2 + stroka[i];
    i := i + 1
  end;
{ Преобразовываем второе слово в число }
N := StrToInt(slovo2);
{ Чтение из входного файла значений размеров файлов пользователей
  и запись их в массив m }
for i := 1 to N do
  begin
    { Читаем очередную строку }
    Readln(f, stroka);
    { Преобразовываем ее значение в число и записываем в массив }
    m[i] : = StrToInt(stroka)
  end;
{ Сортируем массив m методом обмена }
for nomer_prohoda := 1 to N - 1 do
  for lev := 1 to N - nomer_prohoda do
    if m[lev] > m[lev + 1] then { Проводим обмен }
      begin
        vsp := m[lev];
        m[lev] := m[lev + 1];
        m[lev + 1] := vsp
      end;
{ Ищем первое искомое значение }
i := 0;
summa := 0;
while summa + m[i + 1] <= S do
  begin
    { Учитываем следующий элемент }
    summa := summa + m[i + 1];
    { и переходим к нему }
    i := i + 1;
  end;
  { Первое искомое значение - максимальное количество файлов
  k найдено (оно равно i)}
  k := i;
  { Поиск второго искомого значения }
  summa_bez := summa - m[k];
  i := k  { Начинаем с найденного до этого k-го элемента }
```

```
    while summa_bez + m[i + 1] <= S do
      { Переходим к следующему элементу }
      i := i + 1;
      { Второе искомое значение найдено - оно равно m[i]}
    { Вывод ответов }
    writeln(k, ' ', m[i]);
    { Закрытие файла }
    Close(f)
END.
```

Язык Python

```
m = []
# Определение значений S и N, записанных во входном файле
f = open("26.txt", "r")
# Читаем первую строку файла
# и записываем все ее значение в переменную stroka
stroka = f.readln()
# Выделяем первое слово строки
slovo1 = ""
i = 0
while stroka[i] != " ":
  slovo1 = slovo1 + stroka[i]
  i = i + 1
# Преобразовываем первое слово в число
S = int(slovo1}
# Выделяем второе слово строки}
i = i + 1
slovo2 = ""
while i <= len(stroka) - 1:
  slovo2 = slovo2 + stroka[i]
  i = i + 1
# Преобразовываем второе слово в число
N = int(slovo2)
# Чтение из входного файла значений размеров файлов пользователей
# и запись их в массив m (здесь можно использовать метод append)
for i in range(N):
  # Читаем очередную строку
  stroka = f.readln()
  # Преобразовываем ее значение в число и записываем в массив
  m.append(int(stroka))
# Сортируем массив m методом обмена
for nomer_prohoda in range(1, N - 1):
  for lev in range (0, (N - 1) - nomer_prohoda):
```

```
    if m[lev] > m[lev + 1]:  # Проводим обмен
      vsp = m[lev]
      m[lev] = m[lev + 1]
      m[lev + 1] = vsp
# Ищем первое искомое значение
i = -1 # Условно для проверки элемента массива с индексом 0
summa = 0
while summa + m[i + 1] <= S:
  # Учитываем следующий элемент
  summa = summa + m[i + 1]
  # и переходим к нему
  i = i + 1
# Первое искомое значение - максимальное количество файлов k найдено
# (оно равно i)
k = i
# Поиск второго искомого значения
summa_bez = summa - m[k]
i = k  # Начинаем с найденного до этого k-го элемента
while summa_bez + m[i + 1] <= S:
  # Переходим к следующему элементу
  i = i + 1
# Второе искомое значение найдено - оно равно m[i]
# Вывод ответов
print(k, " ", m[i])
# Закрытие файла
f.close()
```

3.14. Задание 27 (задание выполняется с использованием прилагаемых файлов)

Условие

Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Входные данные

Даны два входных файла (файл *A* и файл *B*), каждый из которых содержит в первой строке количество пар N ($1 \leqslant N \leqslant 100000$). Каждая из следующих N строк содержит два натуральных числа, не превышающих 10 000.

Пример организации исходных данных во входном файле:

```
6
1 3
5 12
6 9
5 4
3 3
1 1
```

Для указанных входных данных значением искомой суммы должно быть число 32.

В ответе укажите два числа: сначала значение искомой суммы для файла *A*, затем для файла *B*.

Предупреждение: для обработки файла *B* не следует использовать переборный алгоритм, вычисляющий сумму для всех возможных вариантов, поскольку написанная по такому алгоритму программа будет выполняться слишком долго[1].

Прежде чем представлять программу решения задачи, заметим, что в ней, в отличие от программы выполнения задания 26, в первой строке файла записано одно значение, а в остальных – два. Методика использования всех значений в программе описана ниже.

Еще одна особенность. В задании требуется определить нужные значения для двух файлов, при этом уточняется, что для файла *B* использовать переборный алгоритм нельзя. Можно предположить, что для файла *A* это сделать можно[2].

Опишем такой вариант решения.

Применительно к файлу A задание можно выполнить, как говорится, в лоб: сохранить в двумерном массиве из *некоторого* количества строк и двух столбцов все исходные данные, перебрать все возможные способы выбора одного элемента из каждой строки и найти максимальную сумму, соответствующую условиям задачи. Сформировать все возможные сочетания одного элемента из каждой пары можно так, как решалась задача 1.6.5.

Количество строк записано в первой строке файла. Это значит, что можно сначала составить программу, в которой читается и выводится на экран первая строка файла, т. е. станет известным количество строк в массиве, что даст возможность в новом ва-

[1] В [1] общее количество пар чисел для файла *B* – 60 000.

[2] В [1] общее количество пар чисел для файла *A* – 20.

рианте программы на языках Паскаль и Python[1] описать массив соответствующего размера.

В приведенной ниже программе решения использованы следующие величины:

- a – двумерный массив с исходными данными;
- сум – сумма значений всех элементов отдельного сочетания;
- сум_макс – искомое максимальное значение;
- i – индексы строк массива a;
- i1, i2, …, i20 – индексы элементов каждой строки.

Программа

```
алг
нач цел таб a[1:20, 1:2], цел N, сум, сум_макс
    цел i1, i2, …, i20
  | Ввод исходных данных
  | Чтение первой строки файла и определение значения N
  … (см. ниже)
  нц для i от 1 до N
    | Чтение остальных строк и определение значений a[i, 1] и a[i, 2]
    … (см. ниже)
  кц
  | Определение максимальной суммы
  сум_макс := 0  | Начальное значение
  | Формирование всех сочетаний по одному элементу из каждой строки
  нц для i1 от 1 до 2
    нц для i2 от 1 до 2
      …
        нц для i20 от 1 до 2
          | Расчет суммы элементов сочетания
          сум := a[1, i1] + a[2, i2] + … + a[20, i20]
          | Проверка и сравнение
          если mod(сум, 3) <> 0 и сум > сум_макс
            то
              сум_макс := сум
          все
        кц
      …
    кц
  кц
  | Закрытие файла
  …
```

[1] В программе на языке Python можно сначала объявить «пустой» список, после чего заполнять его, используя метод append.

```
  | Вывод ответа
  вывод нс, сум_макс
кон
```

Прежде чем представлять аналогичные программы на других языках программирования, обсудим методику ввода в программу исходных данных из файла.

Количество пар чисел определяется путем преобразования в число значения в первой строке исходного текстового файла (см. дополнение к разделу 1.5).

Значения пар чисел, записанные в каждой из остальных N строк, можно выделить так, как в программе выполнения задания 26 определялись два числа в первой строке (см. также задачу 2 в п. 1.5.4).

Итак, программы.

Язык Паскаль

```
var a: array[1..20, 1..2] of longint; { Указан размер для файла A в
[1] }
    i1, i2, …, i20: integer;
    i, sum, sum_max: longint;
    f: text; stroka: string;
BEGIN
  Assign(f, '27.txt');
  Reset(f);
    { Читаем первую строку файла
  и записываем ее значение в переменную stroka }
  Readln(f, stroka);
  { Преобразовываем значение в число }
    N := StrToInt(stroka);
    { Чтение остальных строк файла
      и определение значений a[i, 1] и a[i, 2] }
    for i1 := 1 to 20 do
      begin
        { Читаем очередную строку файла
        и записываем все ее значение в переменную stroka }
        Readln(f, stroka);
        { Выделяем первое слово строки }
        slovo1  := '';
        i := 1;
        while stroka[i] <> ' ' do
```

```
          begin
            slovo1 := slovo1 + stroka[i];
            i := i + 1
          end;
        { Преобразовываем первое слово в число
          и записываем его в 1-й элемент
          каждой i-й строки массива a }
        a[i, 1] := StrToInt(slovo1);
      { Выделяем второе слово строки }
        i := i + 1;
        slovo2 := '';
        while i <= length(stroka) do
          begin
            slovo2 := slovo2 + stroka[i];
            i := i + 1
          end;
        { Преобразовываем второе слово в число
          и записываем его во 2-й элемент
          каждой i-й строки массива a}
          a[i, 2] := StrToInt(slovo2);
      end;
    { Определение максимальной суммы }
    sum_max := 0;
    { Формирование всех сочетаний по одному элементу из каждой строки }
  for i1 := 1 to 2 do
    for i2 := 1 to 2 do
      …
       for i20 := 1 to 2 do
         begin
           sum := a[1, i1] + a[2, i2] + … +a[20, i20];
           if (sum mod 3 <> 0) and (sum > sum_max)
               then sum_max := sum
         end;
    { Закрытие файла }
    Close(f);
    { Вывод ответа }
    writeln(sum_max)
END.
```

Язык Python

```
f = open("27.txt", "r")
# Чтение первой строки файла и определение значения N
```

```
stroka = f.readln()
# Преобразовываем ее значение в число
N = int(stroka)
# Объявление и заполнение массива a нулями
a = [ [0, 0] for i in range(20) ]  # с помощью генератора списка [17]
# Указан размер для файла A в [1]
# Чтение остальных строк файла и определение значений a[i, 0] и a[i, 1]
for i in range(20):
  # Читаем очередную строку файла
  # и записываем все ее значение в переменную stroka
  stroka = f.readln()
  # Выделяем первое слово строки
  slovo1 = ""
  i = 0
  while stroka[i] != " ":
    slovo1 = slovo1 + stroka[i]
    i = i + 1
  # Преобразовываем первое слово в число
  # и записываем его в первый элемент каждой строки массива a
  a[i, 0] = int(slovo1)
  # Выделяем второе слово строки
  i = i + 1
  slovo2 = ""
  while i <= len(stroka) - 1:
    slovo2 = slovo2 + stroka[i]
    i = i + 1
  # Преобразовываем второе слово в число
  # и записываем его во второй элемент каждой строки массива a
  a[i, 1] = int(slovo2)
# Определение максимальной суммы
sum_max = 0
# Формирование всех сочетаний по одному элементу из каждой строки
for i0 in range(0, 1):
  for i1 in range(0, 1):
    …
      for i19 in range(0, 1):
        sum = a[0, i0] + a[1, i1] + … + a[19, i19]
          if sum % 3 <> 0 and sum > sum_max:
            sum_max = sum
# Закрытие файла
f.close
# Вывод ответа
print(sum)
```

Конечно, программы получаются достаточно объемными, но они позволяют просто получить решение для одного из файлов (за каждое правильное решение дается 1 балл).

Однако приведенный вариант не может быть применен к файлу *B*. Обсудим вариант программы, с помощью которого можно получить ответ и для файла *B*.

Чтобы получить максимально возможную сумму, будем брать из каждой пары самое большое число. Если полученная при этом сумма будет делиться на 3, ее необходимо уменьшить. Для этого достаточно в одной из пар, где числа имеют разные остатки при делении на 3, заменить ранее выбранное число на другое число из той же пары. При этом разница между числами в паре должна быть минимально возможной.

Основные величины, использованные в соответствующей программе:

- N – количество пар чисел;
- a_макс – наибольшее возможное число в исходных данных (константа, равная 10000);
- a1 – первое число в паре чисел;
- a2 – второе число в паре чисел;
- макс – максимум в паре;
- мин – минимум в паре;
- сумма – сумма выбранных чисел;
- мин_разн – минимальная разность макс – мин, не кратная 3.

Программа

```
алг
нач цел N, a1, a2, a_макс, макс, мин, сумма, мин_разн, i
  | Чтение из первой строки файла значения количества пар чисел
  | и запись его в переменную N
  ...
  a_макс := 10000
  | Начальное значение величины мин_разн
  мин_разн := a_макс + 1
  | Определяем сумму максимальных значений в парах
  сумма := 0
  | Чтение из остальных строк файла значений пар чисел
  нц для i от 1 до N
    | и запись их в переменные a1 и a2
    ...
```

```
    если a1 > a2
      то
        макс := a1; мин := a2
      иначе
        макс := a2; мин := a1
    все
    сумма := сумма + макс
    | При необходимости уточняем значение мин_разн
    если mod(макс - мин, 3) > 0 и макс - мин < мин_разн
      то
        мин_разн := макс - мин
    все
  кц
  | Проверяем сумму
  если mod(сумма, 3) = 0
    то | Уточняем ее в зависимости от значения мин_разн
      если мин_разн > a_макс
        то
          сумма := 0
        иначе
          сумма := сумма - мин_разн
      все
  все
  | Закрытие файла
  ...
  | Вывод ответа
  вывод нс, сумма
кон
```

Язык Паскаль

```
const
  a_max = 10000; {наибольшее возможное число в исходных данных}
var
  N, a1, a2, max, min, summa, min_razn, i: longint;
  f: text; slovo1, slovo2: string;
BEGIN
  { Чтение из первой строки файла значения количества пар чисел
    и запись его в переменную N }
  Assign(f, "27.txt");
  Reset(f);
  { Читаем первую строку файла
  и записываем ее значение в переменную stroka }
  Readln(f, stroka);
```

```
  { Преобразовываем значение stroka в число }
  N : = StrToInt(stroka);
  { Определяем сумму максимальных значений в парах }
  summa := 0;
  min_razn := a_max + 1;
  { Чтение из остальных строк файла значений пар чисел
       и запись их в переменные a1 и a2 }
for i := 1 to N do
  begin
    { Читаем очередную строку }
    Readln(f, stroka);
    { Выделяем в ней первое слово }
  slovo1  := '';
  i := 1;
  while stroka[i] <> ' ' do
    begin
      slovo1 := slovo1 + stroka[i];
      i := i + 1
    end;
  { Преобразовываем первое слово в число }
  a1 := StrToInt(slovo1);
  { Выделяем второе слово строки}
  i := i + 1;
  slovo2 := '';
  while i <= length(stroka) do
    begin
      slovo2 := slovo2 + stroka[i];
      i := i + 1
    end;
  { Преобразовываем второе слово в число }
  a2 := StrToInt(slovo2);
  if a1 > a2 then
    begin
      max := a1;
      min := a2
    end
  else
    begin
      max := a2;
      min := a1
    end;
  summa := summa + max;
      { При необходимости уточняем значение min_razn }
  if ((max - min) mod 3 > 0) and (max - min < min_razn)
```

```
    then min_razn := max - min
  end;
{ Проверяем сумму }
if summa mod 3 = 0 then
  begin
    if min_razn > aMax
      then summa := 0
    else
      summa := summa - min_razn
  end;
  { Закрытие файла }
  …
writeln(summa)
END.
```

Язык Python

```
a_max = 10000
summa = 0
min_razn = a_max + 1
f = open("27.txt", "r")
# Чтение первой строки файла и определение значения N
stroka = f.readln()
N = int(stroka)
# Чтение из остальных строк файла значений пар чисел
# и запись их в переменные a1 и a2
for i in range(1, 20):
  # Читаем очередную строку файла
  # и записываем все ее значение в переменную stroka
  stroka = f.readln()
  # Выделяем первое слово строки
  slovo1 = ""
  i = 0
  while stroka[i] != " ":
    slovo1 = slovo1 + stroka[i]
    i = i + 1
  # Преобразовываем первое слово в число
  # и присваиваем его переменной a1
  a1 = int(slovo1)
  # Выделяем второе слово строки
  i = i + 1
  slovo2 = ""
  while i <= len(stroka) - 1:
    slovo2 = slovo2 + stroka[i]
```

```
    i = i + 1
  # Преобразовываем второе слово в число
  # и присваиваем его переменной a2
  a2 = int(slovo2)
  if a1 > a2:
    max = a1
    min = a2
  else:
    max = a2
    min = a1
  summa = summa + max
  # При необходимости уточняем значение min_razn
  if (max - min) % 3 > 0 and max - min < min_razn:
    min_razn = max - min
# Проверяем сумму
if summa mod 3 == 0:
  if min_razn > aMax:
    summa = 0
  else:
    summa = summa - min_razn
# Закрытие файла
f.close()
# Вывод ответа
print(summa)
```

В заключение напомним, что в задании требуется определить два значения (для файла *A* и для файла *B*). Правильные результаты: 127127 и 399762080.

Приложение 1

Динамическое программирование. Основы

Динамическое программирование[1] – особый метод поиска оптимальных решений, специально приспособленный к так называемым многошаговым, или многоэтапным, операциям [18]. При этом многошаговость либо отражает реальное протекание процесса принятия решений во времени, либо вводится в задачу искусственно за счет расчленения процесса принятия однократного решения на отдельные этапы.

Приведем несколько примеров задач, в которых рассматриваются многошаговая, т. е. представляющая собой последовательность шагов (этапов), операция.

Задача 1. Нужно проложить путь, соединяющий два пункта – *A* и *B* (рис. П1.1), второй из которых лежит к северо-востоку от первого. Прокладка пути состоит из ряда шагов, и на каждом шаге мы можем двигаться либо строго на север, либо строго на восток, т. е. любой путь из *A* в *B* представляет собой ступенчатую ломаную линию, отрезки которой параллельны одной из координатных осей. Затраты на прокладку каждого из таких отрезков известны. Требуется проложить такой путь из *A* в *B*, при котором суммарные затраты минимальны.

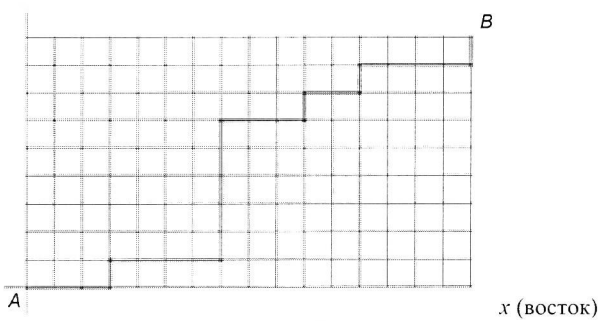


Рис. П1.1

Задача 2. Имеется определенный набор предметов – $П_1$, $П_2$, ..., $П_n$ (каждый в единственном экземпляре); известны их веса – B_1, B_2, ..., B_n и стоимость – C_1, C_2, ..., C_n. Грузоподъемность машины

[1] Термин «программирование» в названии метода происходит от слова «программа» в значении *план действий* и не связан с понятием *компьютерная программа*. Поэтому динамическое программирование иначе называют динамическим планированием. Происхождение же термина «динамический» связано с использованием метода в задачах принятия решений через фиксированные промежутки времени.

равна Q. Спрашивается: какие из предметов нужно взять в машину, чтобы их суммарная стоимость (при суммарном весе $\le Q$) была максимальна?

Задача 3. Имеется какой-то объем денежных средств Q, который должен быть распределен между предприятиями $П_1$, $П_2$, ..., $П_n$. Каждое из предприятий $П_i$ при вложении в него каких-то средств в объеме x приносит доход, зависящий от x, т. е. представляющий собой какую-то функцию $f_i(x)$. Все функции $f_i(x)$ ($i = 1, 2, ..., n$) заданы (разумеется, эти функции – неубывающие). Спрашивается: сколько средств нужно выделить каждому предприятию, чтобы в сумме они дали максимальный доход?

В первой задаче операцией является строительство пути из точки A в точку B. Здесь шаги выделены уже в условии задачи – шагом является прокладывание очередного отрезка пути.

В постановке второй задачи нет упоминания о времени. Но процесс загрузки машины можно представить как состоящий из n шагов, считая за первый шаг принятие решения о том, брать или не брать первый предмет, за второй – то же относительно второго предмета и т. п.

В третьей задаче также можно операцию распределения средств мысленно развернуть во времени в какой-то последовательности и рассматривать решение вопроса о вложении средств в предприятие $П_1$ как первый шаг, в предприятие $П_2$ – как второй и т. д.

Как можно решать подобного рода задачи? Очевидно, что это можно сделать по-разному.

Можно перебрать все возможные варианты решения и выбрать среди них лучший. Например, в первой задаче можно рассмотреть все возможные варианты пути и выбрать тот, на котором затраты минимальны.

А можно выбирать лучший вариант пути шаг за шагом, на каждом этапе расчета оптимизировать только один шаг. Обычно второй способ оптимизации оказывается проще, чем первый, особенно при большом числе шагов[1].

---

[1] При большом числе шагов в ряде случаев решение первым способом вообще не может быть реализовано. В задаче 1, например, общее число возможных вариантов пути равно числу сочетаний из $(N_1 + N_2)$ по N_1, где N_1 – число отрезков от точки A до точки B в восточном направлении, N_2 – в северном. При $N_1 = 10$ и $N_2 = 10$ общее число вариантов равно 184756. При увеличении N_1 и N_2 это число существенно возрастает (в [19] показано, что при $N_1 = 30$ и $N_2 = 30$ компьютер, выполняющий 1 000 000 операций в секунду, рассмотрит все возможные варианты пути за более чем 100 000 лет!).

Такая идея постепенной, пошаговой оптимизации и лежит в основе метода динамического программирования.

На первый взгляд она может показаться довольно тривиальной. В самом деле, чего, казалось бы, проще: если трудно оптимизировать процесс в целом, надо разбить его на ряд шагов и оптимизировать каждый шаг. Это сделать нетрудно – надо выбрать на этом шаге такое управление, чтобы эффективность этого шага была максимальна. Не так ли?

Нет, вовсе не так! Принцип динамического программирования отнюдь не предполагает, что каждый шаг оптимизируется о т д е л ь н о, независимо от других. Напротив, управление на каждом шаге должно выбираться дальновидно, с учетом его последствий в будущем. Что толку, если мы выберем на данном шаге управление, при котором эффективность э т о г о шага максимальна, если он приведет нас к проигрышу на последующих шагах?

А как выбрать оптимальное управление на том или ином шаге? Ведь на каждом шаге (естественно, кроме первого) система может находиться в общем случае в нескольких состояниях (в различных точках местности, на которой сооружается путь из A в B, с различным весом груза, который еще можно взять в машину и т. п.), а в каждом из этих состояний в общем случае возможно несколько вариантов управления (строить очередной участок пути в северном или в восточном направлении, брать или не брать i-й предмет, вкладывать в i-е предприятие один из возможных объемов средств или не вкладывать вообще). Какое же решение является оптимальным? Ведь мы не знаем даже, чем закончился предыдущий шаг.

Так как ответить на этот вопрос непросто, давайте решим другую задачу: определим, какое управление является лучшим для *каждого* из состояний системы на том или ином шаге (иными словами, сделаем разные предположения о том, чем закончился предыдущий шаг). Такое управление назовем условным оптимальным управлением для этого состояния, а выигрыш, который дает это управление, – условным оптимальным выигрышем (условное потому, что оно выбирается исходя из условия, что предыдущий шаг кончился так-то и так-то).

Выбрать такой вариант управления можно, сравнив все возможные управления по следующему критерию: *сумма выигрыша от реализации того или иного управления на данном шаге и условного оптимального выигрыша в состоянии системы, в которое она*

перейдет в результате этого управления (помните о необходимости на каждом шаге принимать дальновидные решения?)[1].

Очевидно, что для использования такого критерия необходимо знать условные оптимальные выигрыши для всех состояний системы на следующем шаге. А из этого в свою очередь следует очень важный вывод: определение условных оптимальных параметров надо начинать с последнего шага, затем найти их для предпоследнего шага и т. д.

Как отмечается в [18], полное понимание основных положений динамического программирования, как правило, возможно только после рассмотрения ряда примеров, поэтому перейдем к решению.

Задача 1

Разделим расстояние от *A* до *B* в восточном направлении, скажем, на 7 частей, а в северном – на 5 (в принципе, дробление может быть сколь угодно мелким). Тогда любой путь из *A* в *B* состоит из 7 + 5 = 12 отрезков, направленных на восток или на север (рис. П1.2). Проставим на каждом из отрезков число, выражающее (в каких-то условных единицах) стоимость прокладки пути по этому отрезку. По требованию условия задачи необходимо выбрать такой путь из *A* в *B*, при котором сумма чисел, стоящих на отрезках, минимальна.

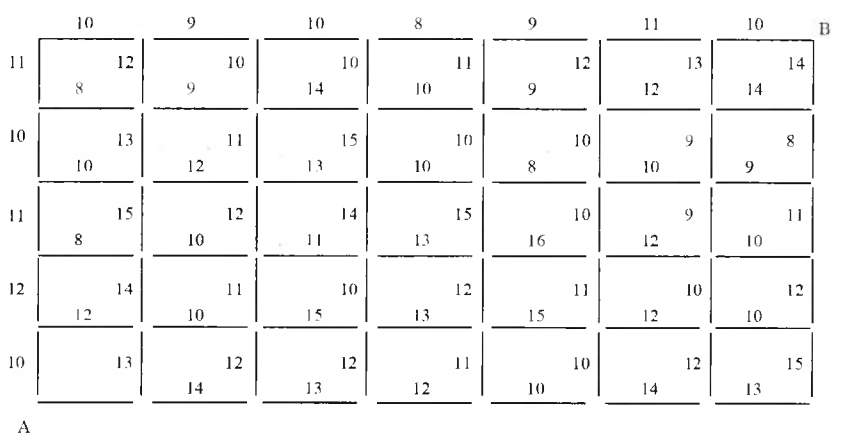


Рис. П1.2

[1] Не вызывает сомнения тот факт, что именно по такому критерию пришлось бы выбирать оптимальный вариант управления в случае, когда какое-то состояние было бы *исходным*.

Будем рассматривать сооружаемый путь как управляемую систему, перемещающуюся под влиянием управления (строительства очередного участка) из начального состояния A в конечное B. Состояние системы перед началом каждого шага будем характеризовать двумя координатами: восточной x и северной y, обе целочисленные ($0 \le x \le 7$, $0 \le y \le 5$). Для каждого из состояний системы (узловой точки прямоугольной сетки на рис. П1.2) мы должны найти условное оптимальное управление: идти нам на север или на восток? Выбирается это управление так, чтобы стоимость всех оставшихся до конца шагов (включая данный) была минимальна. Эту стоимость мы договорились называть «условным оптимальным выигрышем» для данного состояния системы (хотя в данном случае это, конечно, не выигрыш, а проигрыш).

Как указывалось выше, определение условных оптимальных параметров (управления и выигрыша) надо начинать с последнего, 12-го шага. Рассмотрим отдельно правый верхний угол нашей прямоугольной сетки (рис. П1.3). Где мы можем находиться после 11-го шага? Только там, откуда за один (последний) шаг можно попасть в B, т. е. в одной из точек B_1 или B_2. Если мы находимся в точке B_1, у нас нет выбора (управление вынужденное): надо идти на восток, и это обойдется нам в 10 единиц. Запишем это число 10 в кружке у точки B_1, а оптимальное управление покажем короткой стрелкой, исходящей из B_1 и направленной на восток. Для точки B_2 управление тоже вынужденное (на север), затраты до конца равны 14, мы их запишем в кружке у точки B_2. Таким образом, условная оптимизация последнего шага сделана, условный оптимальный выигрыш для обоих состояний этого шага найден и записан в соответствующем кружке.

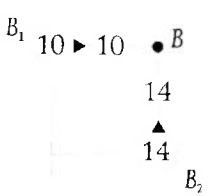


Рис. П1.3

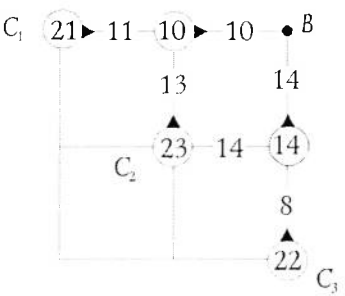


Рис. П1.4

Теперь давайте оптимизировать предпоследний (11-й) шаг. После предпредпоследнего (10-го) шага мы могли оказаться в одной из точек C_1, C_2, C_3 (рис. П1.4). Найдем для каждой из них условное оптимальное управление и условный оптимальный выигрыш. Для точки C_1 управление вынужденное – идти на восток; обойдется это нам до конца в 21 единицу (11 на данном шаге плюс 10 – условный оптимальный выигрыш для точки B_1, записанный в кружке). Число 21 записываем в кружок при точке C_1. Для точки C_2 управление уже не вынужденное: мы можем идти как на восток, так и на север. В первом случае мы затратим на данном шаге 14 единиц и от B_2 до конца – еще 14, всего 28 единиц. Если пойдем на север, затратим 13 + 10 = 23 единицы. Значит, условное оптимальное управление в точке C_2 – идти на север (внимательный читатель, очевидно, заметил, что найдено это управление по критерию, о котором говорилось выше). Отмечаем его стрелкой, а число 23 записываем в кружок у точки C_2. Для точки C_3 управление снова вынужденное – идти на север; обойдется это до конца в 22 единицы (ставим стрелку на север, число 22 записываем в кружок у точки C_3).

Аналогично, пятясь от предпоследнего шага назад, найдем для каждой точки условное оптимальное управление, которое обозначим стрелкой, и условный оптимальный выигрыш, который запишем в кружке. Вычисляется он так: расход на данном шаге складывается с уже оптимизированным расходом, записанным в кружке, куда ведет стрелка. Таким образом, на каждом шаге мы оптимизируем только этот шаг, а следующие за ним уже оптимизированы.

Конечный результат показан на рис. П1.5. На нем видно, что, в какой бы из узловых точек мы ни находились, мы знаем, куда идти (стрелка) и во что нам обойдется путь до конца (число в кружке), а в кружке у точки A записан оптимальный выигрыш (минимальные затраты) на все сооружение пути из A в B!

Теперь остается построить путь, ведущий из A в B, самым дешевым способом. Для этого надо только «слушаться стрелок», т. е. прочитать, что они предписывают делать на каждом шаге. Такая оптимальная траектория пути отмечена на рис. П1.5 оттененными кружками.

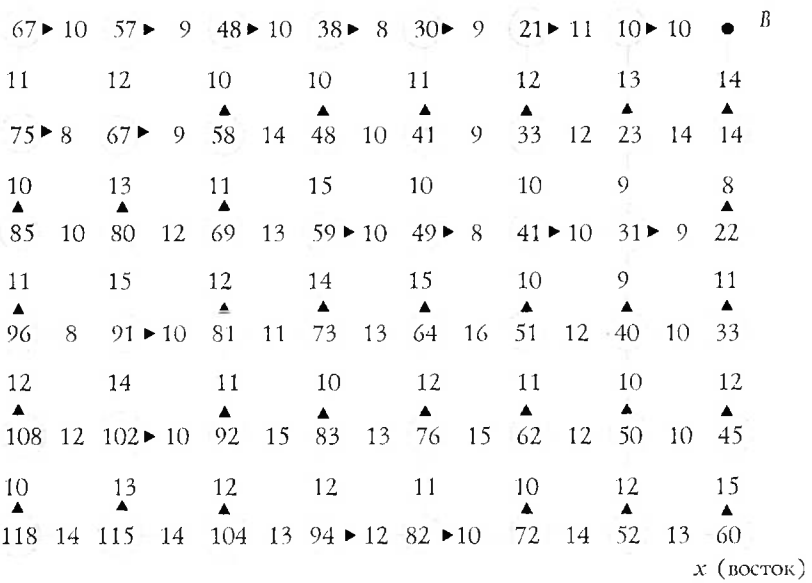

y (север)
67 ► 10 57 ► 9 48 ► 10 38 ► 8 30 ► 9 21 ► 11 10 ► 10 ● B
11 12 10 10 11 12 13 14
75 ► 8 67 ► 9 58 14 48 10 41 9 33 12 23 14 14
10 13 11 15 10 10 9 8
85 10 80 12 69 13 59 ► 10 49 ► 8 41 ► 10 31 ► 9 22
11 15 12 14 15 10 9 11
96 8 91 ► 10 81 11 73 13 64 16 51 12 40 10 33
12 14 11 10 12 11 10 12
108 12 102 ► 10 92 15 83 13 76 15 62 12 50 10 45
10 13 12 12 11 10 12 15
118 14 115 14 104 13 94 ► 12 82 ► 10 72 14 52 13 60
x (восток)

Рис. П1.5

Приложение 2

Нахождение наибольшего общего делителя двух натуральных чисел (алгоритм Евклида)

Древнегреческий математик Евклид в своей знаменитой работе «Начала» (330–320 гг. до н. э.) изложил алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел. Суть этого (считающегося самым древним) алгоритма состоит в следующем. Если число a больше числа b, то нужно от a отнять b, в противном случае, наоборот, от b отнять a и повторять эти действия до тех пор, пока a не станет равно b. После этого искомый НОД будет равен одному из полученных чисел. Сказанное иллюстрирует следующая схема:

$$\text{НОД}(a, b) = \begin{cases} \text{НОД}(a - b, b), & \text{если } a > b \\ \text{НОД}(a, b - a), & \text{если } b < a. \\ a \text{ (или } b), & \text{если } a = b \end{cases}$$

Например, при a = 26, b = 18 имеем: НОД(26, 18) = НОД(8, 18) = НОД(8, 10) = НОД(8, 2) = НОД(6, 2) = НОД(4, 2) = НОД(2, 2) = 2.

Приведем фрагмент программы, работающий по этому алгоритму:

```
нц пока a <> b
  если a > b
    то
      a := a - b
    иначе
      b := b - a
  все
кц
НОД := a | Или НОД := b
```

Возможен ряд усовершествований программы, ускоряющих ее работу [20], однако и усложняющих ее.

Приложение 3

Сортировка массива методом обмена

В данном приложении описан один из простых методов сортировки массивов. Рассматривается сортировка одномерного массива a, состоящего из n элементов. Сортировка проводится в порядке неубывания значений.

Заметим, что запомнить всю программу сортировки сложно, поэтому следует знать особенности описанного метода, на основе которых можно разработать соответствующую программу.

Сортировка обменом – метод, при котором все соседние элементы массива попарно сравниваются друг с другом и меняются местами в том случае, если предшествующий элемент больше последующего (при сортировке в порядке неубывания). Этот процесс повторяется (n - 1) раз.

Например, требуется провести сортировку массива:

30, 17, 73, 47, 22, 11, 65, 54.

Методика сортировки отражена на рис. П3.1 (представлены первые два прохода обработки).

Сравниваемые элементы	Обмен
Первый проход по массиву:	
1) 30 и 17	Проводится
2) 30 и 73	Нет
3) 73 и 47	Проводится
4) 73 и 22	Проводится
5) 73 и 11	Проводится
6) 73 и 65	Проводится
7) 73 и 54	Проводится
Полученный массив: 17, 30, 47, 22, 11, 65, 54, 73	
Второй проход по массиву:	
1) 17 и 30	Нет
2) 30 и 47	Нет
3) 47 и 22	Проводится
4) 47 и 11	Проводится
5) 47 и 65	Нет
6) 65 и 54	Проводится
7) 65 и 73	Нет
Полученный массив: 17, 30, 22, 11, 47, 54, 65, 73	

Рис. П3.1

Если индексы «левых» элементов в паре сравниваемых обозначить лев, то фрагмент программы на школьном алгоритмическом языке, реализующий описанные действия на одном проходе, выглядит так:

```
нц для лев от 1 до n - 1
  | Если "левый" элемент в паре сравниваемых больше "правого"
  если a[лев] > a[лев + 1]
    то | Проводим их обмен
     всп := a[лев]
     a[лев] := a[лев + 1]
     a[лев + 1] := всп
  все
кц
```

а вся процедура сортировки оформляется следующим образом:

```
| Проходим по массиву n - 1 раз,
нц для номер_прохода от 1 до n - 1
  | Сравнивая пары элементов
  нц для лев от 1 до n - 1
    если a[лев] > a[лев + 1]
      то | Проводим обмен
        всп := a[лев]
        a[лев] := a[лев + 1]
        a[лев + 1] := всп
    все
  кц
кц
```

Лирическое отступление ☺. Если последовательность сортируемых чисел расположить вертикально (первый элемент массива – внизу) и проследить за перемещением элементов (рис. П3.2), то можно увидеть, что большие элементы, подобно пузырькам воздуха в воде[1], «всплывают» на соответствующую им позицию. Поэтому сортировку таким способом называют еще сортировкой методом «пузырька», или «пузырьковой сортировкой».

[1] В воде всплывают «легкие» пузырьки.

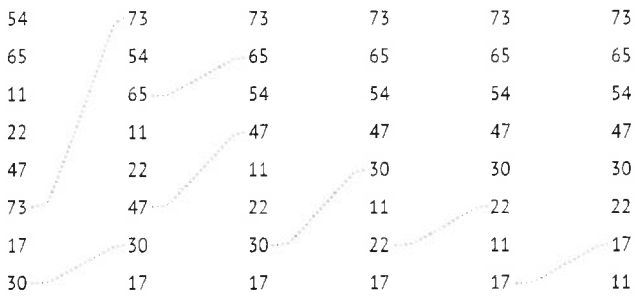


Рис. П3.2

Можно усовершенствовать программу, учитывая следующее обстоятельство. В ходе первого прохода максимальный элемент постепенно смещается вправо и в конце концов занимает свое (которое он должен занимать в упорядоченном массиве – крайнее правое) место в массиве (см. рис. П3.1). После этого его можно исключить из дальнейшей обработки. Затем процесс повторяется, и свое место занимает второй по величине элемент, который также исключается из дальнейшего рассмотрения. Так продолжается до тех пор, пока весь массив не будет упорядочен.

Выпишем пары индексов элементов, сравниваемых на каждом проходе с учетом сказанного только что, в виде табл. П3.1.

Таблица П3.1

Номер прохода по массиву	Индексы				
1-й	1 – 2	2 – 3	...	(n – 2) – (n – 1)	**(n – 1)** – n
2-й	1 – 2	2 – 3		**(n – 2)** – (n – 1)	
...					
(n – 1)-й	**1** – 2				

Если индекс «левого» элемента в последней паре сравниваемых на каждом проходе чисел обозначить посл_лев (соответствующие значения в табл. П3.1 выделены полужирным начертанием), то можно так оформить фрагмент программы, осуществляющей сортировку:

```
нц для посл_лев от n - 1 до 1 шаг -1
  нц для лев от 1 до посл_лев
    если a[лев] > a[лев + 1]
      то | Проводим обмен
        ...
    все
  кц
кц
```

Можно также в качестве параметра «наружного» оператора цикла использовать номер прохода, а конечное значение параметра «внутреннего» оператора цикла связать с номером прохода согласно табл. П3.1:

```
нц для номер_прохода от 1 до n - 1
  нц для лев от 1 до n - номер_прохода
    если a[лев] > a[лев + 1]
      то | Проводим обмен
        ...
    все
  кц
кц
```

По нашему мнению, такой вариант более запоминаем.

Можно также прекратить проходы по массиву, как только он станет отсортированным (в общем случае это может произойти менее чем за n – 1 проходов). Признак, по которому целесообразно установить факт отсортированности массива, – на каком-то проходе обмена элементов местами не было. Однако при этом программа усложняется.

Литература

1. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2021 года по информатике и ИКТ // https://fipi.ru/ege/demoversii-specifikacii-kodifikatory#!/tab/151883967-5. (Демоверсии, спецификации, кодификаторы 2021. Информатика и ИКТ.)

2. Сайт Константина Юрьевича Полякова // http://kpolyakov.spb.ru/school/ege.htm.

3. Система программирования КуМир // https://www.niisi.ru/kumir/.

4. Кодификатор элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2021 году единого государственного экзамена по информатике и ИКТ // https://fipi.ru/ege/demoversii-specifikacii-kodifikatory#!/tab/151883967-5. (Демоверсии, спецификации, кодификаторы 2021. Информатика и ИКТ.)

5. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2020 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2020. Информатика и ИКТ.)

6. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2019 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2019. Информатика и ИКТ.)

7. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2018 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2018. Информатика и ИКТ.)

8. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2017 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2017. Информатика и ИКТ.)

9. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2016 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2016. Информатика и ИКТ.)

10. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2015 года по информатике и ИКТ // http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory. (Демоверсии, спецификации, кодификаторы 2015. Информатика и ИКТ.)

11. Система программирования PascalABC.NET // http://pascalabc.net/

12. *Кабанов А.* Тренажер для подготовки к КЕГЭ // http://92.63.100.177/

13. Спецификация контрольных измерительных материалов для проведения в 2021 году единого государственного экзамена по информатике и ИКТ // https://fipi.ru/ege/demoversii-specifikacii-kodifikatory#!/tab/151883967-5. (Демоверсии, спецификации, кодификаторы 2021. Информатика и ИКТ.)

14. *Мирончик Е. А.* Чеширский Кот путешествует по графу // Информатика. 2015. № 10.

15. *Андреева Е. В.*, *Босова Л. Л.*, *Фалина И. Н.* Математические основы информатики. М.: БИНОМ, 2005.

16. http://kpolyakov.spb.ru/school/ege/retro.htm.

17. *Златопольский Д. М.* Основы программирования на языке Python. 2-е издание. М.: ДМКПресс, 2018.

18. *Вентцель Е. С.* Исследование операций: задачи, принципы, методология. М.: Наука, 1988.

19. *Окулов С. М.* Программирование в алгоритмах. М.: БИНОМ. Лаборатория знаний, 2002.

20. *Златопольский Д. М.* Программирование: типовые задачи, алгоритмы, методы. М.: БИНОМ. Лаборатория знаний, 2007.

21. *Златопольский Д. М.* Подготовка к ЕГЭ по информатке в 2020 годы. Решение задач по программированию. М.: ДМКПресс, 2020.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Златопольский Дмитрий Михайлович

Подготовка к ЕГЭ по информатике в компьютерной форме

Главный редактор *Мовчан Д. А.*

dmkpress@gmail.com
Корректор *Абросимова Л. А.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 60×90 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 19. Тираж 200 экз.

Отпечатано в ООО «Принт-М»
142300, Московская обл., Чехов, ул. Полиграфистов, 1

Веб-сайт издательства: **www.dmkpress.com**