***Степулёнок Денис Олегович***

# РАЗБОР ЗАДАЧ ЛЕНИНГРАДСКОЙ ОБЛАСТНОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ПРОГРАММИРОВАНИЮ

*От редакции.*

*В статье представлены задачи Ленинградской областной олимпиады школьников по информатике, которая проходила на базе Ленинградского областного института развития образования (ЛОИРО) в январе 2004 года. Областные олимпиады являются третьим, предпоследним этапом в многоуровневой системе проведения олимпиад в России. Участники заключительного этапа олимпиады по информатике от Ленинградской области неоднократно занимали призовые места на Всероссийском уровне.*

*С каждым годом олимпиадные задачи по информатике, а по сути они являются задачами по программированию, усложняются. Чтобы решить такую задачу, школьник кроме базовых алгоритмов должен*

*– владеть знаниями, выходящими далеко за пределы школьной программы (например, знаниями по аналитической геометрии, теории графов, комбинаторике, теории вероятностей и т. д.);*

*– хорошо программировать на Паскале или Си, использовать различные типы данных;*

*– уметь оценивать алгоритмы с точки зрения их эффективности и оптимальности.*

*Вот почему стало целесообразно привлекать к подготовке и проведению олимпиад по информатике областного, районного и даже школьного уровня «играющих тренеров» – студентов, участвующих в студенческих чемпионатах по программированию.*

*Последнюю Ленинградскую областную олимпиаду школьников организовали и провели под руководством заведующей кабинетом информатики ЛОИРО В.Г. Савицкой представители Клуба программистов СПбГЭТУ «ЛЭТИ»: Денис Степулёнок, Антон Попович, Алексей Преображенский, Павел Зубарев и Сергей Оршанский (последний – студент СПбИТМО, чемпион мира по программированию среди студенческих команд 2004 года). В этой олимпиаде приняли участие около 40 школьников – победителей районного этапа. Олимпиада проводилась в два тура, где участникам было предложено решить в общей сумме 10 задач, различающихся не только тематикой, но и сложностью реализации. При проверке результатов олимпиады использовалась автоматическая тестирующая система, учитывающая как полные решения (если программа участника правильно отработала на всех предложенных тестах), так и частичные. Предпочтение отдавалось полным решениям с набавлением поощрительных баллов, что должно было стимулировать стремление школьника найти наиболее оптимальный вариант решения, ведь при тестировании каждой задачи учитывался еще лимит времени.*

*Подробнее со всеми материалами олимпиады можно ознакомиться в Интернет на сайте http://acm.eltech.ru/2004.*

*Перед задачами областных олимпиад ставится важная цель – они должны стать начальным звеном в цепи еще более сложных, но и более интересных задач Всероссийского и Международного уровня.*

## А. АРИФМЕТИЧЕСКАЯ ПРОГРЕССИЯ

Для проведения парада по случаю Нового года требуется некоторое количество военнослужащих. На параде военнослужащие занимаются исключительно построением в шеренги. Шеренга называется регулярной, если военнослужащие в ней либо все одного роста, либо рост возрастает от одного конца шеренги к другому каждый раз на одну и ту же величину, образуя тем самым арифметическую прогрессию.

Чтобы достойно выступить на следующем параде, создайте из имеющихся в вашем распоряжении военнослужащих максимальную регулярную шеренгу.

Входные данные

$N$ – количество военнослужащих, имеющихся в наличии ($0 \le N \le 10000$).

Высоты военнослужащих в миллиметрах – $A_1\ A_2\ A_3 \ldots\ A_{N-1}\ A_N$ ($1 \le A_i \le 10000$).

Выходные данные

Количество военнослужащих в максимальной регулярной шеренге.

Высоты военнослужащих в указанной регулярной шеренге – в возрастающем или в убывающем порядке. Если существует несколько ответов – выведите любой из них.

**Указания к решению** (полный текст решения с программой к этой и последующим задачам приведен на диске к журналу):

Удобнее искать арифметическую прогрессию в отсортированном массиве.

Простейший алгоритм поиска максимальной арифметической прогрессии имеет сложность $O(N^3)$:

• Выбираем два последних члена арифметической прогрессии (пусть предпоследний имеет индекс $i$, а последний – $j$). Вычисляем разность прогрессии $D = A[j] - A[i]$.

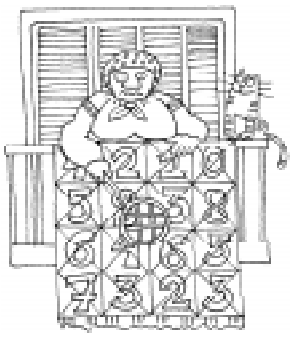
• Третий с конца член прогрессии должен быть равен $A[i] - D$. Бежим от $i$–1-го элемента к началу массива и ищем высоту $A[i] - D$. Если находим – увеличиваем длину прогрессии на 1, вычитаем из найденной высоты $D$ и ищем следующее значение, иначе – заканчиваем работу алгоритма, нам известна длина прогрессии, разность и последний член.

Как ускорить алгоритм? Можно использовать двумерный массив $L$ размером $N^2$. $L_{i,j}$ – максимальная длина прогрессии, у которой предпоследний элемент имеет индекс $i$, а последний – $j$. Динамический переход таков:

$$L_{ij} = \begin{cases} L_{ti} + 1, \text{ если } A[i] \ne A[i], \text{ где} \\ \quad t - \text{индекс элемента со значением } A[i] - D; \\ 2, \text{ если } A[i] \ne A[i], \text{ и элемента} \\ \quad \text{со значением } A[i] - D \text{ не существует;} \\ i - j + 1, \text{ если } A[i] = A[i]. \end{cases}$$

Для быстрого поиска индекса элемента по его значению можно использовать хеш-таблицу (используется в решении – суммарное время работы $O(N^2)$) или двоичный поиск (суммарное время работы $O(N \log N)$).

## В. ДИАГОНАЛЬНЫЕ СУММЫ



Дана квадратная таблица целых чисел $N \times N$. Нужно посчитать суммы чисел по всем

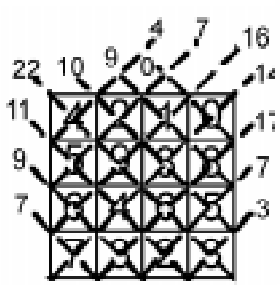



**Рисунок 1.**

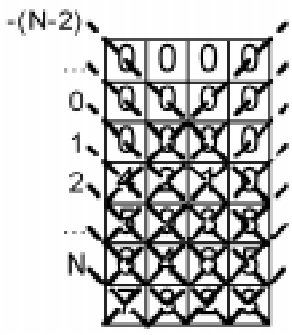



**Рисунок 2.**

диагоналям таблицы (см. рисунок 1).

Входные данные

$N$ – размер таблицы ($1 \le N \le 100$).

$A_{ij}$ – элементы таблицы ($0 \le A_{ij} \le 10000$). Все числа разделены пробелами или переводами строки.

Выходные данные

На первой строчке вывести $2N - 1$ сумм по диагоналям начиная с верхнего правого угла таблицы. На второй строчке вывести $2N - 1$ сумм по диагоналям начиная с верхнего левого угла.

**Указания к решению:**

Для более удобного подсчета диагональных сумм дополним матрицу сверху нулями как показано на рисунке 2.

Матрица:

```
var A :
array [-98..100,1..100] of integer;
```

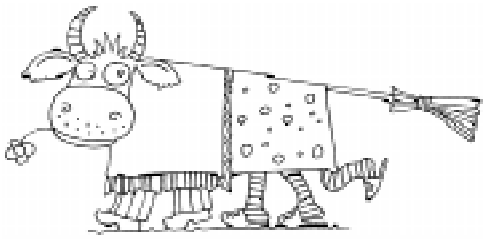
В начале программы заполним всю матрицу нулями:

```
fillchar(A,sizeof(A),0);
```

При чтении исходных данных заполним строки с 1-ой по $N$-ую.

Вычисление сумм производится очевидным циклом (см. листинг 1).

## C. СУММА ПРОСТЫХ



Найдите все пары простых чисел ($A$, $B$), такие, что их сумма также является простым числом и не превосходит $N$.

**Указания к решению:**

Рассмотрим ряд простых чисел: 2, 3, 5, 7, 11, 13, 17, ...

Очевидно, что существует ровно одно четное простое число – 2 (остальные четные числа делятся на 2). Тогда, если простое число – четное, то, оно равно 2.

Итак, пусть простые числа $A$, $B$, $C$ удовлетворяют условиям задачи, то есть $A \le B$ и $A, B < C$.

Тогда возможны 4 случая (таблица 1).

Итак, первым числом в паре всегда будет 2. Задача свелась к поиску всех простых нечётных чисел до 1000000. Основная проблема задачи – быстрый поиск и хранение простых чисел. Используем алгоритм

**Листинг 1.**

```
for k:=-(N-2) to N do begin
  Sum := 0; { В переменной Sum - текущая сумма }
  i := k; { Начинаем подсчет суммы со строки k }
  for j:=1 to N do begin { В каждую сумму входят N элементов }
    Sum := Sum + A[i,j]; { Прибавляем очередной элемент }
    i := i+1; { Сдвигаемся на одну строку вниз }
  end;
  write(Sum,' '); { Выводим сумму и разделяем ее пробелом }
                  { со следующей }
end;
writeln; { Перевод строки }
{ Точно такой же фрагмент используем для вывода второй строчки }
{ выходного файла, только цикл по j от N к 1 }
```

Таблица 1.

| | *A* | *B* | |
|---|---|---|---|
| 1 | четное | четное | $A = 2, B = 2 \Rightarrow C = A + B = 4$ – не является простым числом, этот случай нам не подходит |
| 2 | нечетное | нечетное | $C = A + B =$ *нечетное* + *нечетное* = *четное* = 2, но $A, B < C$, а 2 – минимальное простое число – этот случай нам также не подходит |
| 3 | нечетное | четное | $A > 2, B = 2$, но по условию $A \leq B$ – не подходит |
| 4 | четное | нечетное | $A = 2$, *B – нечетное* $\Rightarrow$ *C – нечетное*, этот случай единственный, который подходит. То есть надо выводить только пары (*A*, *B*), такие что: $A = 2$, *B – нечетное, C* = 2 + *B – простое нечетное* |

«Решето Эратосфена» – его «изюминка» в том, что он очень простой и одновременно «быстрый».

*Идея алгоритма:*

Выписываем все числа от 2 до 1000000: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ... , 1000000

Идем слева – направо. Находим первое не вычеркнутое число (его считаем простым) и вычеркиваем все числа кратные ему: 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, 17, ~~18~~, ... , ~~1000000~~. На каждом шаге выбираем следующее не вычеркнутое число и вычеркиваем все кратные: 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, ... , ~~1000000~~. После того, как мы дойдем до 1000, останутся не вычеркнутыми только простые числа (объясните, почему не надо идти до 1000000).

См. листинг 2.

### D. СТРОКА ГРЕЯ

Последовательность строк Грея *S* задана правилами:

1) *S* [0] = ‘’;

2) *S* [i] := *S* [*i* – 1] + *C* [*i*] + + *S* [*i* – 1];

где *C* [*i*] – это *i*-ая буква латинского алфавита (*C* [1]=‘*a*’, *C* [2]=‘*b*’...), а знак ‘+’ обозначает слияние строк.

Таким образом:

*S* [1] = ‘*a*’;

*S* [2] = ‘*aba*’;

*S* [3] = ‘*abacaba*’;

*S* [4] = ‘*abacabadabacaba*’ и. т. д.

Дано число *k*. Определите, какой символ стоит на *k*-ом месте в строке *S* [26]. Гарантируется, что *k* лежит в пределах от 1 до длины строки *S* [26] включительно.

**Указания к решению:**

Очевидно, что длина строки Грея всегда нечетна.

*Теорема.* На всех нечетных позициях стоит буква ‘*a*’, на четных позициях буквы ‘*a*’ нет.

Докажем это по индукции: *S* [1] = ‘*a*’ – верно, пусть для *S* [*i* – 1] – верно. Докажем что верно для *S* [*i*].

На рисунке 3 видно, что в начале строки *S* [*i*] нечетные позиции совпадают с нечетными позициями строки *S* [*i* – 1]. По-

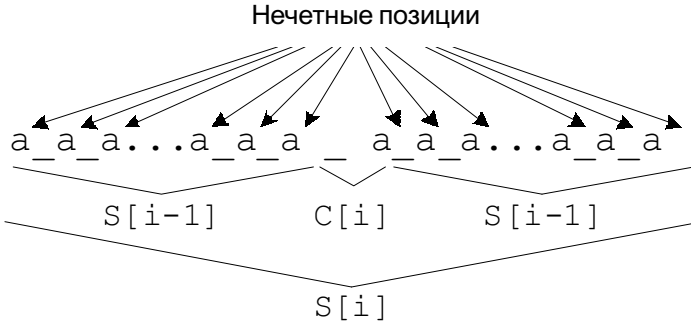



**Рисунок 3.**

**Листинг 2.**

```
var
  P : array [2..1000000] of boolean; { Создаем массив P }
            { из элементов логического типа. После работы алгоритма }
           { элемент массива P[i] будет хранить является ли число i }
           {простым }
  i : Longint; { Текущее не вычеркнутое число }
  j : Longint; { Число, кратное i }
  Count : Longint; { Количество пар чисел A,B }
                   { удовлетворяющих условию задачи }
begin
  fillChar(P,sizeOf(P),true); { Инициализация: заполняем массив P }
                { значениями true (истина) - выписываем все числа.}
                { Потом будем их вычёркивать }
  for i:=2 to 1000 do
    if P[i] then begin { Если i не вычеркнуто (является простым }
                       { числом), то вычеркнем все числа кратные i }
      j := i+i; { Первое число кратное i }
      while j<=1000000 do begin { Пока не вышли за границы массива P }
        P[j] := false; { Вычеркиваем }
        j := j+i; { Следующее число кратное i }
      end;
    end;
  Count := 0; { Теперь остается только посчитать количество пар (A,B) }
  for i:=3 to 1000000-2 do
    if P[i] and P[i+2] then Count := Count + 1;
                                                { A=2, B=i, C=A+B=2+i }
  writeln(Count);
  for i:=3 to 1000000-2 do { И вывести сами пары }
    if P[i] and P[i+2] then writeln( '2 ',i ); { A=2, B=i, C=A+B=2+i }
end;
```

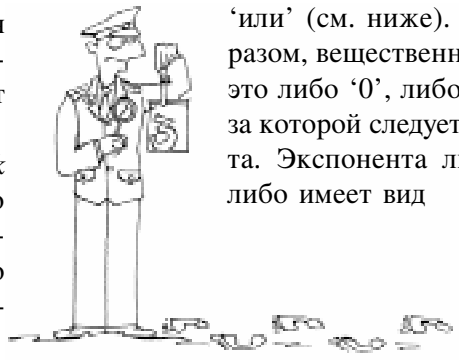
скольку длина $S[i-1]$ по лемме нечетная, символ $C[i]$ (не равный ‘$a$’) занимает четную позицию, тогда первый символ ‘$a$’ второй строки $S[i-1]$ занимает четную позицию, а, следовательно, и далее все нечетные символы попадают на нечетные места, а четные символы – на четные места.

Аналогично можно доказать, что если из строки Грея удалить все символы ‘$a$’, то на нечетных позициях окажутся символы ‘$b$’, если затем удалить все символы ‘$b$’, то на нечетных позициях будут ‘$c$’ и так далее.

Дальше все просто: делим позицию $k$ нацело на 2. Если остаток – единица, то позиция нечетная – выводим ‘$a$’, иначе заменяем $k$ на частное и снова делим нацело на два. Если остаток – единица, то выводим ‘$b$’ и так далее.

## E. ВЕЩЕСТВЕННОЕ ЧИСЛО

Во входном файле дано машинное («вещественное число»), записанное в стандартном виде. Необходимо определить, правильно ли оно записано. Формат записи вещественного числа приведен ниже в форме Бэкуса-Наура ($ – пустая строка).



Символ ‘|’ означает ‘или’ (см. ниже). Таким образом, вещественное число – это либо ‘0’, либо мантисса, за которой следует экспонента. Экспонента либо пуста, либо имеет вид

<e><знак><не_ноль><любые_цифры>...
<вещ_число> ::= 0 | <мантисса><экспонента>
<мантисса> ::= <не_ноль><дробная_часть>
<не_ноль> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<дробная_часть> ::=
$ |.<любые_цифры><не_ноль>
<любые_цифры> ::=
$ | <цифра><любые_ цифры>
<цифра> ::= 0 | <не_ноль>
<экспонента> ::=
$ | <e><знак><не_ноль>
<любые_цифры>
<e> ::= E | e
<знак> ::= + | -

**Решение:**

Нужно проверить, является ли содержимое строки *S* вещественным числом, то есть подходит ли оно под вышеописанную грамматику. Для точной проверки опишем грамматику в виде конечного автомата. Конечный автомат будет состоять из состояний и переходов. В начальный момент времени возможным будет только начальное состояние. Каждый символ строки *S* будет делать доступным другое множество состояний. Если в конце окажется доступным конечное состояние – выводим YES, иначе выводим NO (см. рисунок 4).

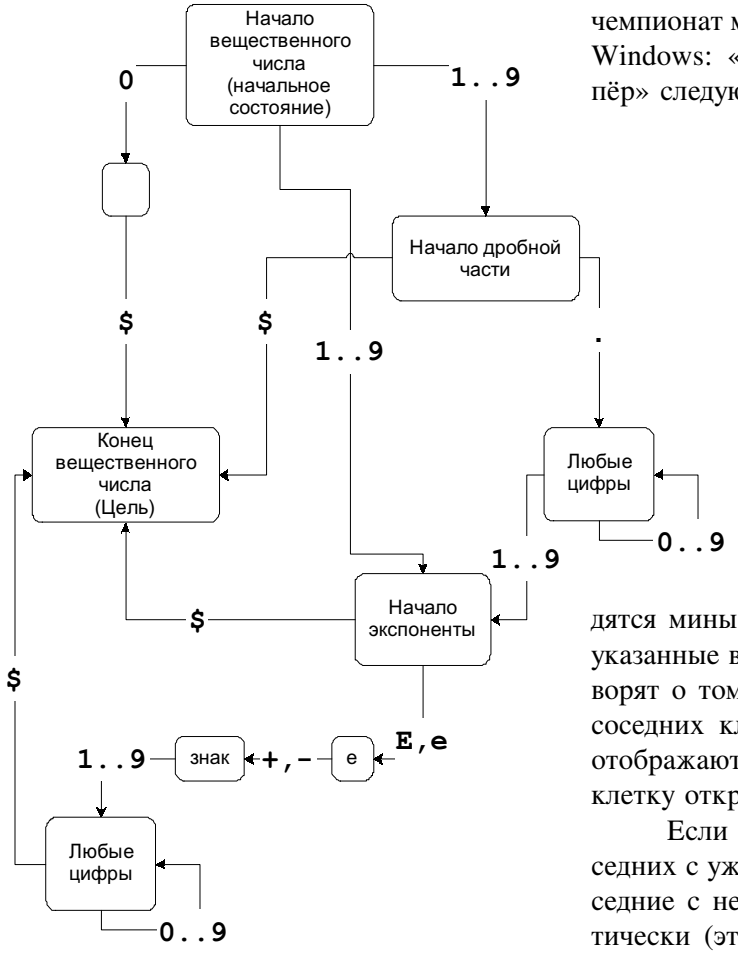



**Рисунок 4.**

**F. САПЁР**

Недавно было решено организовать чемпионат мира по игре в стандартную игру Windows: «Сапёр». Правила игры в «Сапёр» следующие.

Игра происходит на прямоугольном клеточном поле. В начале игры все поля закрыты. В некоторых из них скрываются мины, тогда как в других – нет. Игрок последовательно открывает некоторые клетки. Если он открывает клетку с миной, то он проиграл, игра окончена.

Необходимо открыть все клетки поля, где нет мин, при этом пометить флажками все клетки поля, в которых находятся мины. Подсказками являются числа, указанные в клетках, где нет мин – они говорят о том, сколько мин располагается в соседних клетках (от 1 до 8). Эти числа отображаются только тогда, когда данную клетку открывает игрок (см. рисунок 5).

Если оказывается, что в клетках, соседних с уже открытой, мин нет, то все соседние с ней клетки открываются автоматически (это можно видеть на рисунке 1; по эстетическим соображениям число ‘0’ не пишется).

По этим данным мы сразу можем сказать, в каких клетках точно находятся мины (см. рисунок 6).

Очень частой является ситуация, когда неоткрытым остается только один крайний столбец, причем цифры в клетках столбца, соседнего с крайним, показывают количество мин в соседних клетках именно крайнего столбца, так как мин во втором и в третьем с краю столбце нет (см. рисунок 7).

Такие ситуации очень сложно разрешаются человеком, поэтому ваш друг, желающий потренироваться перед чемпионатом, попросил написать вас программу, которая по данным второго с края столбца определяет, если это возможно, в каких клетках крайнего столбца находятся мины. Это возможно сделать, например, если в одной из клеток второго с края столбца указано число 3 (см. рисунок 8).

Тогда можно точно сказать, что во всех трех соседних клетках находятся мины. Затем сразу можно определить, в каких клетках они находиться не могут (см. рисунок 9).

После этого мы можем заявить, что во второй снизу клетке крайнего левого столбца есть мина, так как в клетках, соседних с 3-ей снизу клеткой второго слева столбца располагается 2 мины, одна из которых найдена, и только одна клетка осталась не изученной. Значит именно там и находится мина. Далее мы можем аналогично восстановить полную картину.

Иногда бывает, что точно определить, где находятся мины, а где не находятся, невозможно (см. рисунок 10).

**Рисунок 5.** Типичная ситуация в игре «Сапёр».

**Рисунок 6.**

**Рисунок 7.**

**Рисунок 8.**

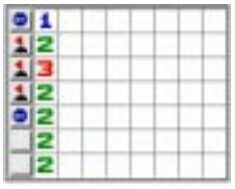


**Рисунок 9.**

**Рисунок 10.**

Иногда бывает и такое, что поле явно ошибочно, и не существует такого расположения мин, при котором клетки соседнего с крайним столбца не могут принимать такие значения (см. рисунок 7, обратите внимание на верхнюю часть столбца).

Вам надо написать программу, которая по данным второго с края столбца определит, сколько существует вариантов расстановки мин в последнем столбце.

Входные данные:

$N$ – высота поля ($2 \le N \le 10000$). Далее идут $N$ чисел $a_i$ ($0 \le a_i \le 3$) – значения $i$-ого поля сверху в предпоследнем столбце.

Выходные данные:

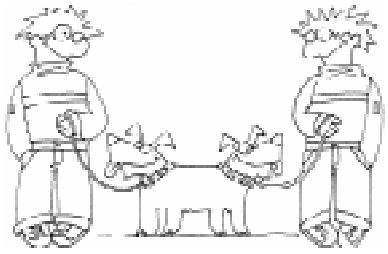
Количество вариантов расстановки мин в последнем столбце.

В лексикографическом порядке возможные варианты расстановки мин. Расстановки выводить вверху вниз, обозначая: ‘#’ – мина есть, ‘_’ – мины нет.

**Указания к решению:**

Пусть известно, что в самой верхней клетке $b_1$ мин. Тогда в следующей клетке будет $b_2 = a_1 - b_1$ мин. В третьей клетке будет $b_3 = a_2 - b_1 - b_2$ мин. В $i$-ой клетке будет $b_i = a_{i-1} - b_{i-1} - b_{i-2}$ мин. То есть достаточно выбрать количество мин в первой клетке – мины в остальных клетках расставятся однозначно. Количество мин в первой клетке может быть 0 или 1, то есть возможно максимум два варианта расстановки мин.

### G. ПАЛИНДРОМ



Мальчику Пете подарили компьютер. Недавно ему надо было найти файл на жестком диске. Так он познакомился с регулярными выражениями. Регулярное выражение – это строка, которая помимо обычных символов (букв, цифр и знаков подчеркивания) может содержать также и спецсимволы. Спецсимволы бывают двух видов: ‘?’ и ‘*’ . Вопросительный знак обозначает любой символ, а звездочка – любую (возможно даже пустую) строчку из любых символов. Петя придумал новый спецсимвол: ‘!’ , который обозначает ровно три любых символа.

Узнав на занятии математического кружка о палиндромах (строках, которые одинаково читаются как слева направо, так и справа налево, например: «арозаупалапалапуазора», «янестарбратсеня», «удаввазду»), Петя придумал задачу, но никак не может ее решить. Помогите ему решить задачу.

Дана строка, представляющая собой регулярное выражение. Необходимо сказать, можно ли содержащиеся в ней спецсимволы заменить по вышеприведенным правилам обычными символами так, чтобы получившаяся строка была палиндромом.

Входные данные

В первой строке файла находится число *N* – количество регулярных выражений. Далее следуют *N* строк, содержащих регулярные выражения, каждое выражение не превышает по длине 255 символов и состоит только из латинских букв, цифр, знаков подчеркивания, вопросительных и восклицательных знаков и звездочек.

Выходные данные

Каждая строка выходного файла должна содержать либо единственное слово YES, либо слово NO. Ответ YES, если строка во входном файле может быть преобразована в палиндром путем правильной замены спецсимволов, и слово NO в противном случае.

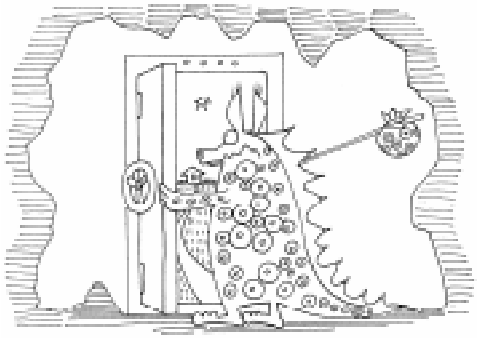
**Указания к решению:**

1. Для упрощения задачи заменим в строке исходных данных все знаки ‘!’ на ‘???’.

2. Будем решать задачу методом динамического программирования, то есть будем сводить задачу определения, является ли строка палиндромом, к задаче с меньшей длиной строки.

Будем идти одновременно с начала и конца строки, то есть будем рассматривать первый и последний символ строки. Тогда возможны следующие ситуации (см. таблицу 2).

### H. DOOM IV



Игровое пространство представляет собой совокупность прямоугольных пещер на плоскости, параллельных осям координат. Пещеры соединены между собой телепортами. Любая пара пещер может быть соединена одним или несколькими двусторонними телепортами. Из-за сопротивления гиперпространственной среды скорость телепортации обратно пропорциональна расстоянию между концами телепорта. Для простоты мы будем считать, что время телепортации (в наносекундах) в точности равно проходимому расстоянию (в миллиметрах).

Каждый монстр хочет попасть в свою домашнюю пещеру, а находиться он может совершенно в другом месте. Вам дан план пещер уровня и схема телепортов. Необходимо вывести минимальное время (в нано-

Таблица 2.

| № | Первый символ | Последний символ | Действия |
|---|---|---|---|
| 1 | Буква | Буква | • Если буквы равны => удаляем первый и последний символ и сводим задачу к анализу строки на 2 символа короче<br>• Если буквы не равны => получить палиндром невозможно => выводим NO и заканчиваем алгоритм |
| 2 | Буква | ? | Заменяем '?' на такую же букву и уменьшаем строку на 2 символа |
| 3 | ? | Буква | |
| 4 | ? | ? | Заменяем оба '?' на любую букву и уменьшаем строку на 2 символа |
| 5 | Буква | * | Если хотя бы с одной стороны '*' => вместо звёздочки мы можем подставить «перевёрнутую» другую сторону и получим палиндром => выводим YES и заканчиваем алгоритм |
| 6 | ? | * | |
| 7 | * | * | |
| 8 | * | Буква | |
| 9 | * | ? | |
| 10 | Остался один символ или пустая строка | | Выводим YES и заканчиваем алгоритм |

секундах), требуемое каждому монстру для попадания в свою домашнюю пещеру.

Различные пещеры не совпадают, не касаются друг друга и не пересекаются.

Предполагается также, что монстры перемещаются внутри пещер с бесконечной скоростью.

Одним телепортом может одновременно пользоваться любое количество монстров в обоих направлениях.

Входные данные

$N$ ($1 \le N \le 250$) – количество пещер. $N$ строк содержат координаты $X_1$, $Y_1$, $X_2$, $Y_2$ ($-10^6 \le X_1, Y_1, X_2, Y_2 \le 10^6$; $X_1 < X_2$, $Y_1 < Y_2$) – левая нижняя и правая верхняя координаты пещеры (в миллиметрах). $M$ ($0 \le M \le N^2$) – количество телепортов. $M$ строк содержит номера двух пещер, соединяемых телепортом. Телепорт начинается и заканчивается в центре пещеры (на пересечении диагоналей). $K$ ($0 \le K \le 10000$) – количество монстров. $K$ строк содержат по два числа: $F$ – номер пещеры, из которой выходит очередной монстр, $T$ – номер его домашней пещеры.

Выходные данные

Вам необходимо определить минимальное время, которое потребуется каждому монстру для возвращения домой, и вывести в файл это самое время, (в наносекундах) – с точностью до третьего знака после запятой. Если достичь $T$ (домашней пещеры) невозможно, следует вывести '*Impossible*'.

**Указания к решению:**

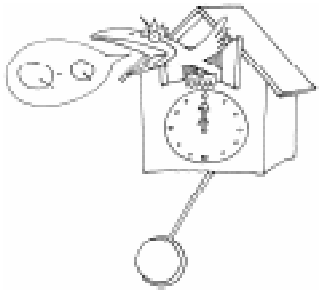
Если присмотреться к этой задачи, то можно «увидеть» в сети пещер граф. Пещеры – вершины графа. Телепорты – дуги графа. Тогда задача разбивается на 3 подзадачи:

• Сформировать граф в виде матрицы смежности;

• Найти кратчайший путь между каждой парой вершин;

• Считать номера пещер и вывести ответ.

## I. ИНТРОСПЕКТИВНОЕ ЧИСЛО



Во входном файле дано число $Q$. В выходном файле необходимо вывести

*Q*-разрядное число в системе счисления с основанием *Q*, такое, что первый (старший) разряд числа сообщает о количестве единиц в этом числе, второй разряд говорит о количестве двоек и т. д., предпоследний разряд говорит о количестве цифр *Q* – 1 в числе, а последний разряд сообщает о количестве нулей в этом числе. Если таких чисел несколько, вывести любое из них. Если таких чисел не существует, вывести единственное число '0'. Для вывода цифр более 9 в системах счисления с основанием более 10, использовать большие буквы латинского алфавита ('A' – 'Z').

**Указания к решению:**

Напишем переборное решение задачи в надежде уловить закономерность в интроспективных числах.

Получаем следующий файл:

```
- 2 -
- 3 -
- 4 -
0202
2101
- 5 -
12002
- 6 -
- 7 -
2110003
- 8 -
21010004
- 9 -
210010005
- 10 -
2100010006
- 11 -
21000010007
- 12 -
210000010008
```

Файл содержит все интроспективные числа до *Q* = 12. Начиная с *Q* = 7 явно прослеживается закономерность. Теперь можно написать итоговое решение.

## J. ИГРА В ТОЧКИ

Среди школьников и студентов всего мира быстро набирает популярность игра «в точки» (другое название – Феодалы). Существует несколько модификаций игры, для определенности будем использовать следующие правила игры:

– игра ведется на листочке в клеточку (19 на 19 клеток);

– игроков двое, они по очереди ставят точки разного цвета (у каждого игрока свой цвет) на любом свободном пересечении линий сетки (в уголках клеточек). Всего уголков 20 × 20;

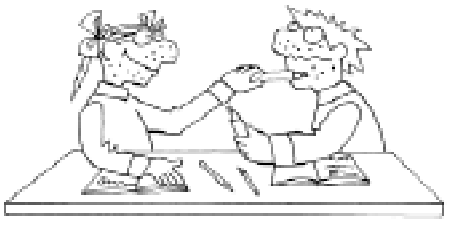
– некоторые точки являются *активными*, тогда как другие – *пассивными*. Любая свежепоставленная точка является активной. В дальнейшем некоторые из них могут неоднократно менять свое состояние (об этом ниже);

– контуром называется последовательность активных точек, в которой любые две соседние точки граничат на плоскости либо по горизонтали, либо по вертикали, либо по диагонали, в том числе первая граничит с последней;

– после каждого хода каждого игрока выполняется так называемое *замыкание*, то есть рассматриваются контуры ходившего игрока, и все точки противника, входящие во внутреннюю область, по крайней мере, одного контура, объявляются пассивными, а все такие пассивные точки ходившего объявляются активными. Кроме того, все свободные пересечения, входящие во внутреннюю область, по крайней мере, одного контура объявляются *запрещенными*, то есть ходить в них в дальнейшем никому нельзя;

– игра продолжается определенное число ходов (по договоренности игроков), после чего подсчитываются очки каждого из участников – число пассивных точек противника на момент окончания игры.

Ваша задача состоит в том, чтобы написать программу, определяющую по последовательности ходов финальный счет (промоделировать игру).



Входные данные:

В первой строке одно целое число $1 \le N \le 400$ – количество ходов. Далее следуют строки с координатами точек ($1 \le X_i \le 20$, $1 \le Y_i \le 20$): во второй строке находятся координаты первой точки первого игрока, в третьей строке – первой точки второго и т. д. по две координаты в каждой строке.

Выходные данные:

В выходной файл вывести 20 строк по 20 символов – финальное поле.

Обозначения:

‘-’ – пусто (ничья пустая клетка);

‘,’ – пустая клетка первого игрока (в нее ходить нельзя);

‘.’ – пустая клетка второго игрока (в нее ходить нельзя);

‘A’– точка первого игрока (активная);

‘B’– точка второго игрока (активная);

‘a’– пленная (пассивная) точка первого игрока;

‘b’– пленная (пассивная) точка второго игрока.

В последней строке выходного файла должны находиться два целых числа разделенных двоеточием – счет игры (очки первого игрока: очки второго игрока).

**Указания к решению:**

При моделировании игры будем хранить в памяти поле – 2-мерный массив символов. Каждому уголку листа соответствует один символ. Будем моделировать игру по ходам: на каждом ходу добавлять очередную точку и находить контуры, образованные точками текущего игрока. Для поиска контуров образованных точками текущего игрока прелагается использовать заливку внешней области – применяется алгоритм заливки с затравкой. Чтобы внешняя область всегда была непрерывна нужно вокруг поля оставить слой в одну свободную клетку. При этом затравку (начало заливки) можно всегда помещать в клетку (0; 0). После заливки все символы, которые не попали во внешнюю область, следует изменить по следующим правилам:

| Символ | Ход игрока A | Ход игрока B |
|---|---|---|
| A | – | Заменяем на ‘a’ |
| B | Заменяем на ‘b’ | – |
| a | Заменяем на ‘A’ | – |
| b | – | Заменяем на ‘B’ |
| _ , . | Заменяем на ‘.’ | Заменяем на ‘,’ |

*Полный текст решений задач и программы приведены на диске к журналу.*

***Степулёнок Денис Олегович,***
***студент 5 курса СПбГЭТУ,***
***руководитель Клуба программистов***
***СПбГЭТУ «ЛЭТИ».***