Московская городская олимпиада по информатике

1 тур. 8 февраля 2004 года

Задачи и решения

Автор решений — С.В.Шедов

Москва. 2004

Задача A Наибольшее произведение

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	80 баллов

Дано N целых чисел. Требуется выбрать из них три таких числа, произведение которых максимально.

Формат входных данных

Во входном файле записано сначала число N — количество чисел в последовательности ($3 \le N \le 10^6$). Далее записана сама последовательность: N целых чисел, по модулю не превышающих 30000.

Формат выходных данных

В выходной файл выведите три искомых числа в любом порядке. Если существует несколько различных троек чисел, дающих максимальное произведение, то выведите любую из них.

Примеры

a.in	a.out
9 3 5 1 7 9 0 9 -3 10	9 10 9
3 -5 -30000 -12	-5 -30000 -12

Система оценки

Решения для ограничения $3 \le N \le 100$ оцениваются 30 баллами, для ограничения $3 \le N \le 5000$ — 60 баллами.

Решение

Сначала попробуем решить задачу очевидным способом, который сразу приходит на ум. Переберем все тройки чисел и выберем из них такую, которая имеет максимальное произведение.

Приведем фрагмент реализации такого способа на языке Pascal:

```
var
   n : longint; {число элементов в последовательности}
   a : array [1..MaxN] of integer; {массив из элементов последовательности}
   i, j, k : longint;
   t, max : longint;
   i1, i2, i3 : longint; {индексы найденной максимальной тройки}
begin
     ...
     max := -MaxLongInt-1;
     {перебираем все различные тройки элементов последовательности}
     for i := 1 to n do
         for j := 1 to n do
             for k := 1 to n do
                 if  (i<>j) and (j<>k) and (i<>k) then {все три индекса различны}
                  begin
                      t := longint(a[i]) * a[j] * a[k];
                      if t >= max then
                       begin
                            max := t;
                            i1 := i; i2 := j; i3 := k;
                       end;
                  end;
      writeln(a[i1], ' ', a[i2] ,' ', a[i3]);
end.
```

В этом решении существуют две проблемы. Во-первых, оно успевает сработать примерно только при $N<100$. Нетрудно подсчитать, что количество раз, которое выполняется тело цикла (количество итераций цикла) равно $N*N*N$ или N^3. Для подсчета времени работы программы удобно считать, что за 1 секунду выполняется порядка 1 миллиона итераций цикла (разумеется, если в теле цикла содержатся вызовы функций или другие циклы, то эта оценка неверна).

Кроме того, у решения есть еще один недостаток. Элементы последовательности по модулю могут достигать 30000 и, вообще говоря, их произведение может выйти за пределы 32-битного целого типа longint. Можно схитрить и решить эту проблему, используя, например, тип extended для переменных *t* и *max*.

Однако есть более красивое решение, которое полностью решает задачу с учетом ограничений, поставленных в задаче и при этом не прибегает ни к каких хитростям. Рассмотрим его.

В задаче нам необходимо найти такие три числа, произведение которых максимально. То есть, какие бы другие три числа мы не выбирали, их произведение будет всегда меньше или равно максимальному. Теперь давайте сообразим, какие именно числа в последовательности могут давать максимальное произведение. Первое, что приходит на ум – три максимальных числа последовательности. Для последовательностей с неотрицательными элементами это действительно так.

Рассмотрим пример:

3	1	5	0	***9***	4	***6***	2	***6***	6

Когда все числа в последовательности неотрицательны, то максимальное произведение дадут именно три максимальных элемента. В нашем примере это `9*6*6=324`.

Остается понять, как искать три максимальных элемента. Алгоритм нахождения максимального элемента массива широко известен и не требует пояснений. Но нам надо найти не только максимальный элемент, но и «второй» и «третий» максимумы. В нашем примере первый максимум = 9, второй (следующий по значению) = 6, третий = 6. Обратите внимание, что максимумы могут совпадать по значению.

К счастью, задача поиска трех максимумов решается несложно. Пусть в переменных max1, max2, max3 хранятся первый, второй и третий максимум соответственно. Присвоим им начальные значения, равные –30000. В процессе работы программы они будут либо улучшены, либо оставлены без изменений (в случае, если последовательность состоит только из минимально возможных чисел –30000).

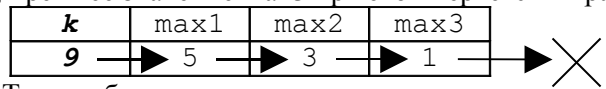
Воспользуемся идеей «проталкивания сверху вниз» очередного элемента в текущие три максимума. Снова обратимся к нашему примеру. Пусть мы уже просмотрели четыре элемента последовательности и правильно заполнили переменные max1, max2 и max3.

max1	Max2	max3
5	3	1

Мы считали из входного файла очередное число последовательности ***k*=9**. Сначала сравним его с переменной max1.

```
if k > max1 then
 begin
       max3 := max2;
       max2 := max1;
       max1 := k;
 end
```

Поскольку 9>5, то произведем «проталкивание» нового максимума – запишем ***k*** в max1, а в max2 – старое значение max1 (ведь оно было больше или по крайней мере равно max2!). В max3 запишется старое значение max2, прежнее значение max3 при этом теряется – хранить его нет больше смысла.



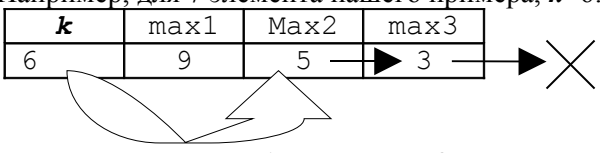
Таким образом, мы получим:

max1	max2	max3
9	5	3

Если окажется, что ***k*≤**max1, тогда следует его сравнить с max2.

```
if k > max2 then
 begin
       max3 := max2;
       max2 := k;
 end
```

Например, для 7 элемента нашего примера, ***k*=**6:



Мы не изменяем max1, вместо max2 записывается ***k***, а на место max3 становится прежний max2.

В противном случае сравнивается ***k*** и max3.

```
if k > max3 then max3 := k;
```

Можно реализовывать не «проталкивание сверху вниз», а «проталкивание снизу вверх». Программная логика в этом случае претерпит лишь небольшие изменения:

```
if k>max3 then max3 := k;
if k>max2 then
 begin
     max3 := max2;
     max2 := k;
 end;
if k>max1 then
 begin
     max2 := max1;
     max1 := k;
 end;
```

Но у нас в последовательности могут быть еще и отрицательные числа! Произведение двух отрицательных чисел положительно и поэтому необходимо найти два минимальных отрицательных числа. Произведем поиск первого и второго минимума min1 и min2 в последовательности аналогичным методом.

Теперь сравним два произведения: min1*min2 и max2*max3. Нам надо выбрать максимальное из них. Очевидно, что максимальным произведением из трех элементов будет максимальный элемент последовательности max1, умноженный на максимум из min1*min2 и max2*max3.

Вспомним, что в первом решении у нас были проблемы с переполнением при перемножении трех элементов последовательности. Чтобы избежать этих проблем здесь, мы не будем непосредственно производить перемножение трех чисел, а будем сравнивать между собой min1*min2 и max2*max3.

Однако и это еще не все! Тем и замечательны олимпиадные задачи, что мы должны учитывать все возможные случаи, которые допускают ограничения на входные данные. А что будет, если все числа в последовательности отрицательные? Тогда, в случае min1*min2>max2*max3 наш алгоритм найдет далеко не максимальное произведение. Ясно, что в этом случае ответом задачи будут просто три максимальных элемента массива, так как максимальным произведением (отрицательным!) будет max1*max2*max3.

Все остальные случаи полностью покрываются нашим алгоритмом. В том числе и различные случаи с нулем. В случае последовательности из отрицательных элементов и нескольких нулей максимальное произведение будет равно 0, но 0 в этом случае в нашем алгоритме обязательно попадет в max1, а остальные два элемента значения играть не будут. Если же в последовательности есть хотя бы один положительный элемент, то max1 никогда не будет равен нулю, поэтому если нулю оказались равны max2 и max3, то при наличии отрицательных элементов максимальное произведение будет формироваться из min1, min2 и max1.

Приведем полный текст решения данной задачи на языке Pascal. Обратите внимание на то, что мы не создаем массив и не храним последовательность элементов в памяти. Поскольку задача решается в один проход по массиву, то для нашего алгоритма этого и не требуется – в каждый момент времени нам требуется знать только один текущий элемент последовательности. Мы обрабатываем его, а затем «забываем».

```
program MOI2004_1;
var
   n : longint; {число элементов в последовательности}
   max1, max2, max3 : longint;
     {первый, второй и третий максимумы в последовательности}
   min1, min2 : longint;
     {первый и второй минимумы в последовательности}
   k : integer; {текущий элемент последовательности}
   i : longint;
begin
     assign(input, 'a.in');
     reset(input);
     assign(output, 'a.out');
     rewrite(output);
     max1 := -30000; max2 := max1; max3 := max1;
     min1 := 30000; min2 := min1;
     readln(n);
     {читаем по очереди все элементы последовательности}
     for i:=1 to n do
```

```
      begin
         read(k);
         if k > max1 then
           begin
            max3 := max2; max2 := max1; max1 := k;
           end
         else
         if k > max2 then
           begin
            max3 := max2; max2 := k;
           end
         else
         if k > max3 then
              max3 := k;

         if k < min1 then
          begin
              min2 := min1; min1 := k;
          end
         else
         if k < min2 then
              min2 := k;
      end;

      if (max1 > 0) and (min1*min2 > max2*max3) then
         writeln(max1, ' ', min1, ' ', min2)
      else
         writeln(max1, ' ', max2, ' ', max3);
      close(input);
      close(output);
end.
```

Задача B Покупка билетов

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	100 баллов

За билетами на премьеру нового мюзикла выстроилась очередь из N человек, каждый из которых хочет купить 1 билет. На всю очередь работала только одна касса, поэтому продажа билетов шла очень медленно, приводя «постояльцев» очереди в отчаяние. Самые сообразительные быстро заметили, что, как правило, несколько билетов в одни руки кассир продаёт быстрее, чем когда эти же билеты продаются по одному. Поэтому они предложили нескольким подряд стоящим людям отдавать деньги первому из них, чтобы он купил билеты на всех.

Однако для борьбы со спекулянтами кассир продавала не более 3-х билетов в одни руки, поэтому договориться таким образом между собой могли лишь 2 или 3 подряд стоящих человека.

Известно, что на продажу i-му человеку из очереди одного билета кассир тратит A_i секунд, на продажу двух билетов — B_i секунд, трех билетов — C_i секунд. Напишите программу, которая подсчитает минимальное время, за которое могли быть обслужены все покупатели.

Обратите внимание, что билеты на группу объединившихся людей всегда покупает первый из них. Также никто в целях ускорения не покупает лишних билетов (то есть билетов, которые никому не нужны).

Формат входных данных

Во входном файле записано сначала число N — количество покупателей в очереди ($1 \le N \le 5000$). Далее идет N троек натуральных чисел A_i, B_i, C_i. Каждое из этих чисел не превышает 3600. Люди в очереди нумеруются начиная от кассы.

Формат выходных данных

В выходной файл выведите одно число — минимальное время в секундах, за которое могли быть обслужены все покупатели.

Примеры

b.in	b.out
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12
2 3 4 5 1 1 1	4

Решение

Попробуем найти решение задачи, постепенно увеличивая размер очереди. Пусть у нас в очереди стоит всего один человек. Поскольку каждому человеку нужен один и только один билет (даже если 2 или 3 билета купить быстрее), то ответом задачи будет число A_1.

Теперь перейдем к очереди из 2 человек. Они могут купить билеты раздельно, тогда время покупки составит $A_1 + A_2$. Кроме того, они могут объединиться и купить билеты вместе, причем купить два билета по условию задачи может только первый человек. Из двух возможностей выбираем такую, которая займет меньше времени и запоминаем это время в элементе D[2] специального массива. Таким образом, D[2] – минимальное время покупки билетов очередью из 2 первых человек.

Добавим в очередь третьего человека. Какие варианты есть у нас? Если третий человек покупает билет самостоятельно, то первые два, очевидно, не зависят от него. Тогда (внимание!) мы уже решили эту задачу! Мы только что нашли минимальное время покупки билетов для очереди из двух человек, оно хранится в D[2]. Общее время покупки в этом случае составит D[2]+A_3. Допустим, второй и третий человек решат купить билеты вместе. Ответом в этом случае будет $B_2 + A_1$ (два билета покупает второй человек и первый человек покупает билет самостоятельно). Наконец, договориться смогут и все три человека. Тогда они купят билеты за C_1 – именно столько времени займет покупка трех билетов у первого стоящего в очереди человека. Запишем лучший вариант в D[3] – это будет минимальное время покупки билетов очередью из 3 первых человек.

Рассмотрим все вышесказанное на примере из условия.

N=5

i	A_i	B_i	C_i
1	5	10	15
2	2	10	15
3	5	5	5
4	20	20	1
5	20	1	1

Заполним массив D начальными значениями, D[1] = A_1, остальные элементы пока неизвестны.

D[1]	D[2]	D[3]	D[4]	D[5]
5	???	???	???	???

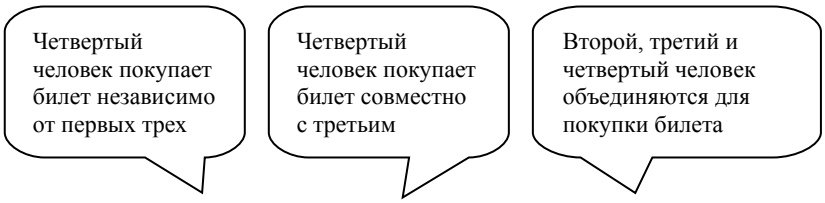
Переходим к выяснению значения D[2]. Имеем, что при покупке билетов раздельно время составит $A_1 + A_2 = 5 + 2 = 7$, а при покупке вместе - 10. Итак, D[2]=min(7,10)=7

D[1]	D[2]	D[3]	D[4]	D[5]
5	**7**	???	???	???

Добавляем в очередь третьего человека. Если он покупает билет отдельно, то общее время равно D[2] + A_3 = 7 + 5 = 12. Если второй и третий договариваются между собой, то общее время будет D[1] + B_2 = 5 + 10 = 15. Наконец, в случае совместной покупки билетов тремя любителями мюзиклов, им удастся это сделать за время C_1 = 15. Разумеется, выбираем самый быстрый вариант и в данном случае D[3]=min(12,15,15)=12.

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	**12**	???	???

Дальше будем действовать совершенно аналогично. Добавляем к очереди четвертого человека. Выпишем формулу для D[4].



D[4] = min (D[3]+A_4 , D[2] + B_3 , D[1] + C_2)

Итак, D[4]=min(12+20, 7+5, 5+15)=min(32, 12, 20)=12

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	12	**12**	???

Продолжая рассуждения аналогично, в общем виде получаем следующую формулу:

D[i] = min (D[i-1]+A_i , D[i-2] + B_{i-1} , D[i-3] + C_{i-2})

Ответом к задаче будет служить элемент массива D[N]. В нашем случае это D[5]. Дорешаем наш пример: D[5]=min(6+20, 5+20, 7+5)=min(26, 25, 12)=12

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	5	6	**12 – ответ задачи**

Принцип, используемый при решении этой задачи называется *динамическим программированием*. В данном случае мы решаем полную задачу, постепенно расширяя ее размерность и используя решения более маленьких «подзадач».

Приведем полное решение задачи на языке Pascal.

```
const
   MaxN = 5000;
var
   n : word; {число человек в очереди}
   a, b, c : array [0 .. MaxN] of integer;
   d : array [0 .. MaxN] of longint;
```

```
   i : integer;

function min(a,b : longint) : longint;
begin
     if a<b then min:=a else min:=b;
end;

begin
     assign(input, 'b.in');
     reset(input);
     assign(output, 'b.out');
     rewrite(output);

     readln(n);
     for i := 1 to n do
         read(a[i], b[i], c[i]);

     d[0] := 0;
     d[1] := a[1];
     d[2] := min(a[1]+a[2], b[1]);

     for i := 3 to n do
         d[i] := min( d[i-1] + a[i], min( d[i-2] + b[i-1], d[i-3] + c[i-2] ) );

     writeln(d[n]);
     close(input);
     close(output);
end.
```

Задача C Вырезанные фигуры

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	8 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	120 баллов

Эпидемия гриппа не обошла стороной семиклассника Алешу. Скучая дома, Алеша решил вырезать фигурки из листа клетчатой бумаги. Лист состоял из M строк и N столбцов клеточек. Сначала Алеша нарисовал на листе границы фигур. Количество фигур было не меньше 2. Чтобы фигуры получались ровными, границы фигур Алеша рисовал строго по линиям имеющейся клеточной разметки листа (при этом некоторые границы фигур могли пройти по границам листа). Форма фигур могла быть любой, но при этом все фигуры были связными (фигура называется *связной*, если из любой ее клетки можно добраться до любой другой, ходя только по клеткам фигуры и перемещаясь каждый раз в одну из 4-х соседних по стороне клеток). Никакие две фигуры не имели общих точек, в том числе не касались углами клеток.

Затем Алеша вырезал нарисованные фигуры, делая разрезы только по их границам. При этом оставшаяся часть листа осталась связной (то есть не распалась на несколько частей).

Лист с вырезами Алеша отсканировал. Сканер в своей памяти по результатам сканирования построил таблицу, состоящую из нулей и единиц, из M строк и N столбцов (строки нумеруются сверху вниз от 1 до M, стролбцы — слева направо от 1 до N). Каждый элемент таблицы соответствовал клеточке исходного листа. Единица обозначала, что соответствующая клетка листа осталась на месте, ноль — соответствующая клетка была вырезана.

На рис. 1 приведен пример клетчатого листа, а на рис. 2 — соответствующая ему таблица в памяти сканера:

Рис 1.

Исходный клеточный лист с вырезанными фигурами
Размер листа: M=6, N=12.
Количество вырезанных фигур: 3

1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	0	0	0	0	0	1
1	0	0	1	1	1	1	0	0	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1
1	1	0	0	1	1	1	0	1	0	1	1
1	1	0	0	1	1	1	1	1	1	1	1

Рис 2.

Такая таблица строится в памяти сканера

После этого сканер представил полученную таблицу в специальном, описанном ниже формате и передал информацию на компьютер. Напишите программу, которая по полученной информации установит:

Пункт 1. Сколько клеток было вырезано из листа?

Пункт 2. Сколько фигур было вырезано?

Описание формата представления таблицы

Последовательность подряд идущих по горизонтали или вертикали единиц будем называть полосой. Полосу можно задаеть 4 числами:

- направление (0—горизонтальная, 1—вертикальная)
- (i, j) — координаты начальной клетки полосы (начальной является самая левая клетка для горизонтальной полосы, и самая верхняя — для вертикальной), i — номер строки клетки, j — номер столбца
- d — длина полосы (количество подряд стоящих единиц).

Всю таблицу разобьем на полосы, состоящие из единиц так, чтобы каждая единица принадлежала хотя бы одной полосе. При этом полосы могут пересекаться, а также накладываться. Таким образом, таблица представляется в виде описания всех полос, которое удовлетворяет трем дополнительным требованиям:

- В каждой клетке начинается не более одной полосы.
- Полосы перечислены в порядке следования их начальных клеток (клетки перечисляются по строкам сверху вниз, в строке — слева направо).
- Общее число полос не превышает 256000.

Заметим, что таблица может быть представлена в виде полос разными способами, но каждое представление позволяет однозначно восстановить таблицу.

Формат входных данных

Во входном файле записано сначала число P (1 или 2) — номер пункта задачи, ответ на который требуется получить. Далее записаны размеры исходного листа — числа M и N ($1 \le M \le 4000$, $1 \le N \le 4000$). Затем записано число K ($0 \le K \le 256000$) — количество полос в описании полученной таблицы. Затем идет K четверок чисел, описывающих полосы (полосы перечисляются в порядке начальных клеток полос: по строкам сверху вниз, в строке — слева направо).

Формат выходных данных

В выходной файл выведите искомое количество (если P=1, то — количество клеток, вырезанных из листа, если P=2, то — количество фигур, вырезанных из листа).

Примеры

c.in	c.out
1 40 400 2 1 1 100 40 0 1 101 1	15959
2 40 400 2 1 1 100 40 1 1 101 1	2
1 6 12 17 0 1 1 10 1 1 4 4 0 1 5 2 0 1 7 6 1 2 1 5 1 2 5 4 0 2 6 1 1 2 12 5 0 3 4 4 0 3 10 3 0 4 2 2 1 4 11 3 1 5 2 1 0 5 6 2 1 5 9 2 0 6 1 2 0 6 5 7	22
2 6 12 12 0 1 1 12 1 1 12 6 1 2 1 5 0 2 4 3 0 3 4 4 0 3 10 3 1 3 11 4 0 4 1 5 1 5 2 2 0 5 5 3 1 5 9 2 0 6 5 8	3

Система оценки

Полное решение каждого из пунктов 1 и 2 оценивается 60 баллами. Решение для ограничения $1 \le M \le 100$, $1 \le N \le 100$ в каждом из пунктов будет оцениваться 20 баллами.

Решение

Вначале рассмотрим решение первого пункта задачи. Необходимо по внутреннему формату представления данных сканера перевести их в удобный нам формат - таблицу. Заметим, что задача, подобная данной, сплошь и рядом встречается в реальной жизни.

Наш сканер является полностью задокументированным устройством, кроме того, каждое представление позволяет однозначно восстановить таблицу. Саму таблицу мы хранить в памяти не будем – памяти просто не хватит. Можно было бы организовать побитовое хранение матрицы, но это сопряжено с техническими проблемами, которых мы постараемся избежать при помощи эффективного алгоритма. Будем получать таблицу последовательно по строкам, считая по ходу количество белых клеток. Еще раз подчеркнем, что саму таблицу мы в памяти хранить не будем.

Сначала решим упрощенную задачу. Допустим, что у нас не бывает вертикальных полос, а есть только горизонтальные. Рассмотрим решение на примере. Пусть дан фрагмент исходной таблицы:

I\j	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												

Она кодируется следующими пятью горизонтальными полосами:

```
1 полоса: 1 5 4 – начальная клетка полосы (1,5), ее длина равна 4.
2 полоса: 1 7 3
3 полоса: 2 2 2
4 полоса: 2 7 5
5 полоса: 2 11 1
```

Расставим в таблице цифры, соответствующе полосам, которые описывают данную клетку.

I\j	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1	1,2	1,2	2			
2		3	3				4	4	4	4	4,5	

Как видим, некоторые клетки могут описываться сразу несколькими полосами. Считываем из файла самую первую полосу и храним ее в переменных `newi`, `newj` и `d` – начальная клетка полосы и ее длина. Затем идем по матрице до тех пор, пока текущая клетка не станет равна начальной клетке первой полосы. Очевидно, что все пройденные до этого момента клетки – белые. Не забываем считать их.

Начнем писать программу, постепенно уточняя ее:

```
for i := 1 to m do
  begin
   for j := 1 to n do
    begin
         if (i = newi) and (j=newj) then
         begin
         ... {дошли до полосы, надо что-то делать}
         end else
         inc(counter); {увеличиваем счетчик белых клеток}
     end;
  end;
```

Для нашего примера мы пройдем четыре белые клетки, пока не поравняемся с первой полосой. (`i=1, j=5, d=4`). Теперь обрабатываем полосу, в которую мы пришли. В переменной `maxj` будем хранить столбец, где кончается первая полоса. Очевидно, что `maxj=j+d-1=5+4-1=8`.

i\j	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1	1	**maxj**	2			
2		3	3				4	4	4	4	4,5	

Таким образом, при дальнейшем проходе по первой строке, если номер столбца `j` будет меньше, чем `maxj`, то текущая клетка – черная, поскольку на ней лежит первая полоса.

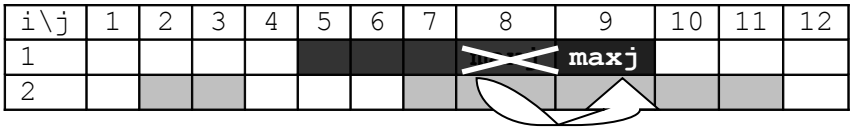
На этом можно считать обработку первой полосы законченной. Поскольку эта полоса нам больше не понадобится, то считываем в переменные `newi, newj` и `d` следующую полосу. Так как по условию в каждой клетке начинается не более одной полосы, то обработку новой полосы мы проведем позже: когда придем в клетку, где она начинается. Обратите внимание, что в нашем алгоритме мы пользуемся тем фактом, что полосы в исходном файле задаются по строкам слева направо, иначе бы данный алгоритм был совершенно неприменим!

Вот как будет выглядеть текущая версия программы:

```
for i := 1 to m do
  begin
   for j := 1 to n do
    begin
         if (i = newi) and (j=newj) then
```

```
            begin
                maxj := j+d-1;
              {обновляем указатель на окончание черной полосы}
                readln(newi, newj, d); {читаем новую полосу}
            end;
            if (j > maxj) then inc(counter);
            {данная клетка белая, так как не покрывается
             горизонтальной полосой}
        end;
    end;
```

Дальше мы пройдем две черные клетки (1,5) и (1,6), пока не поравняемся с началом второй полосы. `i=1, j=7, d=3`. Обрабатываем вторую полосу, то есть определим где она кончается. `maxj=j+d-1=7+3-1=9`. Указатель `maxj` сейчас указывает на столбец 8, а новая полоса заканчивается в 9 столбце, значит, указатель нужно передвинуть на 9, так как новая полоса простирается дальше прежней.



Такую проверку необходимо вставить в нашу программу:

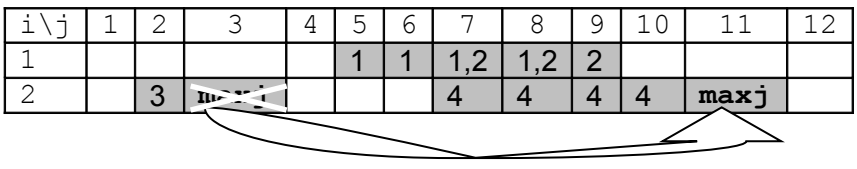
```
if maxj < j+d-1 then
   maxj := j+d-1;
```

Таким образом, мы подсчитываем белые клетки тогда, когда текущий столбец `j` не покрывается никакой горизонтальной полосой, то есть `j > maxj`. Для первой строки мы найдем 7 белых клеток. А что необходимо сделать при переходе на новую строку? Разумеется – обнулить указатель `maxj`. Он был актуален только для предыдущей строки, а на новой строке будут уже новые полосы.

Вот как будет выглядеть уточненная версия нашей программы:

```
for i := 1 to m do
  begin
   for j := 1 to n do
    begin
         if (i = newi) and (j=newj) then
         begin
            if maxj < j+d-1 then
               maxj := j+d-1;
            readln(newi, newj, d);
         end;
         if (j > maxj) then inc(counter);
     end;
    maxj := 0;
    {получили очередную строку, переходим на следующую}
  end;
```

Теперь переходим к обработке второй строки. Как только мы поравняемся с началом очередной полосы, то указатель `maxj` сместится на столбец 3, и пока j не превысит это значение, белые клетки считаться не будут. Затем мы пройдем три белые клетки и дойдем до четвертой полосы и здесь опять указатель изменит свое значение. В отличие от предыдущей строки, он не удлиняет текущую полосу, а указывает на конец новой полосы, но сути дела это не меняет.



Обратите внимание, что когда мы дойдем до пятой полосы, то указатель `maxj` обновлен не будет, так как пятая полоса заканчивается одновременно с четвертой.

Вернемся к исходной задаче и добавим вертикальные полосы. Поскольку мы, как и раньше, будем производить проход по строкам, то вертикальная полоса, которая началась в какой-то предыдущей строке, может влиять на текущую.

Рассмотрим пример:

i\j	1	2	3	4	5	6
1		1			2	2
2		1				
3		1,3		4		5
4		3		4		

Полосы 1, 3 и 4 – вертикальные, 2 и 5 – горизонтальные (для полосы в одну клетку на самом деле это не существенно). Когда мы будем обрабатывать вторую строку, то последней считанной полосой будет вторая, и мы уже не будем «помнить» информацию о первой полосе. Тем не менее, первая полоса дает на второй строке одну черную клетку и мы должны ее учесть.

Для этого заведем массив X из N элементов, где в X[j] записываем в какой строке заканчивается максимальная из просмотренных ранее вертикальных полос j-го столбца. Очевидно, что в процессе работы программы элементы этого массива могут изменяться. Например, после обработки первой полосы X[2]=3, а после обработки третьей значение X[2] будет изменено на 4. В четвертом столбце у нас только одна вертикальная полоса, значение X[4]=4 изменено не будет.

Таким образом, если `i<=x[j]`, то текущая клетка покрывается какой-то вертикальной полосой, рассмотренной ранее. Изменим проверку того, является ли текущая клетка белой:

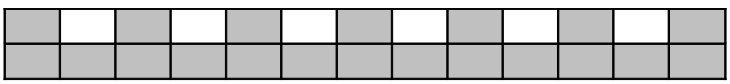
```
if (i > x[j]) and (j > maxj) then inc(counter);
```

Программа претерпит небольшие изменения. Приведем ее полный текст:

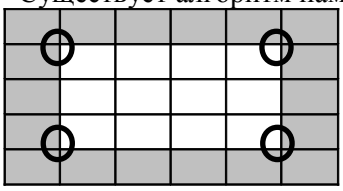
```
const
   MaxN = 4000;
var
   p : byte; {пункт задачи}
   m, n : word; {размер листа}
   k, ck : longint;
      {общее количество полос, количество считанных полос}
   newi, newj : word; {начальная клетка очередной полосы}
   dir : byte; {направление полосы}
   d : word; {длина полосы}
   x : array [1..MaxN] of integer;
      {массив для текущего состояния вертикальных полос}
   maxj : word;
      {максимальное окончание полосы в текущей горизонтали}
   i, j : word;
   counter : longint; {счетчик вырезанных клеток}

procedure init;
begin
   assign(input, 'c.in');
   reset(input);
   fillchar(x, sizeof(x), 0);
   readln(p, m, n, k);
   readln(dir, newi, newj, d);
   ck := 1;
   counter := 0;
   maxj := 0;
end;

procedure print;
var
   i, j : integer;
begin
    assign(output, 'c.out');
    rewrite(output);
    writeln(counter);
    close(output);
```

```
    close(input);
end;

procedure PutNewLine(dir, i, j, d : word; var maxj : word);
begin
      if (dir = 0) then {горизонтальное направление}
       begin
           if maxj < j+d-1 then
              maxj := j+d-1;
             {обновляем окончание горизонтальной
               полоски в текущей строке}
       end
      else {вертикальное направление}
       begin
           if x[j]<i+d-1 then
              x[j] := i+d-1;
             {обновляем окончание вертикальной полоски
              в данном столбце}
       end
end;

procedure solve;
begin
   for i := 1 to m do
    begin
     for j := 1 to n do
      begin
           {если текущая клетка - начало новой полосы}
           if (i = newi) and (j=newj) then
           begin
              {обработаем эту полосу}
              PutNewLine(dir, i, j, d, maxj);
              {есть ли еще полосы для считывания}
              if ck<k then
               begin
                 readln(dir, newi, newj, d);
                 inc(ck)
               end
           end;
           if (i > x[j]) and (j > maxj) then inc(counter);
           {данная клетка белая, так как не покрывается ни
           вертикальной, ни горизонтальной полосой}
       end;
     maxj := 0; {получили очередную строку,
                 переходим на следующую}
    end;
end;

begin
   init;
   solve;
   print;
end.
```

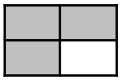
Перейдем к решению второго пункта задачи. Мы произвели перевод данных из внутреннего формата сканера в таблицу, но саму таблицу, как уже отмечалось выше, в памяти мы хранить не можем. Поэтому необходимо подсчитывать число фигур «на лету», имея в распоряжении одну или несколько строк. Можно применить один из известных алгоритмов послойного связывания, но наряду со сложностями в реализации он обладает недостатком – низкой производительностью на примерах, где области постоянно приходится связывать заново, например:

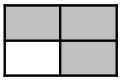


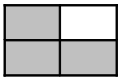
Существует алгоритм намного проще! Допустим, Алеша вырезал прямоугольник.

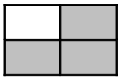


Назовем внешними углами прямоугольника шаблоны вида:





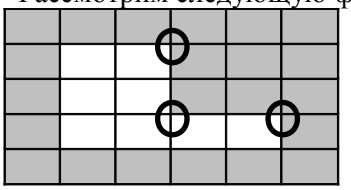




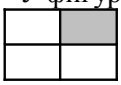
Поскольку фигуры не соприкасаются между собой, в том числе по диагонали, то существует всего 4 вида внешних углов. Кроме того, у каждого прямоугольника ровно четыре внешних угла. Если бы Алеша вырезал только прямоугольники, то достаточно было бы посчитать количество внешних углов, а затем разделить их количество на четыре.

Что меняется, когда Алеша вырезает не только прямоугольники?

Рассмотрим следующую фигуру:

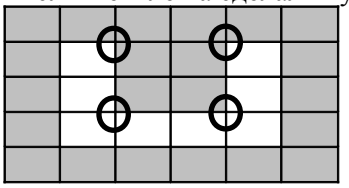


У фигуры появился один внутренний угол:



Очевидно, что внутренние углы – инвертированные шаблоны внешних углов. Количество внешних углов увеличилось на 1 – исчезает один старый внешний угол и появляются два новых.

Если же Алеша сделал внутренний вырез из прямоугольника:



То у фигуры появляются два внутренних угла, но появляются и два новых внешних угла! Легко заметить, что какие бы вырезы не делал Алеша, разность между количеством внешних углов и количеством внутренних углов остается постоянной. Это – инвариант. Поэтому число фигур:

$$F = \frac{Out - In}{4}$$

где Out – количество внешних углов всех фигур, In – количество внутренних углов всех фигур.

Реализовать этот способ в программе очень несложно. В каждый момент времени необходимо хранить только две строки таблицы и сравнивать с шаблоном квадраты размера 2x2.

Тесты к задачам олимпиады, а также решения жюри в электронном виде можно скачать с сайта олимпиады www.olympiads.ru/moscow