Московская городская олимпиада по информатике

2 тур. 15 февраля 2004 года

Задачи и решения

Авторы решений — С.В.Шедов,
Д.Н.Королев, П.И.Митричев

Москва – 2004

Задача D Квадрат

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	60 баллов

Требуется в каждую клетку квадратной таблицы размером NxN поставить ноль или единицу так, чтобы в любом квадрате размера KxK было ровно S единиц.

Формат входных данных

Во входном файле записаны три числа — N, K, S ($1 \le N \le 100$, $1 \le K \le N$, $0 \le S \le K^2$).

Формат выходных данных

В выходной файл выведите заполненную таблицу. Числа в строке должны разделяться пробелом, каждая строка таблицы должна быть выведена на отдельной строке файла. Если решений несколько, выведите любое из них.

Примеры

d.in	d.out
3 2 1	0 0 0 0 1 0 0 0 0
4 2 2	1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0

Примечание

Частичные решения для случая K=2 будут оцениваться примерно половиной баллов.

Решение

Это задача на идею. Когда ее знаешь, то решение кажется очевидным. Однако придумать такое решение самому иногда (а точнее даже очень часто) не так-то просто. Раскроем секрет задачи: достаточно сгенерировать любой квадрат ***K*K***, содержащий ровно ***S*** единиц, а затем просто заполнить им квадрат ***N*N***.

Теперь проделаем все вышесказанное более аккуратно и подробно.

Шаг первый. Необходимо получить любой квадрат размером ***K*K***, в котором будет ***S*** единиц. Сделаем это каким-либо простым способом. Например, сначала заполним квадрат нулями. Затем будем проходить его по строкам и ставить единицы до тех пор, пока не поставим столько, сколько нам нужно. Если за полный проход по квадрату нам так и не удалось поставить ***S*** единиц, то это значит, что задача не имеет решения. Такой случай возможен только при $S > K^2$, а это противоречит условию.

Приведем функцию на языке Pascal, которая генерирует необходимый квадрат. Функция **gen** получает размер квадрата **k**, необходимое число единиц в нем **s** и массив для записи результата. Функция возвращает **true**, если удалось сгенерировать квадрат, отвечающий нашим требованиям и **false** в противном случае:

```
const
   MaxK=100;

type
   Square = array [1..MaxK, 1..MaxK] of byte;
   {квадрат KxK, повторением которого получаем ответ}
```

```
function gen(k, s : word; var a : Square) : boolean;
var
   i, j : integer;
   CurS : word; { сколько единиц уже удалось поставить в квадрате KxK }
Begin
   CurS:=0;
   fillchar(a, sizeof(a), 0);   {вначале заполняем квадрат нулями}
   for i := 1 to k do
     for j := 1 to k do
        if CurS < s then   {если число поставленных единиц меньше необходимого}
         begin
            a[i, j] := 1;  {то ставим очередную единицу}
            inc(CurS);
         end
     end;
     if CurS < s then gen := false else gen := true;
end;
```

Шаг второй. Распространим полученный квадрат размера ***K*K*** на искомый квадрат ***N*N***. Сначала рассмотрим, почему это действительно приводит к правильному результату.

Пусть ***N*** = 7, ***K*** = 3, ***S*** = 4. Приведенная выше функция **`gen`** получит следующий квадрат:

1	1	1
1	0	0
0	0	0

Назовем его образцом и заполним им квадрат размера 7*7:

I\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Обратите внимание, что в заполненном таким образом квадрате любой подквадрат размером 3x3 имеет сумму, равную 4. Почему? Рассмотрим различные подквадраты 3x3 и проследим, что происходит с их суммой. Подквадрат с левым верхним углом (1,1) является образцом, которым мы заполняли большой квадрат, поэтому его сумма, разумеется, равна четырем. Сдвинемся вправо и посмотрим на подкварат с левым верхним углом в ячейке (1,2)

I\j	1	2	3	4	5	6	7
1	*1*	1	1	*1*	1	1	1
2	*1*	0	0	*1*	0	0	1
3	*0*	0	0	*0*	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Первый и второй столбец нового квадрата принадлежат и старому. А что же приобретенный в результате сдвига новый столбец? Он тоже был в старом, поскольку 4 столбец большого квадрата заполняется тем же самым квадратом-образцом! Выделенный квадрат можно свести к образцу поменяв столбцы – третий на место первого, а первый и второй сдвинуть вправо. А поскольку от перемены мест столбцов сумма элементов квадрата не меняется, то и новый квадрат имеет требуемую сумму.

Теперь рассмотрим квадрат, сдвинутый на одну строку вниз. Сдвигаясь вниз, мы целиком «потеряли» первую строчку квадрата-образца, но и снова «приобрели» ее! Новый квадрат тоже легко формируется из квадрата-образца, но к перестановке столбцов необходимо добавить перестановку строк.

I\j	1	2	3	4	5	6	7
1	1	***1***	***1***	***1***	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	***1***	***1***	***1***	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Легко видеть, что как бы мы не сдвигали подквадрат, мы всегда теряем те же самые значения ячеек, что и приобретаем в результате сдвига! Это относится и к случаю, когда мы «упираемся» в границы квадрата *N*N*.

I\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Реализовать заполнение квадрата *N*N* в программе предельно просто. Приведем ключевой фрагмент программного кода:

```
if gen(k, s, a) then       {генерируем образец}
   for i := 1 to n do      {заполняем образцом квадрат размером N*N}
    begin                  {результат выводим сразу в файл}
        for j := 1 to n do
            write(a[(i-1) mod k+1,(j-1) mod k+1], ' ');
        writeln;
    end;
```

Задача E Поле Чудес

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	60 баллов

Для игры в "Поле чудес" используется круглый барабан, разделенный на сектора, и стрелка. В каждом секторе записано некоторое число. В различных секторах может быть записано одно и то же число.

Однажды ведущий решил изменить правила игры. Он сам стал вращать барабан и называть игроку (который барабана не видел) все числа подряд в том порядке, в котором на них указывала стрелка в процессе вращения барабана. Получилось так, что барабан сделал целое число оборотов, то есть последний сектор совпал с первым.

После этого ведущий задал участнику вопрос: какое наименьшее число секторов может быть на барабане? Напишите программу, отвечающую на этот вопрос.

Формат входных данных

Во входном файле записано сначала число N — количество чисел, которое назвал ведущий ($2 \le N \le 30000$). Затем записано N чисел, на которые указывала стрелка в процессе вращения барабана. Первое число всегда совпадает с последним (в конце стрелка указывает на тот же сектор, что и в начале). Числа, записанные в секторах барабана, — натуральные, не превышающие 32000.

Формат выходных данных

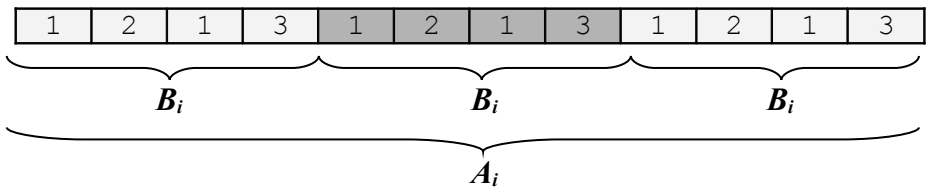
Выведите минимальное число секторов, которое может быть на барабане.

Примеры

e.in	e.out
13 5 3 1 3 5 2 5 3 1 3 5 2 5	6
4 1 1 1 1	1
4 1 2 3 1	3

Решение

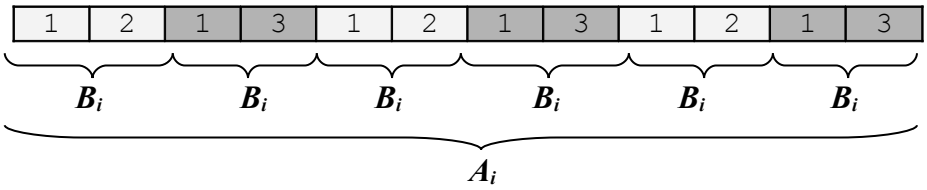
Отбросим последнее число, которое сказал ведущий и сформулируем задачу по другому: дана последовательность чисел A_i длины N. Найти подпоследовательность B_i минимальной длины, повторением которой получена исходная последовательность.



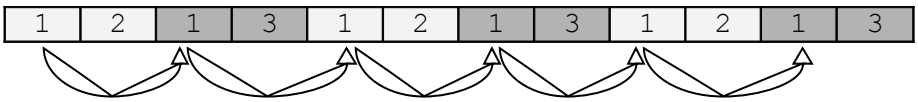
Заметим, что такая последовательность B_i всегда существует: по крайней мере это сама исходная последовательность A_i (повторенная один раз).

Ограничения задачи дают возможность решать задачу перебором с некоторыми отсечениями. Пусть последовательность A_i длины N была получена повторениями последовательности B_i длины K. Поскольку последовательность B_i уложилась в последовательности A_i целое число раз, то N делится на K. Кроме того, K может изменяться в пределах от **1** до N. Будем проверять, является ли K делителем N и если да, образована ли A_i из B_i. Сделать это можно несколькими способами.

Рассмотрим пример. $N = 12$, $K = 2$. Проверяем, образована ли последовательность A_i повторениями из последовательностей длины два. Для этого разобьем A_i на N/K=6 частей и проверим, что получившееся таким образом подпоследовательности B_i одинаковы.



Сначала «по цепочке» проверим, что все первые элементы последовательностей B_i равны.



В программе это будет выглядеть так:

```
j := 1;
while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
```

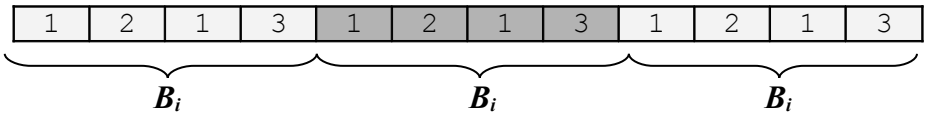
Затем проверим на равенство все вторые элементы. В нашем примере мы сразу получим неравенство ($2 \neq 3$), поэтому повторением последовательности длины 2 исходная последовательность получена не была.

В общем случае необходимо проверить «по цепочке» на равенство все элементы последовательностей B_i от 1 до K.

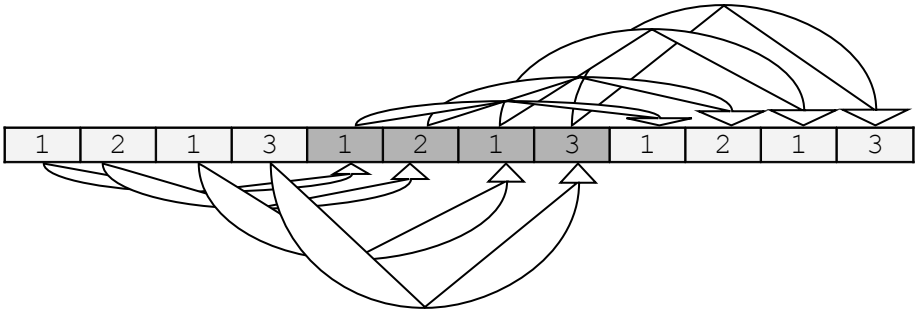
Данный алгоритм реализуется следующим образом на языке Pascal:

```
res := 0;
for k := 1 to n do {перебираем все возможные длины в порядке возрастания}
  if n mod k=0 then
   begin
       Ok := true;
       for i := 1 to k do
       begin
            j := i;
            while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
            if j <= n-k then Ok := false;      { текущая проверка не прошла}
        end;
        if Ok then             { все проверки прошли успешно,
          begin                  значит последовательность B найдена }
              res := k;
              break;
          end;
    end;
```

Можно было поступить по другому. Рассмотрим на примере той же последовательности. N=12, K=4. Разбиваем A_i на N/K=3 части и опять проверим, совпадают ли все получившееся после такого разбиения подпоследовательности B_i



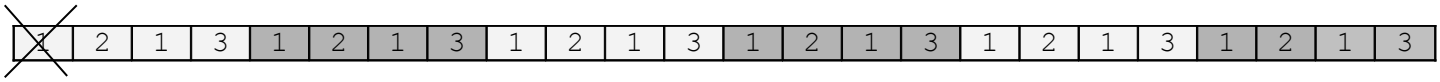
Для этого будем последовательно сравнивать на равенство каждый элемент первой подпоследовательности с соответствующим элементом второй, каждый элемент второй последовательности – с соответствующим элементом третьей и так далее. Здесь мы пользуемся свойством транзитивности равенства, то есть свойством, что из равенств a=b и b=c следует, что a=c.



Такой способ запрограммировать еще проще. Мы последовательно проходим по последовательности A_i (счетчик `j`) и каждый раз сравниваем `a[j]` и `a[j+k]`.

```
res := 0;
  for k := 1 to n do {перебираем все возможные длины в порядке возрастания}
    if n mod k=0 then
     begin
        j := 1;
        while (j <= n-k) and (a[j] = a[j+k]) do inc(j);
        if j > n-k then
         begin {все проверки прошли успешно, значит последовательность B найдена}
             res := k;
             break;
         end;
     end;
```

Заметим, что отсечение `n mod k=0` является очень эффективным. Например, число 30000 имеет всего 50 делителей, и мы проверяем 50 длин подпоследовательностей B_i, а не 30000. Однако при больших ограничениях, например $N>10^6$ переборное решение может уже не успевать находить ответ за заданное время. Для таких случаев задача имеет более красивое решение. Запишем исходную последовательность A_i два раза подряд и выкинем из получившейся последовательности первое число.

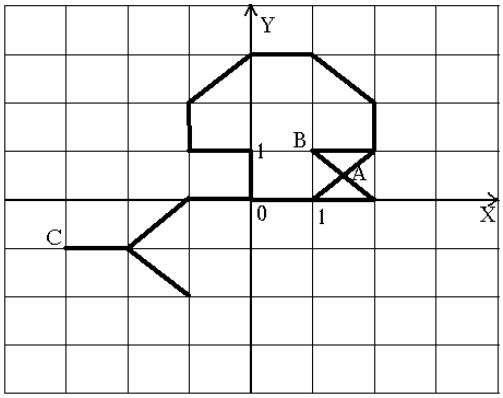


В построенной таким образом последовательности будем искать первое вхождение последовательности A_i. Индекс найденного вхождения и будет минимальной длиной последовательности B_i (подумайте, почему это так!) Заметим, что мы всегда найдем вхождение A_i, так как вторая половина построенной последовательности - это A_i.

Таким образом задача свелась к нахождению подстроки в строке. Известны алгоритмы поиска подстроки с линейным относительно длины строки временем исполнения, например алгоритм Кнута-Морриса-Пратта. Подробно об этом алгоритме можно прочитать, например, в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ», М.: МЦНМО, 2000.

Задача F Юный поджигатель

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	120 баллов



На клеточном поле введена система координат так, что центр координат находится в точке пересечения линий сетки и оси направлены вдоль линий сетки.

На этом поле выложили связную фигуру, состоящую из спичек. Использовались спички двух типов:

- Спички длины 1 выкладывались по сторонам клеток.
- Спички длины $\sqrt{2}$ выкладывались по диагоналям клеток.

Ребенок хочет сжечь фигуру. При этом он может поджечь ее в одной точке, имеющей целочисленные координаты (например, в точке A на рисунке поджигать фигуру нельзя, а в точках B и C — можно).

Известно, что огонь распространяется вдоль спички равномерно (но по каждой спичке — со своей скоростью). Спичка может гореть в нескольких местах (например, когда она загорается с двух концов; или когда в середине диагональной спички огонь перекидывается с одной спички на другую — огонь расползается по вновь подожженной спичке в обе стороны).

Напишите программу, которая определит, в какой точке нужно поджечь фигуру, чтобы она сгорела за минимальное время.

Формат входных данных

Во входном файле записано сначала число N — количество спичек ($1 \le N \le 40$). Затем идет N пятерок чисел вида X_1, Y_1, X_2, Y_2, T, задающих координаты концов спички и время ее сгорания при условии, что она будет подожжена с одного конца (гарантируется, что каждая спичка имеет длину 1 или $\sqrt{2}$, все спички образуют связную фигуру, и положение никаких двух спичек не совпадает). Все координаты — целые числа, по модулю не превышающие 200, время сгорания — натуральное число, не превышающее 10^7.

Формат выходных данных

Выведите координаты целочисленной точки, в которой нужно поджечь фигуру, чтобы она сгорела за наименьшее время, а затем время, за которое в этом случае фигура сгорит. Время должно быть выведено с точностью не менее 2-х знаков после десятичной точки. Если решений несколько, выведите любое из них.

Примеры

f.in	f.out
1 0 0 1 1 1	0 0 1.00
5 0 0 0 1 1 1 0 0 1 10 0 0 1 0 1 0 0 1 1 1 2 2 1 1 1	0 0 3.25
3 1 1 1 2 10 1 2 2 2 10 1 1 2 2 50	2 2 35.00
16 0 0 0 1 1 -2 -1 -3 -1 1 -2 -1 -1 0 1 -2 -1 -1 -2 1 -1 0 0 0 1 0 3 1 3 1 1 3 2 2 1 2 2 2 1 1 2 1 1 0 1 2 0 1 1 1 2 0 1 0 1 2 1 1 1 1	0 0 4.50

0 0 1 0 1	0 1 -1 1 1	-1 1 -1 2 1	0 3 -1 2 1	

Примечание

Частичные решения для случая, когда время сгорания каждой из спичек равно 1 (вне зависимости от ее длины), будут оцениваться приблизительно половиной баллов.

Решение

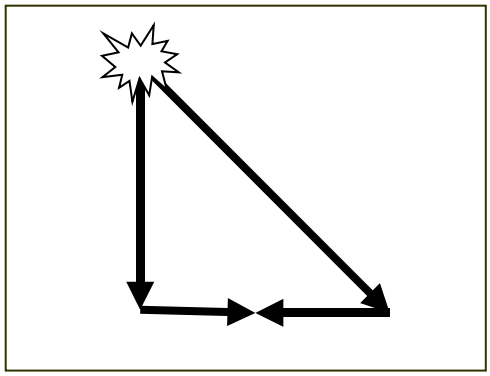
Прежде всего заметим, что спички могут пересекаться только концами, либо серединами. Других случаев пересечения быть не может. Таким образом, если разбить каждую спичку на две «половинки» вдвое меньшей длины, по полученные «полуспички» пересекаться будут только концами. Если все координаты исходных спичек умножить на два, то координаты всех «полуспичек» также будут выражаться целыми числами. Далее, если на два умножить также и время горения всех спичек, то время горения «полуспички» также будет выражаться целым числом секунд. Будем считать, что мы так уже поступили, и далее спичками именуются именно такие «полуспички».

Мы перешли к аналогичной задаче с вдвое большим количеством спичек с целыми координатами, пересекающихся только концами. Построим граф, ребрами которого будут спички, а вершинами – их концы. Задача сводится к нахождению такой вершины графа, при «поджигании» которой весь граф сгорает за минимальное время.

Будем решать задачу перебором по всем потенциальным «точкам поджигания». Так как в любом случае необходимо в выходной файл вывести кроме координат оптимальной точки общее время сгорания, задачу о нахождении времени сгорания графа при «поджигании» данной вершины тоже надо уметь решать.

Пусть нам дан граф, в котором на каждом ребре записано время сгорания соответствующей ему спички (эту величину будем называть весом или длиной ребра), и в нем зафиксирована некоторая вершина. Как найти время сгорания этого графа при «поджигании» этой вершины?

Ясно, что время сгорания графа равно расстоянию от зафиксированной вершины до наиболее удаленной от нее точки графа. Именно «наиболее удаленной точки», а не «наиболее удаленной вершины»! Простейший пример, где эти понятия различаются –треугольник:



Допустим, в приведенном выше примере времена горения вертикальной и диагональной спичек – одна секунда, а время горения горизонтальной спички – четыре секунды. Тогда при поджигании этой фигуры в верхней точке через секунду огонь достигнет обоих концов основания. Еще две секунды потребуется пламени, чтобы сжечь основание. Таким образом, хотя самая удаленная вершина находится на расстоянии 1 от точки поджигания, суммарное время сгорания такой фигуры равно трем секундам.

Вычислим кратчайшие расстояния от данной вершины до всех остальных. Кратчайшее расстояние соответствует моменту времени, когда огонь достигнет данной вершины. Для нахождения расстояний можно использовать, например, алгоритм Флойда.

Алгоритм Флойда находит кратчайшие расстояния между всеми парами вершин в графе, делая число операций, пропорциональное кубу числа вершин графа. Программа, реализующая алгоритм Флойда, выглядит следующим образом:

```
for k:=1 to N do
  for i:=1 to N do
    for j:=1 to N do
      if a[i,j]<a[i,k]+a[k,j] then a[i,j]:=a[i,k]+a[k,j];
```

Остановимся на описании алгоритма Флойда более подробно. Пусть в матрице A[i,j] записаны длины ребер графа (элемент A[i,j] равен весу ребра, соединяющего вершины с номерами i и j, если же такого ребра нет, то в соответствующем элементе записано некоторое очень большое число, например, 10^9). Построим новые матрицы $C_k[i,j]$ (k=0,…,N). Элемент матрицы $C_k[i,j]$ будет равен минимальной возможной длине такого пути из i в j, что в качестве промежуточных вершин в этом пути используются вершины с номерами от 1 до k. То есть рассматриваются пути, которые могут проходить через вершины с номерами от 1 до k (а могут и не проходить через какие-то из этих вершин), но заведомо не проходят через вершины с номерами от k+1 до N. В матрицу записывается длина кратчайшего из таких путей. Если таких путей не существует, записывается то же большое число, которым обозначается отсутствие ребра.

Сформулируем следующие факты.

В матрице $C_0[i,j]$ записаны длины путей, которые не содержат ни одной промежуточной вершины. Таким образом, матрица $C_0[i,j]$ совпадает с исходной матрицей A[i,j].

В матрице $C_N[i,j]$ записаны минимальные длины путей, которые в качестве промежуточных вершин используют все вершины графа — то есть длины кратчайших путей, которые мы хотим получить.

Если у нас уже вычислена матрица $C_{k-1}[i,j]$, то элементы матрицы $C_k[i,j]$ можно вычислить по следующей формуле: $C_k[i,j]:=\min(C_{k-1}[i,j], C_{k-1}[i,k]+C_{k-1}[k,j])$. В самом деле, рассмотрим кратчайший путь из вершины i в вершину j, который в качестве промежуточных вершин использует только вершины с номерами от 1 до k. Тогда возможно два случая:

Этот путь не проходит через вершину с номером k. Тогда его промежуточные вершины — это вершины с номерами от 1 до k–1. Но тогда длина этого пути уже вычислена в элементе $C_{k-1}[i,j]$.

Этот путь проходит через вершину с номером k. Но тогда его можно разбить на две части: сначала мы из вершины i доходим оптимальным образом до вершины k, используя в качестве промежуточных вершины с номерами от 1 до k–1 (длина такого оптимального пути вычислена в $C_{k-1}[i,k]$), а потом от вершины k идем в вершину j опять же оптимальным способом, и опять же используя в качестве промежуточных вершин только вершины с номерами от 1 до k ($C_{k-1}[k,j]$).

Выбирая из этих двух вариантов минимальный, получаем $C_k[i,j]$.

Последовательно вычисляя матрицы C_0, C_1, C_2 и т.д. мы и получим искомую матрицу C_N кратчайших расстояний между всеми парами вершин в графе.

Заметим теперь, что при вычислении очередной матрицы C_k нам нужны лишь элементы матрицы C_{k-1}, поэтому можно не хранить в памяти N таких матриц, а обойтись двумя — той, которую мы сейчас вычисляем, и той, которую мы вычислили на предыдущем шаге. На самом деле оказывается, что даже это излишне — все вычисления можно производить в одной матрице (подумайте, почему).

Вернемся к решению нашей задачи.

После нахождения кратчайших путей из «поджигаемой» вершины во все остальные, нам известно время, за которое огонь достигнет каждой из вершин. Теперь нужно проверить все ребра-спички на предмет того, сгорели ли они уже в процессе перемещения огня до вершин, а если нет, то нужно найти время, когда догорит данное ребро. Максимальное из этих времен даст ответ.

Пусть огонь достигает концов спички со временем сгорания L в моменты времени T1 и T2. Если T1 = T2+L или T2 = T1+L, то огонь передавался по этой спичке, и максимум из T1 и T2 и будет тем временем, к которому спичка сгорит полностью. Отметим, что разность между T1 и T2 не может быть больше L (подумайте, почему).

Пусть теперь разность между T1 и T2 не равна L. Это значит, что к разным концам спички огонь подошел разными путями, она будет гореть одновременно с обеих сторон и догорит где-то посередине. Напомним, что под спичкой мы понимаем половину спички, и пересекаться не в концах она уже ни с чем не может. То есть поджечь такая спичка никакую другую спичку не может – с обеих ее концов уже и так бушует пламя! В простейшем случае, если спичка подожжена одновременно с обоих концов, она сгорает за время L/2. Трудность в том, что в общем случае время возгорания концов спички может не совпадать.

Можно вычесть одно и то же число из T1 и из T2, т.е. мы можем перейти к задаче, где один конец загорается во момент времени 0, а второй – в момент времени T. Общее время сгорания спички в таком случае будет равно T + (L-T)/2. Прийти к этой формуле проще всего из следующих соображений. Пусть спичка имеет длину L сантиметров и горит со скоростью 1 сантиметр в секунду. Тогда первые T секунд она будет гореть с одного конца, и сгорит на T см., а оставшееся время потребуется на то, чтобы

сжечь L-T см со скоростью 2 см/сек, т.е. время сгорания этого куска спички будет равно (L-T)/2. При T = 0 формула дает ответ L/2, а при T = L - ответ L, что полностью согласуется с условием задачи.

Мы полностью умеем решать задачу о нахождении времени сгорания данной фигуры из спичек при ее поджигании в данной точке. Для этого нужно в соответствующем данной фигуре графе найти максимум из времен «догораний» каждого из ребер. И не забыть разделить его пополам – ведь при построении графа мы удвоили время сгорания каждой из спичек.

Теперь мы легко можем решить задачу перебором по вершинам. Проверив все потенциальные точки поджога и выбрав из них ту, при поджоге которой время сгорания минимально, мы найдем ответ. Необходимо учесть, что не каждая из вершин достроенного графа может быть точкой «поджигания». Так как мы раздвоили каждую спичку, в наш граф войдут также вершины, соответствующие серединам спичек. Из всех точек мы должны выбрать те, координаты которых нацело делятся на 2.

Еще одно замечание – в данной задаче мы всюду работали с целыми числами. Но в двух местах это целое число делилось на два – при нахождении координат пересечения спичек и при вычислении времени сгорания спички, подожженной с обеих сторон. Из этого следует, что общее время сгорания можно представить в виде X/4, где X – целое число. Соответственно, дробная часть этого времени будет равна 0.0, 0.25, 0.5 или 0.75, и двух десятичных знаков для ее представления вполне достаточно.

Приведем полный текст программы:

```
Program Matches;

Const
  TaskID='f';
  InFile=TaskID+'.in';
  OutFile=TaskID+'.out';

Const
  MaxN=42; { Ограничение на N }
  MaxG=2*MaxN+1; { Ограничение на число вершин в графе }
  Infinity=MaxLongInt; { "Бесконечное" расстояние }

Var
  N:Integer;                      {             }
  Match:Array[1..MaxN]Of Record   { Входные     }
    X1,Y1,X2,Y2:Integer;          {       данные }
    Time:LongInt;                 {             }
  End;                            {             }

  NG:Integer;                                          {       }
  Vertex:Array[1..MaxG]Of Record                       {       }
    X,Y:Integer;                                       { Граф }
  End;                                                 {       }
  Edge,Distance:Array[1..MaxG,1..MaxG]Of LongInt;      {       }

  Res:Extended;      { Минимальное время сгорания }
  ResX,ResY:Integer; { Оптимальная точка поджога  }

Procedure Load;
Var
  I:Integer;
Begin
  Assign(Input,InFile);
  ReSet(Input);
  Read(N);
  For I:=1 To N Do
    With Match[I] Do
      Read(X1,Y1,X2,Y2,Time);
  Close(Input);
End;
```

```
Function GetVertex(VX,VY:Integer):Integer;
  { Функция, возвращающая номер вершины с заданными координатами.
    При отсутствии нужной вершины она создаётся }
Var
  I:Integer;
Begin
  For I:=1 To NG Do
    With Vertex[I] Do
      If (X=VX) And (Y=VY) Then Begin
        GetVertex:=I;
        Exit;
      End;

  Inc(NG); { Если нужная вершина не найдена }
  With Vertex[NG] Do Begin
    X:=VX;
    Y:=VY;
    For I:=1 To NG-1 Do Begin
      Edge[I,NG]:=Infinity;
      Edge[NG,I]:=Infinity;
    End;
    Edge[NG,NG]:=0;
  End;
  GetVertex:=NG;
End;

Procedure AddEdge(X1,Y1,X2,Y2:Integer; Time:Longint);
  { Функция, добавляющая ребро между двумя точками }
Var
  A,B:Integer;
Begin
  A:=GetVertex(X1,Y1);
  B:=GetVertex(X2,Y2);
  Edge[A,B]:=Time;
  Edge[B,A]:=Time;
End;

Procedure BuildGraph; { Процедура построения графа }
Var
  I:Integer;
Begin
  NG:=0;
  For I:=1 To N Do
    With Match[I] Do Begin
      AddEdge(X1*2,Y1*2,X1+X2,Y1+Y2,Time);
      AddEdge(X1+X2,Y1+Y2,X2*2,Y2*2,Time);
    End;
End;

Procedure FindShortestPaths;
Var
  K,I,J:Integer;
Begin
  Distance:=Edge;
  For K:=1 To NG Do
    For I:=1 To NG Do If Distance[I,K]<Infinity Then
      For J:=1 To NG Do If Distance[K,J]<Infinity Then
        If Distance[I,K]+Distance[K,J]<Distance[I,J] Then
          Distance[I,J]:=Distance[I,K]+Distance[K,J];
End;
```

```
    Function BurnAt(At:Integer):Extended;
      { Функция, вычисляющая время сгорания при поджоге в точке At }
    Var
      I,J:Integer;
      Cur,ThisEdge:Extended;
    Begin
      Cur:=0;
      For I:=1 To NG Do If Distance[At,I]>Cur Then Cur:=Distance[At,I];
      For I:=1 To NG Do
        For J:=I+1 To NG Do If Edge[I,J]<Infinity Then Begin
          If (Distance[At,I]<Distance[At,J]+Edge[I,J]) And
             (Distance[At,J]<Distance[At,I]+Edge[I,J]) Then Begin
            If Distance[At,I]<Distance[At,J] Then
                              ThisEdge:=Distance[At,J]+(Edge[I,J]-(Distance[At,J]-
Distance[At,I]))/2
            Else
                              ThisEdge:=Distance[At,I]+(Edge[I,J]-(Distance[At,I]-
Distance[At,J]))/2;
            If ThisEdge>Cur Then Cur:=ThisEdge;
          End;
        End;
      BurnAt:=Cur;
    End;

    Procedure Solve;
    Var
      I:Integer;
      Cur:Extended;
    Begin
      Res:=Infinity;
      For I:=1 To NG Do
        With Vertex[I] Do
          If Not Odd(X) And Not Odd(Y) Then Begin
            Cur:=BurnAt(I);
            If Cur<Res Then Begin
              Res:=Cur;
              ResX:=X Div 2;
              ResY:=Y Div 2;
            End;
          End;
    End;

    Procedure Save;
    Begin
      Assign(Output,OutFile);
      ReWrite(Output);
      WriteLn(ResX,' ',ResY);
      WriteLn(Res/2:0:2);
      Close(Output);
    End;

    Begin
      Load;
      BuildGraph;
      FindShortestPaths;
      Solve;
      Save;
    End.
```

Задача G Реклама

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

В супермаркете решили время от времени транслировать рекламу новых товаров. Для того, чтобы составить оптимальное расписание трансляции рекламы, руководство супермаркета провело следующее исследование: в течение дня для каждого покупателя, посетившего супермаркет, было зафиксировано время, когда он пришел в супермаркет, и когда он из него ушел.

Менеджер по рекламе предположил, что такое расписание прихода-ухода покупателей сохранится и в последующие дни. Он хочет составить расписание трансляции рекламных роликов, чтобы каждый покупатель услышал не меньше двух рекламных объявлений. В то же время он выдвинул условие, чтобы два рекламных объявления не транслировались одновременно и, поскольку продавцам все время приходится выслушивать эту рекламу, общее число рекламных объявлений за день было минимальным.

Напишите программу, которая составит такое расписание трансляции рекламных роликов. Рекламные объявления можно начинать транслировать только в целые моменты времени. Считается, что каждое рекламное объявление заканчивается до наступления следующего целого момента времени. Если рекламное объявление транслируется в тот момент времени, когда покупатель входит в супермаркет или уходит из него, покупатель это объявление услышать успевает.

Формат входных данных

Во входном файле записано сначала число N — количество покупателей, посетивших супермаркет за день($1 \le N \le 3000$). Затем идет N пар натуральных чисел A_i, B_i, задающих соответственно время прихода и время ухода покупателей из супермаркета ($0 < A_i < B_i < 10^6$).

Формат выходных данных

В выходной файл выведите сначала количество рекламных объявлений, которое будет протранслировано за день. Затем выведите в возрастающем порядке моменты времени, в которые нужно транслировать рекламные объявления.

Если решений несколько, выведите любое из них.

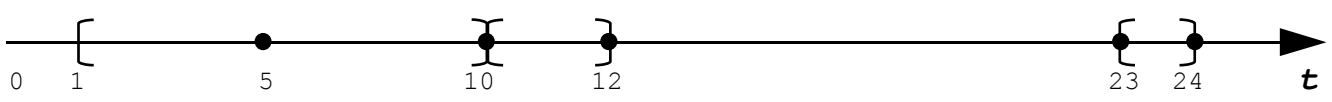
Пример

g.in	g.out
5 1 10 10 12 1 10 1 10 23 24	5 5 10 12 23 24

Решение

При решении задачи всегда очень полезно представить какую-либо ее визуальную интерпретацию. Отобразим все время работы магазина временной осью, а время прихода и ухода покупателей – отрезками на этой оси. Теперь задачу можно переформулировать: поставить на оси минимальное количество точек с целочисленными координатами так, чтобы в каждом отрезке содержалось не менее двух точек.

Пример из условия в такой интерпретации будет выглядеть следующим образом:



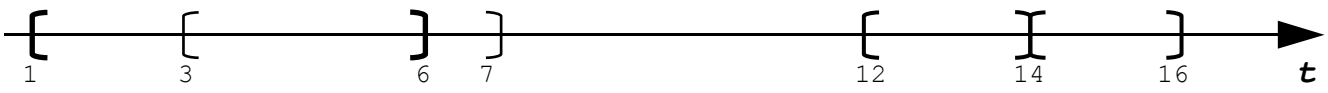
В программе будем использовать следующие типы данных:

```
const
   MaxN=3000;
   Infinity=MaxLongInt; {“бесконечная” координата}

type
  TSegment = record
    left, right : longint;
  end;

var
  n : longint;                                {количество отрезков}
  segment : array [1..MaxN] Of TSegment;      {координаты отрезков}
  point : array [1..MaxN*2] Of longint;       {точки, которые мы расставляем}
```

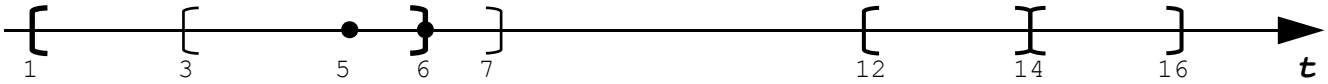
Отсортируем все отрезки по возрастанию правых границ, а при их равенстве – по убыванию левых.

Ограничение на количество отрезков ($N \le 3000$) позволяет применить не только быструю сортировку, но и алгоритм сортировки с квадратичной временной сложностью, например метод «пузырька».

Перейдем теперь к основной части решения. Для этого рассмотрим следующий пример. Пусть дано четыре отсортированных отрезка: [1,6], [3,7], [12,14], [14,16].



В самом первом отрезке [1,6] должны содержаться две точки. Их необходимо поставить как можно правее, то есть в точках с целочисленными координатами 5 и 6. Почему? Поскольку это отрезок с наименьшей правой границей, то раньше него другие отрезки не могли закончиться. Но могли начаться другие отрезки! А чем правее мы располагаем точки, тем больше шанс, что они одновременно попадут и в другие отрезки – что нам выгодно. Еще раз обратим внимание на факт, что не существует более выгодной расстановки точек, чем данная: поскольку никакой другой отрезок раньше наших точек не заканчивается, то мы не упускаем ни одну потенциальную возможность улучшить расстановку точек.



В нашем примере расставленные две точки сразу попали и в отрезок [3,7]. А вот если бы мы поставили точки, например, в координатах 1 и 2, то такая расстановка была бы неэффективной – мы покрыли бы ей только один отрезок, а не два.

Назовем точку 6 последней расставленной точкой, а точку 5 – предпоследней. В программе это запишется следующим образом:

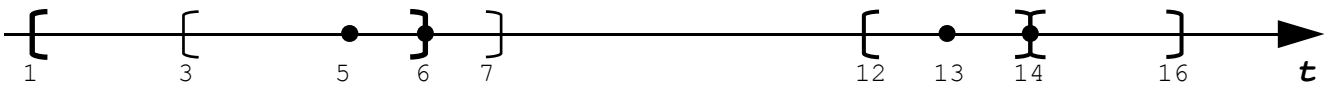
```
PrevLast := right-1;
Last := right;
```

где `right` – правая граница текущего отрезка.

Переходим к следующему отрезку [3,7] – но внутри него уже стоят две точки, поскольку левая граница отрезка меньше, чем предпоследняя расставленная точка.

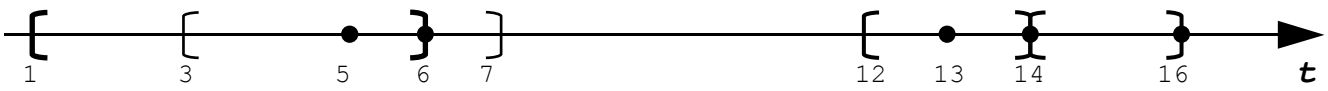
```
if left <= PrevLast then <ничего расставлять не нужно>;
```

Следующий отрезок – [12, 14]. В нем не стоит еще ни одной точки, так как `left > last`. Из аналогичных соображений ставим в нем две точки как можно правее.



Последний отрезок – [14, 16]. В нем уже содержится одна из поставленных точек, так как `Last = left`. Ставим еще одну точку в правую границу отрезка – координата равна 16. При этом предыдущей точкой станет точка 14.

```
PrevLast := Last;
Last := right;
```



Легко видеть, что такой алгоритм расстановки точек всегда дает оптимальный результат. Итак, для каждого отрезка мы смотрим, нужно ли поставить в нем одну или две точки и если да, то ставим их как можно правее. Резюмируя все вышесказанное приведем ключевой фрагмент программного кода, реализующий данную логику:

```
procedure solve;
var
  Last, PrevLast : longint; {две последние поставленные точки}
  i : longint;
begin
  res := 0; {количество поставленных точек}
  Last := -Infinity;
  PrevLast := -Infinity;
  for i := 1 to N do
    with segment[i] do
      if last < left then   {необходимо поставить ещё две точки}
        begin
            inc(res);
            point[res] := right-1;
            inc(res);
            point[res] := right;
            PrevLast := right-1;
            Last := right;
        end
       else
        if PrevLast < left then   {необходимо поставить ещё одну точку}
          begin
              inc(res);
              point[res] := right;
              PrevLast := Last;
              Last := right;
          end;
end;
```

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