Станкевич Андрей Сергеевич

РАЗБОР ЗАДАЧ ВТОРОЙ ВСЕРОССИЙСКОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ПРОГРАММИРОВАНИЮ

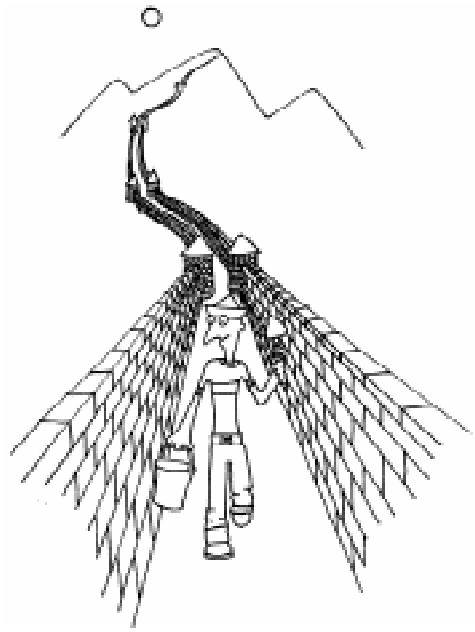
Задача A. Две стены.

У Пети есть набор из N кирпичиков. Каждый кирпичик полностью окрашен в один из K цветов, i-й кирпичик имеет размер $1\times1\times L_i$.

Петя знает, что он может построить из кирпичиков прямоугольную стену толщиной 1 и высотой K, причем первый горизонтальный слой кирпичиков в стене будет первого цвета, второй – второго и т. д. Теперь Петя хочет узнать, может ли он из своего набора построить две прямоугольные стены, обладающие тем же свойством. Помогите ему выяснить это.

Формат входных данных

На первой строке входного файла находятся числа N и K ($1 \le N \le 5000$, $1 \le K \le 100$). Следующие N строк содержат описание Петиных кирпичиков: сначала длина L_i, затем номер цвета C_i ($1 \le L_i \le 100$, $1 \le C_i \le K$). Известно, что у Пети не более 50 кирпичиков каждого цвета.



Формат выходных данных

Выведите на первой строке выходного файла YES, если Петя сможет построить из своих кирпичиков две прямоугольные стены высоты K, j-й слой кирпичиков в каждой из которых будет j-го цвета, и NO – в противном случае. В случае положительного ответа выведите на второй строке в произвольном порядке номера кирпичиков, из которых следует построить первую стену (кирпичики нумеруются в том порядке, в котором они заданы во входном файле, начиная с 1). Если решений несколько, можно выдать любое из них.

Примеры

input.txt	output.txt
5 2 2 1 2 1 2 1 3 2 3 2	NO
2 1 1 1 3 1	YES 1
9 3 7 3 1 1 3 2 2 1 3 2 3 1 4 2 4 1 3 3	YES 2 3 4 9

Решение.

Для решения этой задачи применим динамическое программирование. Пусть сумма длин всех кирпичиков каждого из K цветов равна W (для всех j = 1, 2, ..., K выполняется равенство $\sum_{i:C_i=j} L_i = W$). Поскольку Петя может построить прямоугольную стену, эта сумма для всех цветов одинакова и наше определение имеет смысл. Обозначим для удобства множество номеров кирпичиков цвета j через Z_j.

Заметим, что, для того чтобы Петя мог построить две стены, необходимо и достаточно, чтобы существовало $1 \le V < W$ такое, что для каждого цвета j существовал набор кирпичиков j-го цвета таких, чтобы сумма их длин была равна V, то есть для всех j = 1, 2, ..., K существовало множество $U_j \subset Z_j$, такое что $\sum_{i \in U_j} L_i = V$.

Отметим, что если зафиксировать какое-либо V, то задача о существовании такого множества для конкретного множества Z_j эквивалентна одной из формулировок известной задачи о рюкзаке (можно ли набрать вес V с помощью камней с весами w_i, можно использовать не все камни). Эта одна из известных задач, эффективного решения которой в случае произвольных (не ограниченных сверху) весов w_i неизвестно. Однако в случае, когда веса ограничены (как в нашем случае, когда $L_i \le 100$), эта задача решается с помощью динамического программирования.

Рассмотрим для начала только кирпичики 1-го цвета. Условно перенумеруем их заново от 1 до N_1, где N_1 – количество кирпичиков 1-го цвета. Рассмотрим двумерный массив B, пусть $B[i][j]$ содержит 1, если можно, используя некоторое подмножество первых i кирпичиков, построить стену длиной j, и 0, если этого сделать нельзя. Тогда рекуррентная формула для $B[i][j]$ имеет вид:

$$B[i][j] = \begin{cases} 1, \textit{если } i = 0, j = 0, \\ 0, \textit{если } i = 0, j > 0, \\ B[i-1][j], \textit{если } i > 0, j < L_i, \\ \max(B[i-1][j], B[i-1][j-L_i], \\ \qquad \textit{если } i > 0, j \ge L_i. \end{cases}$$

Отметим, что если заполнять массив B по возрастанию первого индекса, то для очередного элемента используется только элемент предыдущей строки, причем стоящий в ряду с номером не больше текущего, поэтому можно сэкономить память, используя только одну строку и заполняя ее с конца. Предполагая, что данные прочитаны в массив L, и $L[i][j]$ – длина j-го кирпичика цвета i, получаем следующий фрагмент программы, который заполняет массив B для i-го цвета:

```
fillchar(b, sizeof(b), false);
b[0] := true;
for j := 1 to m[i] do
begin
    for t := w downto l[i][j] do
        if b[t - l[i][j]] then
            b[t] := true;
end;
```

Отметим, что если бы мы решали задачу о рюкзаке, то есть для конкретного V хотели бы проверить, что ряд такой длины можно получить из кирпичиков, то ответом было бы $B[M_i][V]$ или, поскольку в нашей программе всегда существует только последняя строка, $B[V]$. Но, поскольку мы не знаем V, так как оно должно быть одним для всех K цветов, то логично сохранить все возможные длины ряда каждого цвета для последующего использования. Более того, поскольку нам надо найти длину ряда, которая бы подошла для *всех* цветов, то достаточно иметь всего один одномерный массив *Can*, который будет содержать true для тех V, для которых можно построить ряд кирпичиков длиной V для всех обработанных цветов. Тогда если после окончания обработки всех цветов найдется $1 \le V < W$ такое, что *Can*[V] = true, то задача имеет ре-

шение, Петя сможет построить две стены – одну $K \times V$ и другую $K \times (W-V)$.

Приведем фрагмент программы, который реализует заполнение массива *Can*:

```
fillchar(can, sizeof(can), true);
for i := 1 to k do
begin
     здесь фрагмент, заполняющий b
     for j := 1 to w do
          can[j] := can[j] and b[j];
end;
```

Таким образом, мы научились отвечать на первый вопрос задачи – можно ли построить две стены:

```
f := false;
for i := 1 to w - 1 do
     if can[i] then
     begin
          f := true;
          v := i;
          break;
     end;

if f then
begin
     writeln("YES");
end else begin
     writeln("NO");
end;
```

Теперь не представляет труда восстановить, из каких кирпичиков надо построить стену. Для этого следует использовать так называемые ссылки назад, то есть для каждого размера слоя j хранить, какой кирпич надо положить последним, чтобы получить такую длину слоя. Оставим конкретную реализацию в качестве упражнения.

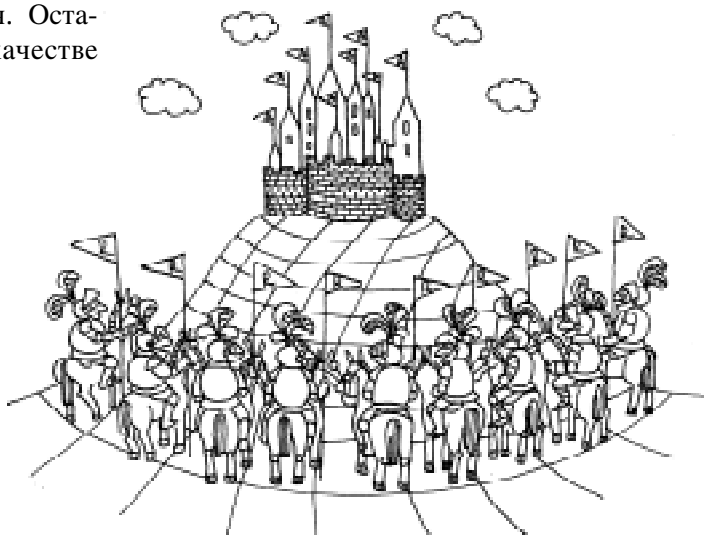
Задача B. Блокада.

Государство Флатландия представляет собой прямоугольник размером $M \times N$, состоящий из единичных квадратиков. Флатландия разделена на K провинций ($2 \le K \le 100$). Каждая провинция представляет собой связное множество квадратиков, то есть из каждой точки провинции можно дойти до любой другой ее точки, при этом разрешается переходить с квадратика на квадратик, только если они имеют общую сторону (общей вершины недостаточно). Во Флатландии нет точки, которая граничила бы более чем с тремя провинциями (то есть четыре квадратика, имеющие общую вершину, не могут принадлежать четырем разным провинциям).

Каждая провинция имеет свой символ. Столица Флатландии находится в провинции, которой принадлежит символ A (заглавная латинская буква A). Провинция называется пограничной, если она содержит граничные квадратики. Провинция, в которой находится столица Флатландии, не является пограничной.

Король соседнего с Флатландией королевства Ректилания решил завоевать Флатландию. Для этого он хочет захватить столицу Флатландии. Однако он знает, что сил его армии недостаточно, чтобы сделать это сразу. Поэтому сначала он хочет окружить столичную провинцию, чтобы ослабить силы противника долгой блокадой, а потом захватить столицу.

Чтобы окружить провинцию, требуется захватить все провинции, с которыми она граничит. Две провинции граничат, если существует два квадратика, имеющие общую сторону, один из которых принадлежит первой из них, а другой – второй. Чтобы захватить провинцию, нужно, чтобы выполнялось одно из двух условий: либо она пограничная, либо гра-



ничит с какой-либо уже захваченной провинцией.

Чтобы сберечь силы своей армии, король Ректилании хочет установить блокаду столичной провинции, захватив как можно меньше провинций. Помогите ему выяснить, сколько провинций требуется захватить. Захватывать столичную провинцию нельзя, поскольку для этого сил армии Ректилании пока недостаточно.

Формат входных данных

Первая строка содержит M и N ($3 \le M, N \le 200$). Следующие M строк содержат N символов каждая и задают карту Флатландии. Символ, находящийся в (i+1)-й строке входного файла на j-м месте, представляет собой символ провинции, которой принадлежит квадратик (i, j). Все символы имеют ASCII-код больше 32 (пробела).

Формат выходных данных

Выведите в выходной файл единственное число – количество провинций, которые требуется захватить. Если установить блокаду невозможно, выведите «–1».

Примеры

input.txt	output.txt
5 6 BBBBBZ BCCCBZ BCAbbZ BDDDbZ 3333Z	4

Решение.

Для решения представим Флатландию в виде неориентированного графа $G=<V,E>$, вершинами которого будут провинции, и две провинции соединим ребром, если они граничат (определение графа и некоторые связанные определения можно найти в решении задачи F).

Заметим теперь, что наша задача – найти наименьшее по мощности (то есть по количеству элементов) связное множество вершин такое, что в него входят все вершины, соседние с вершиной, соответствующей столичной провинции, и хотя бы одна вершина, отвечающая граничной провинции, но не будет входить столичная вершина. Напомним, что множество вершин V называется связным, если существует путь из любой вершины $u \in V$ в любую другую его вершину $v \in V$, проходящий только по вершинам этого множества. Максимальное по включению связное множество графа называется компонентой связности.

Для начала рассмотрим случай, когда этого нельзя сделать. Обозначим вершину, отвечающую столичной провинции, v_0. Удалим из графа эту вершину и рассмотрим получившийся граф:

$$G' = < V' = V \setminus \{v_0\}, E' >,$$

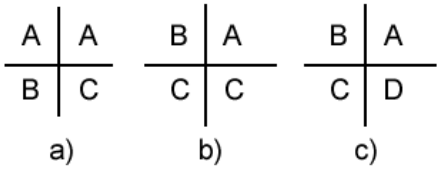
где $E' = E|_{V' \times V'}$.

Предложение. Если граф G' несвязен, то построить искомое множество нельзя.

Доказательство. Действительно, исходный граф связен. Поскольку столичная провинция не является граничной, то все граничные вершины лежат в одной компоненте связности графа G' (мы можем добраться из одной в другую, следуя по граничным квадратикам). Значит, существует некоторое множество вершин, до которых нельзя добраться от граничных, не проходя через столичную вершину. Назовем такие вершины плохими. Обязательно найдется хотя бы одна плохая вершина, соседняя со столичной в исходном графе. Последнее утверждение можно доказать так: возьмем вершину v_{bad} из этого множества и какую-нибудь граничную вершину v_{border}, в исходном графе между ними существовал путь $v_{border}, v_1, v_2, ..., v_0, v_k, ..., v_{bad}$, проходящий через столичную вершину. Тогда следующая за последним появлением v_0 в этом пути вершина (v_k в нашем случае) будет плохой и соседней со столичной. Значит, искомое множество построить нельзя.

Теперь перейдем ко второму случаю, когда граф G' связен. В этом случае хотя бы одно множество, отвечающее нашим требованиям, существует (все множество V'). В дальнейшем тексте будем полагать, что G' связен.

Отметим следующее свойство вершин, соседних со столичной вершиной: множество вершин, соседних со столичной вершиной связно. Не приводя полностью формального доказательства этого факта, чтобы не запутывать изложение, рассмотрим рисунок, поясняющий его:



В случае, если переход от одной соседней провинции к другой осуществляется, как в случаях a) и b), связность не нарушается, а случай c) невозможен по условию.

Итак, чтобы множество вершин удовлетворяло нашим условиям, достаточно, чтобы оно содержало все соседние со столичной провинции, граничную провинцию и в нем существовал путь из граничной провинции в какую-нибудь соседнюю со столичной. Значит, ответом является $\min_{x\in N, y\in B} dist(x,y)+|N|$, где N – множество вершин, соседних со столичной, B – множество граничных вершин и $dist(x,y)$ – длина кратчайшего пути от вершины x до вершины y.

Таким образом, осталось привести способ нахождения длины кратчайшего пути от вершины x до вершины y для всех вершин $x \in N$ и $y \in B$. Для этого будем использовать алгоритм Флойда-Уоршелла, который находит длины кратчайших путей между всеми парами вершин. Алгоритм работает следующим образом: в двумерном массиве a элемент $a[i][j]$ после k шагов алгоритма содержит длину кратчайшего пути из i-й вершины в j-ю, проходящего только по вершинам из множества $M_k = \{1,2,...,k\}$. Соответственно, k-й шаг алгоритма состоит в добавлении в это множество вершины k и пересчета матрицы a. Ниже приведена реализация алгоритма для графа с n вершинами, изначально a инициализируется матрицей смежности.

```
for k := 1 to n do
 for i := 1 to n do
  for j := 1 to n do
   if ((a[i][k] <> 0) and
    (a[k][j] <> 0)) and
    ((a[i][j] = 0) or
    (a[i][k] + a[k][j] < a[i][j]))
   then
    a[i][j] := a[i][k] + a[k][j];
```

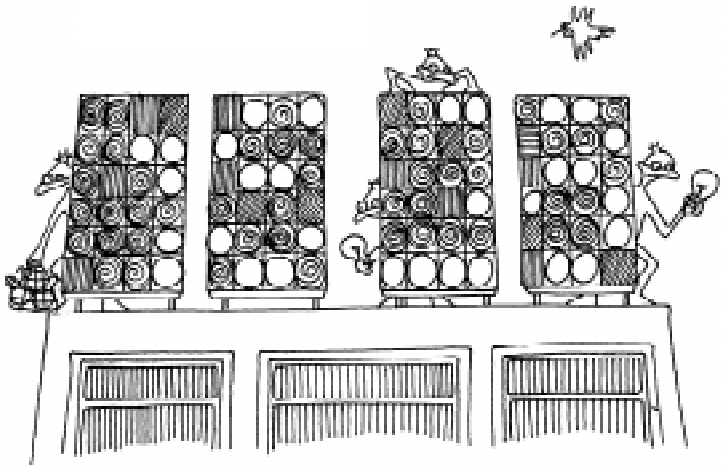
Программа.

Ключевые фрагменты программы приведены на дискете.

Задача C. Электронные часы.

Циферблат новых электронных часов, установленных на главном здании офиса фирмы Macrohard, состоит из 4 прямоугольных панелей, каждая из которых состоит из 6 рядов по 5 лампочек в каждом. Первые две панели отображают цифры, из которых складываются часы, а следующие две – минуты. (Если сейчас меньше 10 часов, первая панель отображает 0).

К сожалению, лампочки, установленные на панелях, были произведены компанией Sveta.Net, которая известна своим принципом «раньше перегорит – больше спрос», вследствие чего на следующий



день люди, проходя мимо офиса компании, видели лишь некоторое подобие цифр, поскольку некоторые лампочки больше не горели.

Петя живет в доме, стоящем прямо напротив офиса компании Macrohard. В первый день после установки часов он зарисовал у себя в блокноте, как выглядят все цифры на панелях (панели однотипные, поэтому одна и та же цифра на различных панелях выглядит одинаково). Теперь Петя хочет узнать, можно ли по текущему изображению на часах однозначно определить, сколько сейчас времени. Помогите ему!

Формат входных данных

При тестировании этой задачи в каталоге, который будет текущим, когда будет запущена Ваша программа, будут находиться два файла. Файл digits.txt содержит 6 строк по 50 символов в каждой. Он будет одинаковым для всех тестов и будет совпадать с приведенным в примере. Вы также можете найти этот файл в каталоге o:\common. Содержимое файла digits.txt задает правильное написание цифр на панелях (первый прямоугольник 6×5 символов задает число 0, следующий – 1 и т. д. до 9). Негорящая лампочка обозначается символом «.» (точка), а горящая – «#» (диез).

Входной файл input.txt содержит 6 строк по 20 символов в каждой – текущее изображение на часах. Первый прямоугольник 6×5 задает первую панель, следующий – вторую, следующий – третью и последний – четвертую.

<table>
<tr><th colspan="2">digits.txt</th></tr>
<tr><td colspan="2">..##.....#..##..####.#..#.####..###.####..##...##.
.#..#...##.#..#....#.#..#.#....#.......#.#..#.#..#
.#..#..#.#....#...#..#..#.###..###....#...##..#..#
.#..#....#...#.....#.####....#.#..#..#...#..#..###
.#..#....#..#......#....#.#..#.#..#..#...#..#....#
..##.....#.####.###.....#..##...##...#....##..###.</td></tr>
<tr><th>input.txt</th><th>output.txt</th></tr>
<tr><td>..##..####.#..#.####
....#....#.........
....#...#..#..#.###.
...#.....#.##.....#
..#......#....#.#..#
.####..##.....#..#..</td><td>23:45</td></tr>
<tr><td>#.##.....#..##..####
.#..#...##.#..#....#
.#.....#......#...#.
.#..#.............#
.#..#.......#......#
..##.....#.####.....</td><td>ERROR</td></tr>
<tr><td>..##........##..####
.#..#....#.#..#....#
.#...........#...#.
.#..#.............#
.#..#.......#......#
..##.......####.....</td><td>AMBIGUITY</td></tr>
</table>

Таблица 1

Формат выходных данных

Если можно точно определить время, которое сейчас отображается на часах, выведите это время в формате hh:mm. Если время нельзя определить однозначно, выведите AMBIGUITY. Если же в часах точно сломалось еще что-то, например, центральный процессор, который управляет лампочками, выведите ERROR.

Примеры приведены в таблице 1.

Решение.

Введем понятие «символ» как массив размера 6x5, в котором 1 соответствует горящей лампочке, а 0 – негорящей. Для двух символов A и B введем понятие их пересечения как массив $C=A*B$, где $c[i][j] = a[i][j]*b[i][j]$. Тогда верно следующее утверждение: чтобы символ A мог быть цифрой i, должно выполняться $A*D_i=A$, где D_i – символ, соответствующий каноническому написанию цифры i.

Помня также, что время лежит в диапазоне 00:00–23:59, получаем следующий алгоритм решения задачи: для каждого возможного времени проверить, может ли оно быть отображено сейчас на табло (для этого каждую цифру на табло надо сравнить с соответствующей канонической цифрой). После этого: если ровно одно время может быть отображено на табло, то это и есть ответ, если более чем одно, то ответ AMBIGUITY, а если ни одно время не подошло, то ответ ERROR.

Программа.

Приведем процедуру проверки того, что символ a может быть цифрой, содержащейся в символе b (в реализации для простоты не будем заменять . на 0, а # на 1, а оставим как есть):

```
type
    tdigit = array [1..6, 1..5] of char;

function check(a, b: tdigit): boolean;
var
    i, j: longint;
begin
    check := false;
    for i := 1 to 6 do
        for j := 1 to 5 do
            if (a[i][j] = ".") and
            (b[i][j] = "#") then
                exit;
    check := true;
end;
```

Теперь проверяем, какое время может быть на табло: часы и минуты, и выводим ответ:

```
for i := 0 to 23 do
begin
    if can[1][i div 10] and
    can[2][i mod 10] then
    begin
        inc(ch);
        hh := i;
    end;
end;

cm := 0;
for i := 0 to 59 do
begin
    if can[3][i div 10] and
    can[4][i mod 10] then
    begin
        inc(cm);
        mm := i;
    end;
end;

if (ch = 0) or (cm = 0) then
begin
   writeln("ERROR");
end
else if (ch = 1) and (cm = 1) then
begin
  writeln(hh div 10, hh mod 10, ":",
      mm div 10, mm mod 10);
end else begin
  writeln("AMBIGUITY");
end;
```

Задача D. Сравнение URL.

Для идентификации ресурсов в сети Internet используются URL (Uniform Resource Locator). URL состоит из нескольких элементов: *протокол*, *хост*, *порт*, *путь*, *файл* и *секция*. Некоторые элементы URL могут быть опущены. Рассмотрим упрощенный формат URL:

[протокол://]хост[:порт][путь/[файл[#секция]]]

Заключенные в квадратные скобки элементы могут быть опущены, то есть, например, можно не указать протокол или секцию. Но, если указан, например, файл, то обязательно должен быть указан путь. Регистр букв в элементах URL не важен.

Рассмотрим кратко все элементы URL.

Протокол – это способ доступа к файлу, URL с разными протоколами и одинаковыми остальными элементами могут указывать на различные ресурсы.

Хост и *порт* – это имя некоторого сервера в сети и способ доступа к нему (порт – натуральное число, не превосходящее 65535).

Путь представляет собой путь к файлу, содержащему запрашиваемый ресурс, от некоторого каталога на сервере, который называется *корневым*. При этом для разделения имен каталогов используется символ «/». Путь, если он не пуст, всегда начинается с символа «/». Специальное обозначение «.» соответствует самому каталогу, «..» – родительскому каталогу.

Файл – это файл, содержащий запрашиваемый ресурс.

Наконец, файл может быть разбит на *секции* каким-либо способом, и можно указать, к какой именно секции вы хотите обратиться.

Различные символы в URL могут быть заменены своими шестнадцатеричными ASCII-кодами с помощью символа %, например, a = %41, Z = %5A. В коде всегда используется ровно две шестнадцатеричные цифры.

Некоторые символы могут встречаться в элементах URL только как шестнадцатеричные коды – все символы, кроме букв латинского алфавита, цифр и символов «.» и «–», а некоторые не могут встречаться вообще: «\», «#», «*», «@», «%», «?», «:», «,», а также символы с ASCII-кодом меньше %20. Символ «/» может встречаться в элементах URL только в пути для разделения входящих в него каталогов. Имя файла не может состоять только из точек.

Рассмотрим примеры URL:

http://neerc.ifmo.ru/school

ftp://somewhere.net:1234/pub/files/coolgame.zip

nobody.nowhere.net/some%20dir/some%20file#some%20info

Ваша цель в этой задаче – помочь разработчикам web-сервера. Для web-сервера отсутствующие части URL имеют следующие значения по умолчанию:

Протокол	http
Порт	80
Путь	*пустая строка*
Файл	index.html
Секция	*пустая строка*

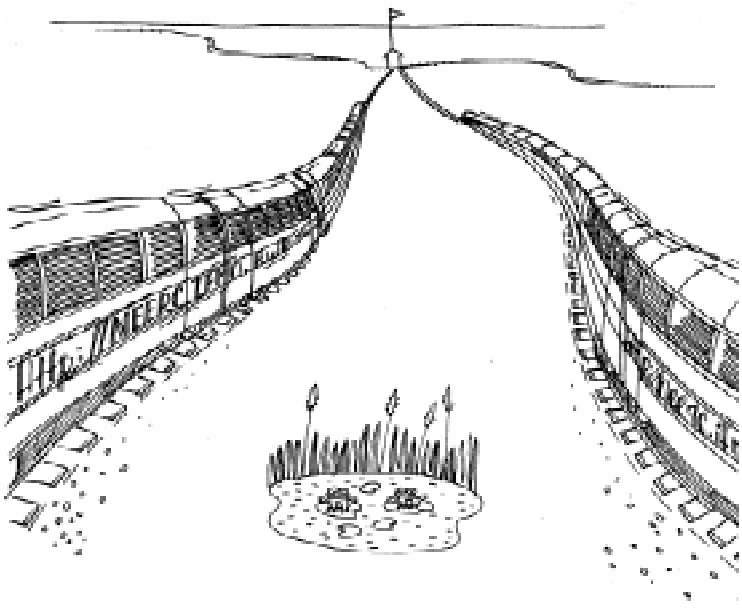
Различные как строки URL могут указывать на один и тот же ресурс, например, следующие три URL:

neerc.ifmo.ru

http://neerc.ifmo.ru:80/index.html#

Http://NEERC.IFMO.Ru/Dir/.././.

Для разграничения доступа к ресурсам необходимо уметь определять, указывают ли два различных URL на один и тот же ресурс. Помогите разработчикам написать соответствующую проверку.



Формат входных данных

Входной файл состоит из двух строк, каждая из них содержит URL. Оба URL удовлетворяют формату, приведенному в условии этой задачи. Длина каждого URL не превосходит 200 символов. Гарантируется, что ни один из промежуточных каталогов на пути к ресурсу не лежит выше корневого каталога (то есть не может встретиться, например, URL http://somewhere.com/../dir/index.html), а также, что имена всех каталогов состоят, по крайней мере, из одного символа (два символа «/» не могут идти подряд в любом месте, кроме как непосредственно после двоеточия после имени протокола).

Формат выходных данных

Выведите YES в выходной файл, если оба URL, приведенные во входном файле, указывают на один и тот же ресурс, и NO – в противном случае.

Примеры

input.txt	output.txt
http://neerc.ifmo.ru:80/index.html# neerc.ifmo.ru	YES
neerc.ifmo.ru/dir/../school NEERC.IFMO.RU/./SCHOOL	YES
neerc.ifmo.ru %6E%65%65%72%63%2E%69 %66%6D%6F%2E%72%75	YES
nowhere.com somewhere.com	NO
http://neerc.ifmo.ru ftp://neerc.ifmo.ru	NO
neerc.ifmo.ru/index.htm neerc.ifmo.ru/index.html	NO
neerc.ifmo.ru:80 neerc.ifmo.ru:8080/index.html	NO
http://somewhere.com somewhere.com/index.html#a	NO

Решение и программа.

Решение этой задачи представляет собой исключительно техническую проблему. Постепенно разделив заданные URL на части, составим из них URL в канонической форме (то есть такой, где имеются все части, все латинские буквы находятся в верхнем регистре и не встречается каталогов .. и .) и затем сравним их как строки.

Функция, приводящая URL к канонической форме, приведена на дискете.

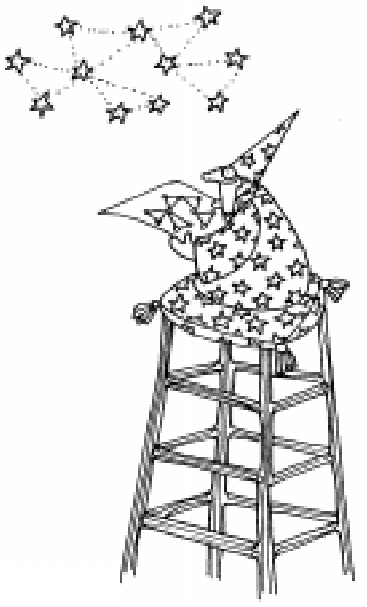
Задача E. Триангуляция Делоне.

Триангуляцией некоторого набора точек на плоскости называется набор невырожденных треугольников, удовлетворяющий следующим свойствам:

1. Вершинами треугольников являются только точки исходного набора. Каждая точка исходного набора является вершиной хотя бы одного треугольника.

2. Два различных треугольника либо не имеют общих точек, либо имеют общую вершину, либо имеют общую сторону (но площадь их пересечения всегда равна 0).

3. Любая точка, лежащая внутри выпуклой оболочки исходного набора точек, принадлежит хотя бы одному треугольнику (она может принадлежать нескольким треугольникам, если является их общей вершиной или принадлежит их общей стороне). (Выпуклой оболочкой некоторого набора точек называется наименьший выпуклый многоугольник, содержащий все эти точки).



Триангуляция называется триангуляцией Делоне, если, кроме того, для нее выполняется следующее условие:

4. Внутри окружности, описанной около любого треугольника из триангуляции, не лежит ни одна из исходных точек (точки могут лежать на окружности, в частности, на ней, очевидно, лежат вершины рассматриваемого треугольника).

Для заданного набора точек найдите количество его триангуляций Делоне (две триангуляции считаются различными, если они отличаются хотя бы одним треугольником).

Формат входных данных

На первой строке входного файла находится число N – количество точек ($3 \le N \le 30$) исходного набора. Следующие N строк содержат по одной паре вещественных чисел – координаты соответствующей точки. Никакие три точки не лежат на одной прямой.

Формат выходных данных

Выведите в выходной файл количество различных триангуляций Делоне указанного набора точек.

Пример

input.txt	output.txt
4 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0	2

Решение.

Эта задача – самая сложная из предлагавшихся на олимпиаде. Для ее решения рассмотрим структуру, двойственную триангуляции Делоне: диаграмму Вороного.

Пусть дан набор точек $P = \{p_i = (x_i, y_i)\}_{i=1}^{N}$.

Определение. Областью Вороного точки $p_i \in P$ относительно набора P называется множество точек p' плоскости, для которых расстояние

$$dist(p', p_i) = \min_{j=1,..,N} dist(p', p_j)$$

(то есть множество точек, для которых она является ближайшей точкой из данного набора).

Разбиение плоскости на области Вороного для точек заданного набора называется диаграммой Вороного данного набора. Отметим, что некоторые точки могут принадлежать одновременно двум и более областям Вороного.

Диаграмма Вороного единственна для данного набора точек. Введем теперь понятие разбиения Делоне.

Определение. Разбиением Делоне назовем разбиение выпуклой оболочки заданного набора точек на выпуклые вписанные многоугольники, которое удовлетворяет свойствам 1–4 из условия с заменой слова «треугольник» на «многоугольник», и, кроме того, на окружности, описанной около каждого из многоугольников, лежат только вершины этого многоугольника.

Тогда любая триангуляция Делоне является триангуляцией какого-либо разбиения Делоне. Оказывается, что разбиение Делоне единственно и однозначно строится по диаграмме Вороного. В дальнейшем тексте зафиксируем набор точек.

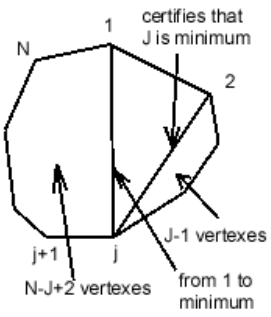
Предложение. Чтобы точка была центром окружности, описанной около какого-нибудь многоугольника разбиения Делоне, необходимо и достаточно, чтобы она принадлежала более чем двум областям Вороного.

Доказательство. Необходимость очевидна – центр описанной окружности равноудален от вершин многоугольника, около которого эта окружность описана, поскольку другие точки не лежат внутри окружности, расстояние до них больше, значит, эта точка принадлежит диаграммам Вороного сразу всех вершин многоугольника, то есть более чем двум. Для доказательства достаточности построим соответствующее разбиение Делоне. Для каждой точки, которая принадлежит более чем двум

областям Вороного, построим многоугольник, вершинами которого будут точки, диаграммам Вороного которых данная точка принадлежит. Легко видеть, что полученное разбиение удовлетворяет свойствам 1–4 (доказательство этого утверждения оставим в качестве упражнения).

Заметим, что мы попутно доказали, что разбиение Делоне единственно (мы однозначно построили его по диаграмме Вороного, и любой многоугольник любого разбиения принадлежит нашему по необходимости). Итак, осталось подсчитать количество способов триангуляции разбиения Делоне. Для того чтобы триангулировать разбиение Делоне, необходимо триангулировать каждый входящий в него многоугольник. Рассмотрим метод подсчета количества способов, которыми можно триангулировать выпуклый многоугольник.

Используем динамическое программирование: пусть мы знаем число способов, которыми можно триангулировать многоугольники с количеством вершин, меньшим N, $C[i]$ – число способов для многоугольника с количеством вершин i, найдем число способов триангулировать многоугольник с количеством вершин N. Рассмотрим вершину 1. Пусть она принадлежит ровно одному треугольнику, тогда количество способов есть $C[N-1]$. В противном случае рассмотрим положение хорды, идущей в вершину с наименьшим номером, пусть это j. Если j=3, то снова надо триангулировать многоугольник с количеством вершин N–1, в противном случае, имеем количество способов триангулировать многоугольник с количеством вершин j–1, умноженное на количество способов триангулировать многоугольник с количеством вершин N–j+2.



Итого

$$C[N] = C[N-1] + C[N-1] + \left| + \sum_{j=4}^{N-1} C[j-1]C[N-j+2] = \right.$$
$$= \sum_{j=3}^{N} C[j-1]C[N-j+2],$$

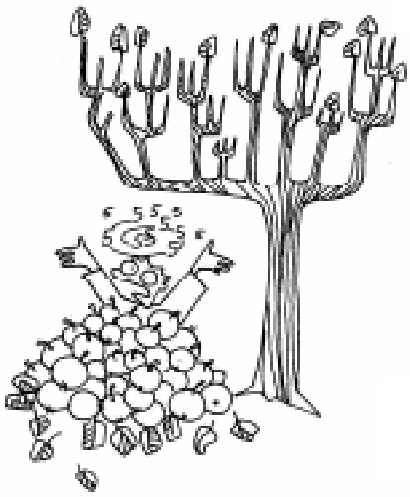
предполагая при этом, что $C[2] = 1$.

Эти числа хорошо известны в комбинаторике в связи с правильными скобочными последовательностями и называются числами Каталана. Так, в наших обозначениях $C[i]$ есть (i–2)-е число Каталана.

Итак, решение задачи выглядит следующим образом: найдем все наборы точек, которые являются вершинами многоугольника из разбиения Делоне (для этого рассмотрим все тройки точек, для каждой тройки найдем множество точек, которые лежат на окружности, описанной около треугольника, вершинами которого являются эти точки: все такие наборы и будут искомыми множествами), и перемножим числа $C[|V_i|]$ для всех таких наборов V_i. Полученное число дает ответ. Отметим также, что из-за того, что числа Каталана растут чрезвычайно быстро, существуют наборы, для которых ответ не помещается в стандартные целые типы данных, соответственно необходимо реализовать длинную арифметику.

Задача F. Яблоко от яблони...

У Пети в саду растет яблоня. Воодушевленный историей об Исааке Нью-



тоне, который, как известно, открыл закон всемирного тяготения после того, как ему на голову упало яблоко, Петя с целью повысить свою успеваемость по физике часто сидит под яблоней.

Однако, поскольку по физике у Пети твердая тройка, яблоки с его яблони падают следующим образом. В какой-то момент одно из яблок отрывается от ветки, на которой оно висит, и начинает падать строго вниз. Если в некоторый момент оно задевает другое яблоко, то то тоже отрывается от своей ветки и начинает падать вниз, при этом первое яблоко не меняет направления своего падения. Вообще, если любое падающее яблоко заденет другое яблоко на своем пути, то оно также начнет падать.

Таким образом, в любой момент каждое яблоко либо висит на ветке, либо падает строго вниз, причем все яблоки, кроме первого, чтобы начать падать, должны сначала соприкоснуться с каким-либо другим падающим яблоком.

Выясните, какие яблоки упадут с Петиной яблони.

Формат входных данных

Первая строка входного файла содержит N – количество яблок на Петиной яблоне ($1 \le N \le 200$). Следующие N строк содержат описания яблок. Будем считать все яблоки шарами. Каждое яблоко задается координатами своей самой верхней точки (той, где оно исходно прикреплено к дереву, длиной черенка пренебрежем) x_i, y_i и z_i и радиусом r_i ($-10000 \le x_i, y_i, z_i \le 10000$, $1 \le r_i \le 10000$, все числа целые). Гарантируется, что изначально никакие яблоки не пересекаются (даже не соприкасаются). Ось OZ направлена вверх.

Формат выходных данных

Выведите на первой строке выходного файла количество яблок, которые упадут с яблони, если начнет падать первое яблоко. На следующей строке выведите номера упавших яблок. Яблоки нумеруются, начиная с 1, в том порядке, в котором они заданы во входном файле.

Примеры

input.txt	output.txt
4 0 0 10 4 5 0 3 1 –7 4 7 1 0 1 2 6	3 1 2 4

Решение.

Эта задача эквивалентна задаче о нахождении достижимого множества вершин в ориентированном графе. Рассмотрим сначала эту задачу, а потом покажем, как наша задача сводится к ней.

Графом называется пара $G = \langle V, E \rangle$, где V – некоторое множество, которое называют множеством вершин графа, а E – отношение на V ($E \subset V \times V$) – множество ребер графа. Если отношение E симметрично (то есть $(u,v) \in E \Leftrightarrow (v,u) \in E$), то граф называют неориентированным, в противном случае граф называют ориентированным.

Обычно ориентированный граф изображают графически следующим образом: вершины графа (элементы множества V) изображают в виде точек или маленьких кружков, а ребра – в виде стрелок – если $(u,v) \in E$ (то есть вершины u и v соединены ребром), то из точки, изображающей вершину u проводят стрелку в вершину v. Пример графа изображен на рисунке.

При программировании вершины графа обычно сопоставляют числам от 1 до N, где $N = |V|$ – количество вершин графа, и рассматривают $V = \{1, 2, ..., N\}$. Для хранения графа в программе можно применить различные методы. Самым простым является хранение матрицы смежности, то есть двумерного массива, скажем A, где $A[i][j] = true$, если $(i, j) \in E$, и $A[i][j] = false$ в противном случае. Существуют также другие способы хранения графа, такие как список ребер или матрица инцидентности, однако в нашей задаче будет достаточно матрицы смежности.

Путем в графе называют последовательность ребер графа:

$U = < (u_1, v_1), (u_2, v_2), ..., (u_l, v_l) >$,

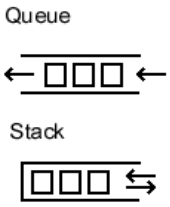
такую что $v_1 = u_2, v_2 = u_3, ..., v_{l-1} = u_l$. Интуитивно понятие пути соответствует последовательности стрелок, в которой каждая следующая начинается из конца предыдущей. Часто также путем называют последовательность вершин $< u_1, u_2, ..., u_l, v_l >$. Начальная вершина u_1 называется началом пути и обозначается beg(U), конечная вершина v_l называется концом пути и обозначается end(U), l называется длиной пути и обозначается len(U). Также рассматривают пустой путь (путь, не содержащий ни одной вершины). Будем считать, что в графе N пустых путей, причем для каждой вершины $v \in V$ есть ровно один пустой путь U_v, для которого $\text{beg}(U_v) = \text{end}(U_v) = v$.

Рассмотрим некоторый ориентированный граф $G = \langle V, E \rangle$. Пусть $v_0 \in V$ – некоторая его вершина. Тогда множеством вершин графа, достижимых из v_0 называется множество вершин, для которых есть путь из v_0 в эту вершину:

$R_{v0} = \{v \in V : \exists путь U : \text{beg}(U) = v_0, \text{end}(U) = v\}$

Отметим, что, благодаря нашему определению, $v_0 \in R_{v0}$, поскольку существует пустой путь из v_0 в v_0. Например, для первой вершины графа, изображенного на рисунке, $R_1 = \{1, 2, 4\}$.

Рассмотрим задачу нахождения множества вершин, достижимых из данной. Для этого можно применить процедуру поиска в ширину или в глубину. Поиск в ширину осуществляется следующим образом: заводится очередь, в которую сначала помещается вершина v_0, при этом вершина v_0 помечается как обработанная. Затем, пока очередь непуста, производится следующая операция: из головы очереди извлекается вершина, все непомеченные вершины, в которые из нее ведет ребро, помечаются и помещаются в очередь.



После окончания поиска в ширину множество помеченных вершин совпадает со множеством вершин, достижимых из данной. Поиск в глубину осуществляется аналогично, за тем исключением, что, вместо очереди, используется стек, то есть вершины извлекаются из того же конца, куда и добавляются. Отметим, что реализация поиска в глубину в программе обычно проще, поскольку в качестве стека можно использовать встроенный стек языка программирования, реализуя поиск в виде рекурсивной процедуры:

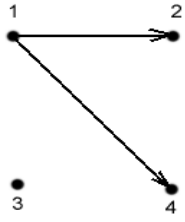
```
procedure find(x: longint);
var
    i: longint;
begin
    u[x] := true;
    for i := 1 to n do
       if a[x][i] and not u[i] then
            find(i);
end;
```

Здесь x – номер вершины, которая сейчас обрабатывается (находится на вершине стека), a – матрица смежности, u – массив отметок. Весь поиск в глубину осуществляется тогда с помощью вызова

```
find(v0);
```

где v0 – номер начальной вершины.

Теперь рассмотрим, как наша задача сводится к задаче поиска множества достижимых вершин. Рассмотрим граф, в котором вершинами будут яблоки, из i-й вершины в j-ю имеется ребро, если i-е яблоко задевает j-е при своем падении. Тогда множество вершин, достижимых из первой, и будет множеством упавших яблок. Граф, соответствующий примеру, приведен на рисунке.



Осталось только выяснить, как проверить, что *i*-е яблоко $<(X_i, Y_i, Z_i), R_i>$ задевает при падении *j*-е $<(X_j, Y_j, Z_j), R_j>$. С этой целью сделаем так, чтобы координаты *X*, *Y* и *Z* соответствовали не верхней точке яблока, а его центру. Для этого отнимем *R* от *Z* у всех яблок. Теперь, предполагая, что Z_i и Z_j – координаты центра соответствующих яблок, достаточно проверить, что:

1) $Z_i > Z_j$,

2) проекции яблок на плоскость, параллельную OXY, пересекаются (чтобы проверить, что два круга пересекаются, достаточно проверить, что $\sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \le R_i + R_j$).

Программа.

Приведем фрагмент программы, который вводит данные и преобразует их в матрицу смежности графа:

```
read(n);
for i := 1 to n do
begin
    read(x[i], y[i], z[i], r[i]);
    z[i] := z[i] - r[i];
end;

for j := 1 to n do
    for k := 1 to n do
    begin
        if (sqr(x[j] - x[k]) +
        sqr(y[j] - y[k]) <=
             sqr(r[j] + r[k])) and
             (z[k] < z[j]) then
           a[j][k] := true;
end;
```

Задача G. Контрольный блок.

Фирма Macrohard разработала новый протокол обмена данными по сети. Каждый блок данных при этом обмене состоит из *N* чисел в диапазоне от 0 до *M*–1 включительно. Чтобы повысить надежность передачи, вместе с блоком данных пересылается контрольный блок такой же длины.

Предположим, что исходный блок состоит из чисел $a_1, a_2, \ldots, a_N$. Тогда контрольный блок состоит из чисел $b_1, b_2, \ldots, b_N$ из диапазона от 0 до *M*–1 включительно, таких, что выполняются следующие равенства: $b_1 = (a_N + b_N) \bmod M$, $b_2 = (a_1 + b_1) \bmod M$, $b_3 = (a_2 + b_2) \bmod M$, ..., $b_N = (a_{N-1} + b_{N-1}) \bmod M$ (обозначение *X* mod *M* означает остаток от деления *X* на *M*, например, 7 mod 4 = 3, 6 mod 2 = 0).

Блоки данных, для которых нельзя построить контрольный блок, удовлетворяющий указанному свойству, считаются подозрительными, и их передача по сети не разрешается.

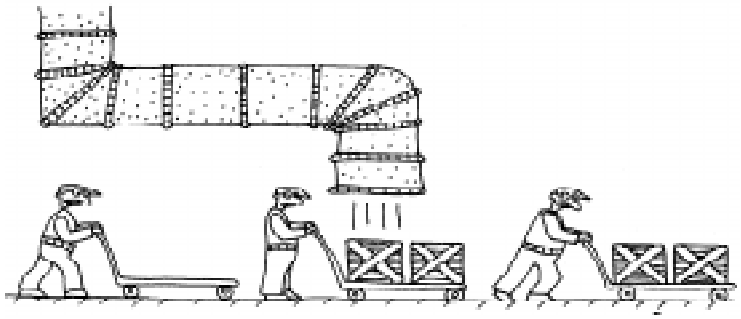
Ваня хочет поступить на работу программистом в фирму Macrohard, и в качестве вступительного задания ему поручили написать процедуру построения контрольного блока для заданного блока данных. Помогите ему!

Формат входных данных

Первая строка входного файла содержит числа *N* и *M* ($1 \le N \le 1000$, $2 \le M \le 10^9$). Следующая строка содержит блок данных, для которого следует построить контрольный блок, числа разделены пробелами.

Формат выходных данных

На первой строке выходного файла выведите YES, если для данного блока данных можно построить контрольный блок, и NO, если нельзя. В случае, если контрольный блок построить можно, на второй строке выведите контрольный блок. Числа разделяйте пробелами. Если решений несколько, можно выдать любое из них.



Примеры

input.txt	output.txt
4 3 1 0 0 2	YES 1 2 2 2
4 3 1 1 1 1	NO

Решение.

Прежде чем приступать к решению этой задачи, отметим одно свойство сложения по модулю: если $A = B \bmod M$, то $(A + C) \bmod M = (B + C) \bmod M$ (*).

Предположим, что для данного блока данных $a_1, a_2, \ldots, a_N$ существует какой-либо контрольный блок $b_1, b_2, \ldots, b_N$. Прибавим к каждому b_i число $(M - b_1)$ по модулю M. Получится новый блок данных той же длины

$c_1 = (b_1 + (M - b_1)) \bmod M$,

$c_2 = (b_2 + (M - b_1)) \bmod M$,

...,

$c_N = (b_N + (M - b_1)) \bmod M$.

Отметим, что $c_1 = 0$.

Благодаря свойству (*), набор $c_1, c_2, \ldots, c_N$ также оказывается контрольным блоком для исходного блока данных $a_1, a_2, \ldots, a_N$. Таким образом, получаем следующее утверждение: для того чтобы для некоторого блока данных $a_1, a_2, \ldots, a_N$ существовал контрольный блок данных $b_1, b_2, \ldots, b_N$, необходимо и достаточно, чтобы существовал контрольный блок с $b_1 = 0$.

Теперь решение задачи не представляет никакого труда: зная b_1, мы без труда из равенства $b_2 = (a_1 + b_1) \bmod M$ находим b_2, зная b_2, – b_3 и т. д. Осталось только после нахождения b_N проверить, что выполняется равенство:

$b_1 = (a_N + b_N) \bmod M$.

Отметим также тот факт, что, для того, чтобы существовал контрольный блок, необходимо и достаточно, чтобы $\sum_{i=1}^{N} a_i \bmod M = 0$, в чем легко убедиться, сложив равенства $b_1 = (a_N + b_N) \bmod M$, $b_2 = (a_1 + b_1) \bmod M$, $b_3 = (a_2 + b_2) \bmod M$, ..., $b_N = (a_{N-1} + b_{N-1}) \bmod M$ по модулю *M*.

Программа.

Приведем полный текст программы:

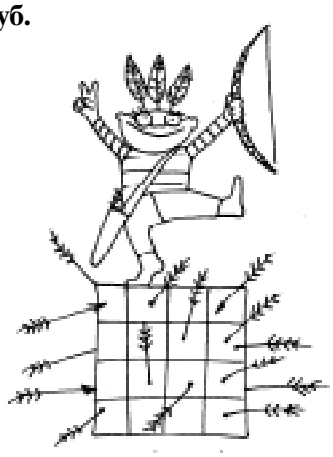
```
program control_block;
var
    a: array [0..1001] of longint;
    i, r, n, m: longint;
begin
    assign(input, "input.txt");
    reset(input);
    assign(output, "output.txt");
    rewrite(output);

    read(n, m);
    r := 0;
    for i := 1 to n do
    begin
        read(a[i]);
        r := (r + a[i]) mod m;
    end;

    if r = 0 then
    begin
        writeln("YES");
        for i := 1 to n do
        begin
            write(r, " ");
            r := (r + a[i]) mod m;
        end;
    end else begin
        writeln("NO");
    end;

    close(input);
    close(output);
end.
```

Задача H. Куб.



Петя склеил из N^3 единичных кубиков большой куб размером $N \times N \times N$. Устав от этой сложной работы, он отправился спать, а утром, проснувшись, с ужасом обнаружил, что

его младший брат Ваня K раз проткнул куб спицей.

При этом Ваня действовал очень аккуратно, каждый раз установив конец спицы точно в центр грани какого-нибудь граничного единичного кубика, он протыкал куб параллельно соответствующей оси координат, при этом целый ряд из N кубиков оказывался испорчен.

Немного успокоившись после этого тяжелого потрясения, Петя заинтересовался, сколько кубиков в его творении осталось неповрежденными. Помогите ему ответить на этот сложный вопрос.

Формат входных данных

На первой строке входного файла находятся числа N и K ($1 \le N \le 1000$, $0 \le K \le 150$). Следующие K строк описывают Ванины преступные действия. Каждая строка содержит три числа – два из них представляют собой соответствующие координаты всех кубиков, проткнутых спицей, а третье, соответствующее координате, в направлении которой был проткнут куб, равно 0. Например, если $N = 3$, тройка (1, 0, 3) означает, что спицей были проткнуты кубики (1, 1, 3), (1, 2, 3) и (1, 3, 3). Все координаты лежат в пределах от 1 до N. Известно, что Ваня никакое действие не выполнял два раза (то есть никакая тройка не встретится во входном файле дважды).

Формат выходных данных

Выведите в выходной файл единственное число – количество неповрежденных кубиков.

Пример

input.txt	output.txt
3 2 2 2 0 2 0 1	22

Решение.

Разделим все кубики на четыре категории: кубики, которые были проткнуты спицей три раза, кубики, которые были проткнуты спицей два раза, один раз, и кубики, которые не были проткнуты спицей вообще. Будем называть эти категории, соответственно, третьей, второй, первой и нулевой. Обозначим количество кубиков этих категорий за Q_3, Q_2, Q_1 и Q_0, соответственно. Очевидно, выполняется равенство $Q_3 + Q_2 + Q_1 + Q_0 = N^3$. Наша задача – узнать Q_0. Если мы узнаем Q_3, Q_2, Q_1, то мы сразу найдем $Q_0 = N^3 - Q_1 - Q_2 - Q_3$.

Отметим, что количество граней кубиков, проткнутых спицами, не зависит от Q_1, Q_2 и Q_3 и равно $2NK$. Отметим также, что у кубиков третьей категории проткнуто спицей 6 граней, у кубиков второй категории – 4, а у кубиков первой – 2. Отсюда получаем равенство $6Q_3 + 4Q_2 + 2Q_1 = 2NK$, или, сокращая на 2, $3Q_3 + 2Q_2 + Q_1 = NK$.

Теперь рассмотрим способ найти количество кубиков третьей категории. Для этого надо найти количество троек проколов, которые пересекаются в одной точке. Будем называть тройку чисел, которые задают прокол, координатами прокола. Отметим, что проколы с координатами (X_1, Y_1, Z_1), (X_2, Y_2, Z_2) и (X_3, Y_3, Z_3) пересекаются в одной точке, если выполняются следующие условия:

1. Нули в этих тройках должны стоять на разных позициях. Это можно записать как:

$$X_1X_2X_3 + Y_1Y_2Y_3 + Z_1Z_2Z_3 = 0$$

(действительно, в каждой тройке ровно один 0, остальные числа – положительные, чтобы сумма трех неотрицательных чисел была равна 0, они все должны быть равны 0).

2. В каждой из трех ненулевых координат ненулевые координаты должны совпадать. Это можно записать в виде

$$\sum_{V=X,Y,Z} \sum_{i=1}^{3} \sum_{j=3}^{3} \left|V_i - V_j\right| V_i V_j = 0$$

(несложно убедиться, что, если хотя бы одна пара ненулевых соответствующих координат не совпадает, соответствующее слагаемое не равно 0, в противном случае все слагаемые равны 0).

Таким образом, перебрав все тройки проколов и проверив для них выполнение указанных двух свойств, мы найдем Q_3.

Теперь найдем количество попарных пересечений проколов. Проколы с координатами (X_1,Y_1,Z_1) и (X_2,Y_2,Z_2) пересекаются в одной точке, если выполняются следующие условия:

1. Нулевые координаты в них не совпадают (выражается формулой $\prod_{V=X,Y,Z}(V_1+V_2)\neq 0$).
2. Ненулевые координаты у них равны (формула $\sum_{V=X,Y,Z}|V_1-V_2|V_1V_2=0$).

Отметим, что на кубик третьей категории приходятся три пары пересекающихся проколов, а на кубик второй категории – одна. Таким образом, если количество пар пересекающихся проколов P, получаем $3Q_3+Q_2=P$. Отсюда

$$Q_2=P-3Q_3,$$
$$Q_1=NK-3Q_3-2Q_2,$$
$$Q_0=N^3-Q_1-Q_2-Q_3.$$

Описанное решение не единственно, однако является одним из наиболее простых в реализации. Как альтернативу приведенным формулам можно рассматривать разбиение проколов на три категории в соответствии с направлением и дальнейший анализ троек и пар пересечений. Отметим лишь, что при указанных ограничениях ($N\le 1000$, $K\le 150$) альтернативные методы, например, запоминающие уже проткнутые кубики или эмулирующие процесс протыкания с дальнейшим подсчетом оставшихся кубиков, неэффективны, так как требуют до $O(N^3)$ действий и от $O(K^3)$ до $O(N^3)$ памяти. Ни то ни другое неприемлемо. Эффективность же описанного метода не зависит от N и есть $O(K^3)$.

Программа.

Полный текст программы приведен на дискете.

Станкевич Андрей Сергеевич, студент СПбГИТМО (ТУ), председатель жюри второй Всероссийской командной олимпиады школьников по информатике.