Задача A. Выражение из скобок

Имя входного файла:	`bracket-expression.in`
Имя выходного файла:	`bracket-expression.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

В этой задаче требуется найти число, сопоставленное выражению из скобок.

Как вы, возможно, знаете, множество *правильных скобочных последовательностей* $\mathcal{S}$ определяется рекурсивно следующим образом:

1. $\varepsilon \in \mathcal{S}$ (пустая строка)
2. $A \in \mathcal{S} \Rightarrow (A) \in \mathcal{S}$ (взятие в скобки)
3. $A, B \in \mathcal{S} \Rightarrow AB \in \mathcal{S}$ (конкатенация)

Например, «(())» $\in \mathcal{S}$ по правилу 2 и потому, что «()» $\in \mathcal{S}$; это, в свою очередь, верно по правилу 2 и потому, что «» $\in \mathcal{S}$; последнее верно по правилу 1. Аналогично, «()()» $\in \mathcal{S}$ по правилу 3 и потому, что «()» $\in \mathcal{S}$; последнее доказано выше.

Сопоставим правильным скобочным последовательностям $X \in \mathcal{S}$ числа $f(X)$ по следующим правилам:

1. $\varepsilon \longrightarrow 1$
2. $(A) \longrightarrow 1 + f(A)$
3. $AB \longrightarrow f(A) \cdot f(B)$

Можно показать, что сопоставление корректно, так как не зависит от порядка вычислений. Заметим, что различным последовательностям может быть сопоставлено одно и то же число.

По заданной правильной скобочной последовательности найдите сопоставленное ей число.

Формат входных данных

В первой строке ввода задано выражение из открывающих («(») и закрывающих («)») скобок длиной от 0 до 40 символов включительно. Гарантируется, что это выражение является правильной скобочной последовательностью.

Формат выходных данных

В единственной строке выведите целое число, сопоставленное данному выражению.

Примеры

bracket-expression.in	bracket-expression.out
(())	3
()()	4

Пояснения к примерам

В первом примере выражение «(())» сопоставлено числу $(1 + 1) + 1 = 3$.

Во втором примере выражение «()()» сопоставлено числу $(1 + 1) \cdot (1 + 1) = 2 \cdot 2 = 4$.

Задача B. Шашки

Имя входного файла:	`checkers.in`
Имя выходного файла:	`checkers.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

На одном сервере для игры в шашки установлено N различных движков. Вася хочет сыграть с движками ровно K партий. Вася очень любит играть белым цветом. Поэтому он хочет максимизировать количество партий, в которых он будет играть белыми.

С каждым движком Вася последовательно проводит серию партий: сначала он играет с первым движком, потом со вторым, и так далее. После серии с последним движком Вася снова играет с первым движком и так далее по циклу. Сколько партий будет сыграно в конкретной серии, выбирает Вася, но серия обязательно должна состоять из одной или из двух партий.

Цветовой историей игрока назовём следующую последовательность цветов: каким цветом играл этот игрок в предыдущей партии, в пред-предыдущей, и так далее. Выбор цвета в конкретной партии определяется по следующему алгоритму. Рассмотрим истории цветов движка и Васи и сравним их как последовательности: пока предыдущие цвета совпадают, переходим к цветам на одну партию назад. Если нашлось различие, найденные цвета «переворачиваются». Если же сыгранных партий недостаточно для того, чтобы определить цвет по этому алгоритму, Вася будет играть белым цветом.

Например, пусть Вася играл в предыдущей партии чёрными, а в предпредыдущей — белыми. Далее, пусть движок играл чёрными в двух предыдущих партиях. Тогда мы сначала сравним чёрный цвет с чёрным (предыдущие партии), а потом Васин белый с чёрным цветом движка (пред-предыдущие партии) и остановимся на этом различии. Эти цвета «переворачиваются»: значит, Вася будет играть чёрными, а движок — белыми.

Помогите Васе максимизировать количество партий, сыгранных белым цветом.

Формат входных данных

Первая строка ввода содержит два положительных целых числа N и K, не превосходящих 20. В следующих N строках перечислены цвета, которыми играли движки до встреч с Васей. Цвета перечислены в исторической последовательности; в частности, последним указан цвет, которым движок сыграл предыдущую партию. Длина каждой такой строки не меньше одного и не больше 100 символов. Каждая строка состоит только из символов «`W`» и «`B`».

Формат выходных данных

Выведите строку, содержащую единственное целое число — максимальное число партий, в которых Вася будет играть белым цветом.

Примеры

checkers.in	checkers.out
2 2 W W	2
2 5 W B	3

Задача C. Выпуклое и компактное

Имя входного файла:	`convexset.in`
Имя выходного файла:	`convexset.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Вам даны n точек на плоскости.

Гарантируется, что точки получены следующим образом:

```
x[i] = random [0..1000]
y[i] = random [0..1000]
```

Здесь функция `random [L..R]` выдаёт случайное целое число от L до R включительно, и все целые числа от L до R имеют равную вероятность.

Выберите k точек так, чтобы периметр выпуклой оболочки выбранных точек был минимальным.

Формат входных данных

В первой строке ввода заданы количество точек n ($3 \leqslant n \leqslant 60$) и число k ($1 \leqslant k \leqslant 15$). Следующие n строк содержат координаты точек x_i y_i (целые числа от 0 до 1000). Точки занумерованы целыми числами от 1 до n. Гарантируется, что вам везёт, и случайно сгенерированные n точек во вводе обладают следующим свойством: никакие две не совпадают и никакие три не лежат на одной прямой.

Формат выходных данных

В первой строке выведите вещественное число — периметр выпуклой оболочки выбранных точек. Во второй строке выведите k различных целых чисел от 1 до n — номера выбранных точек. Если оптимальных ответов несколько, выведите любой из них. Точки можно выводить в любом порядке.

Рассмотрим три вещественных числа: правильный ответ, периметр выпуклой оболочки набора точек, который вы вывели, и вещественное число, которое вы вывели в первой строке. Ваш ответ будет считаться правильным только в том случае, если разница максимального и минимального из этих трёх чисел не превосходит 10^{-6}.

Пример

convexset.in	convexset.out
5 4 0 0 0 1 4 4 2 0 2 1	6 4 5 1 2

Замечание

В примере для наглядности точки не случайны. В тестирующей системе первый тест совпадает с тестом из условия, остальные случайны.

Задача D. Запрещённые слова

Имя входного файла:	`forbidden-words.in`
Имя выходного файла:	`forbidden-words.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

В этой задаче требуется выяснить, кто выигрывает в игре со словарём.

Задан словарь — набор из n различных непустых слов из маленьких букв английского алфавита.

Двое играют в игру, по очереди называя по одной букве, причём нельзя называть букву, уже названную кем-то ранее. Проигрывает тот, после чьей буквы из названных букв можно составить хотя бы одно слово из заданного словаря. Каждую названную букву можно использовать при составлении слова сколько угодно раз.

Кто выигрывает при правильной игре?

Формат входных данных

В первой строке ввода задано целое число n — количество слов ($1 \leqslant n \leqslant 10^5$). В следующих n строках заданы сами слова, по одному на строке. Каждое слово непусто и состоит из маленьких букв английского алфавита. Гарантируется, что все заданные слова попарно различны, а их суммарная длина не превосходит 3 000 000.

Формат выходных данных

Если при правильной игре выигрывает первый игрок, выведите «`First`». В противном случае выведите «`Second`».

Примеры

forbidden-words.in	forbidden-words.out
2 a ab	First
3 da dee ea	Second

Пояснения к примерам

В первом примере проигрывает тот, кто назовёт букву «`a`». При правильной игре все остальные буквы будут названы раньше, а когда они кончатся, очередь хода будет за вторым игроком.

Во втором примере проигрывает тот, кто называет вторую букву из множества $\{\texttt{a}, \texttt{d}, \texttt{e}\}$. До этого момента при правильной игре будут названы 24 буквы, и очередь хода будет за первым игроком.

Задача E. Четыре простых числа

Имя входного файла:	`fourprimes.in`
Имя выходного файла:	`fourprimes.out`
Ограничение по времени:	1 секунда (*2 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Дано целое число n.
Сколько способов представить его в виде суммы четырёх простых чисел?

Формат входных данных

В единственной строке ввода задано целое число n ($1 \leqslant n \leqslant 10^5$).

Формат выходных данных

Выведите одно число: количество способов представить n в виде суммы четырёх простых чисел.

Пример

fourprimes.in	fourprimes.out
9	4

Замечание

Число $9 = 2 + 2 + 2 + 3 = 2 + 2 + 3 + 2 = 2 + 3 + 2 + 2 = 3 + 2 + 2 + 2$.

Задача F. Пересечение множеств

Имя входного файла:	`intset.in`
Имя выходного файла:	`intset.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

У Васи и Пети есть два одинаковых набора карточек. Каждый набор состоит из N карточек. На каждой карточке написано одно целое число. В наборе присутствуют все числа от 1 до N.

Вася случайно выбирает из первого набора ровно L карточек. Петя случайно выбирает из второго набора ровно M карточек. Все варианты выбора равновероятны, а варианты выбора Васи и Пети независимы. После выбора карточек Вася и Петя считают, сколько чисел у них совпало. Если количество совпавших чисел строго меньше, чем K, победителем игры объявляется Петя. В противном случае выигрывает Вася.

Ваша задача — подобрать число K так, чтобы игра была как можно более честной. Формально нужно найти максимальное K, не превышающее N и такое, чтобы вероятность выигрыша Васи была не менее 0.5.

Формат входных данных

В первой строке заданы три целых положительных числа N, L и M ($1 \leqslant L, M \leqslant N \leqslant 1\,000\,000$).

Формат выходных данных

Выведите неотрицательное целое число K. Разрешается решить задачу не совсем точно: решение участника будет принято, если абсолютное значение разности между ответом участника и решением жюри не будет превышать 10.

Пример

intset.in	intset.out
4 1 2	1

Задача G. Медали

Имя входного файла:	`medals.in`
Имя выходного файла:	`medals.out`
Ограничение по времени:	3 секунды (*5 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Одна маленькая, но очень гордая африканская страна готовится к тому, чтобы принять участие в Зимних Олимпийских Играх 2014-го года. Выбраны лучшие спортсмены страны, из которых теперь нужно собрать сборную — определить, кто же поедет в солнечный город Сочи, и кто в какой дисциплине примет участие.

Как известно, на этих Играх впервые будут разыграны комплекты из десяти медалей: вместо привычных золота, серебра и бронзы награждаются лучшие десять спортсменов в каждой дисциплине. За первое место будут давать платиновую медаль, за второе — золотую, за третье — палладиевую, за четвёртое — цезиевую, далее серебро, вольфрам, никель, медь, магний и, наконец, за десятое место можно получить медаль из технического алюминия.

После длительных тренировок стало известно, на какие дисциплины имеет смысл отправлять каждого спортсмена, а также какие медали он за них получит. Ваша задача — распределить спортсменов по дисциплинам так, чтобы получить максимальное количество платиновых медалей; если есть несколько вариантов, то следует получить максимальное количество золотых, и так далее до алюминия. При этом, конечно же, нельзя отправлять на одну дисциплину нескольких спортсменов (по правилам Игр), а кроме того, нельзя заставлять одного спортсмена участвовать в нескольких дисциплинах (это была бы слишком большая нагрузка для него).

Формат входных данных

В первой строке ввода содержится одно число n: количество спортсменов. В следующих n строках содержатся описания спортсменов, по одному на строке. Описание i-го спортсмена состоит из числа k_i (количество различных дисциплин, в которых i-й спортсмен может соревноваться) и следующих за ним k_i пар чисел: номер дисциплины и тип медали, которую спортсмен там получит (1 соответствует платиновой медали, 10 — алюминиевой).

Ограничения: $1 \leqslant n \leqslant 1000$, $k_i \geqslant 1$, $\sum_{i=1}^{n} k_i \leqslant 10\,000$, номера дисциплин лежат в интервале от 1 до 1000, типы медалей — от 1 до 10, все числа во вводе целые.

Формат выходных данных

В первой строке выведите десять чисел: количество медалей каждого типа, начиная с платиновых, которые может получить сборная. Во второй строке выведите n чисел: i-е из них должно быть номером дисциплины, в которой следует принять участие i-му спортсмену (или 0, если этому спортсмену не нужно ни в чём участвовать). Если оптимальных решений несколько, выведите любое из них.

Пример

medals.in	medals.out
5 2 1 1 2 1 2 2 2 3 3 1 1 2 1 3 3 1 4 1	2 1 1 0 0 0 0 0 0 0 2 0 1 3 4

Задача H. Достижимость

Имя входного файла:	`reachability.in`
Имя выходного файла:	`reachability.out`
Ограничение по времени:	3 секунды (*8 секунд для Java*)
Ограничение по памяти:	256 мебибайт

Дан ориентированный граф из n вершин. Рёбер изначально нет.
Ребро из a в b будем обозначать (a, b).
Нужно уметь обрабатывать запросы четырёх типов:

1. «`+ o` v k a_1 ... a_k» — добавить рёбра $(v, a_1), \ldots, (v, a_k)$
2. «`+ i` v k a_1 ... a_k» — добавить рёбра $(a_1, v), \ldots, (a_k, v)$
3. «`- o` v k a_1 ... a_k» — удалить рёбра $(v, a_1), \ldots, (v, a_k)$
4. «`- i` v k a_1 ... a_k» — удалить рёбра $(a_1, v), \ldots, (a_k, v)$

В каждом запросе все a_i различны.
Если ребро добавляется, гарантируется, что перед этим его не было.
Если ребро удаляется, гарантируется, что перед этим оно было.
Гарантируется, что в каждый момент времени граф ацикличен.
Ваша задача — поддерживать матрицу достижимости:

$$a_{ij} = \begin{cases} 1, \text{ если } i \neq j \text{ и из } i \text{ достижима } j \text{ по рёбрам графа} \\ 0, \text{ в противном случае} \end{cases}$$

Формат входных данных

В первой строке заданы целые числа n — количество вершин, q — количество запросов, а также два числа A и B, которые пригодятся для вывода данных ($1 \leqslant n \leqslant 400$, $1 \leqslant q \leqslant 800$, $1 \leqslant A, B \leqslant 10^9$). Далее следуют q строк, каждая из которых содержит описание запроса в формате, указанном выше.

Формат выходных данных

После каждого запроса выведите значение выражения

$$\left(\sum_{i,j=1}^{n} a_{ij} A^i B^j \right) \bmod 2^{32}$$

на отдельной строке.

Пример

reachability.in	reachability.out
2 4 3 7 + o 1 1 2 - i 2 1 1 + i 1 1 2 - o 2 1 1	7 0 3 0

Задача I. Вращающиеся лазеры

Имя входного файла:	`revolving-lasers.in`
Имя выходного файла:	`revolving-lasers.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

В этой задаче требуется двигать робота по плоскости так, чтобы уклоняться от лазеров сколь угодно долго.

На плоскости находится n лазеров. Каждый лазер состоит из источника, установленного в точке с целыми чётными координатами, и бесконечного луча с началом в этой точке. Лазер с номером i равномерно поворачивается на t_i градусов в секунду: *отрицательное* число означает поворот по часовой стрелке, *положительное* — против. Изначально все лазеры направлены вверх сонаправленно с осью Oy. Никакие два источника не установлены в одной и той же точке. Лазеры не мешают друг другу, то есть лучи и источники не препятствуют распространению других лучей по плоскости.

Мы управляем роботом, который начинает движение в начале координат, а в течение каждой секунды либо остаётся на месте, либо равномерно и прямолинейно перемещается в одну из восьми соседних точек с целыми координатами. Две различные точки с целыми координатами (x_1, y_1) и (x_2, y_2) считаются соседними, если обе их координаты отличаются не более чем на единицу: $|x_2 - x_1| \leqslant 1$ и $|y_2 - y_1| \leqslant 1$.

Требуется построить маршрут, по которому робот может ходить и избегать лазеров сколь угодно долго, или же выяснить, что такого маршрута не существует. Избегание лазеров означает, что робот не может находиться на источнике лазера, а также пересекать луч лазера или касаться его. Размерами объектов и толщиной луча лазера следует пренебречь.

Формат входных данных

В первой строке ввода задано целое число n — количество лазеров ($0 \leqslant n \leqslant 3$). В следующих n строках описаны лазеры, по одному в строке. Каждый лазер i задаётся тремя целыми числами x_i, y_i и t_i через пробел — координаты источника и скорость поворота в градусах в секунду ($2 \leqslant x_i, y_i \leqslant 10$, x_i и y_i чётны, $-3 \leqslant t_i \leqslant 3$). Гарантируется, что никакие два источника не находятся в одной точке.

Формат выходных данных

Выведите непустую строку S не более чем из 10 000 символов, соответствующую найденному маршруту. Каждый символ строки S_i задаёт действия робота в течение i-й секунды. После выполнения всех действий, заданных в маршруте, положения лазеров и робота должны соответствовать изначальным. Символы, кодирующие направление движения, соответствуют положению стрелок на цифровой клавиатуре: при нахождении в точке (x, y)

цифры

7	8	9
4	5	6
1	2	3

означают перемещение в точки

$(x-1, y+1)$	$(x, y+1)$	$(x+1, y+1)$
$(x-1, y)$	(x, y)	$(x+1, y)$
$(x-1, y-1)$	$(x, y-1)$	$(x+1, y-1)$

.

Если возможных ответов несколько, можно вывести любой из них. В случае, если избегать лазеров сколь угодно долго невозможно, выведите вместо маршрута слово «`None`».

Примеры

revolving-lasers.in	revolving-lasers.out
1 2 2 1	69*555..(177)..55*9*555..(89)..55*75112*555..(86)..55*
2 2 2 1 4 2 -1	None

Пояснения к примерам

В первом примере маршрут из 360 действий показан в сокращённом варианте для удобства чтения: фрагменты вида «*ddd..(X)..dd*» означают повторение цифры d ровно X раз. Робот огибает единственный источник лазера против часовой стрелки и возвращается в начало координат.

Во втором примере избегать лазеров сколь угодно долго невозможно.

Задача J. Змейки на камне

Имя входного файла:	`snakes2.in`
Имя выходного файла:	`snakes2.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

На большом камне улеглись несколько трёхмерных змеек. Каждая из них держит в зубах свой собственный хвост. Камень плоский, и на нём задана двумерная декартова система координат, делящая его на квадратные клетки со стороной 1.

Проекция каждой змейки на поверхность камня представляет собой последовательность клеток, в которой две соседние клетки различны и имеют общую сторону, в том числе последняя и первая клетка. Будем говорить, что змейка *проходит через клетку* a раз, если эта клетка a раз присутствует в последовательности клеток, на которые спроецировалась змейка.

Будем называть *перекрёстком* клетку, через которую проходит больше одной змейки, или одна змейка, но несколько раз. Змейка проходит через перекрёсток *сверху*, если соответствующая часть змейки дальше от камня, чем все остальные части змеек, которые спроецировались на перекрёсток; аналогично, змейка проходит через перекрёсток *снизу*, если соответствующая часть змейки ближе к камню.

Змейки улеглись на камне таким образом, что перекрёстки бывают только двух типов. В первом случае через перекрёсток проходит одна и та же змейка, один раз сверху и один раз снизу. Во втором через перекрёсток проходят две змейки, одна сверху и другая снизу. Кроме того, никакие два перекрёстка не расположены в соседних по стороне клетках, и все змейки проходят перекрёстки по прямой.

Пусть i-я змейка проходит через перекрёстки r_i раз, при этом некоторые из них она может проходить дважды, а некоторые один раз. Каждый из перекрёстков она может проходить либо сверху (обозначим это символом «+»), либо снизу (обозначим это символом «-»). Способ прохождения перёкрестков змейкой будем задавать строкой, состоящей из r_i символов «+» и «-», записанных для всех перекрёстков, встреченных в порядке обхода змейки от головы до хвоста.

Даны k последовательностей клеток, соответствующих k змейкам ($1 \leqslant k \leqslant 3$). Общее число перекрёстков для этих последовательностей не превосходит 7, при этом каждая змейка проходит через какой-нибудь перекрёсток хотя бы один раз. Координаты клеток находятся в пределах от 1 до $m = 25$.

Нужно найти такой способ прохождения змейками перекрёстков, чтобы змейки могли, не отпуская зубами своих хвостов, занять такое положение в пространстве, что проекция змеек на камень будет представлять собой k непересекающихся окружностей, если пренебречь толщиной змеек.

Формат входных данных

В первой строке ввода заданы одно целое число k — количество змеек ($1 \leqslant k \leqslant 3$). В каждой из следующих k строк задана одна из змеек. Описание змейки с номером i начинается с целого числа l_i — её длины ($4 \leqslant l_i \leqslant m^2$). За ним следуют l_i пар целых чисел $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$, …, x_{i,l_i}, y_{i,l_i} — координаты клеток, через которые проходит змейка, в порядке обхода ($1 \leqslant x_{i,j}, y_{i,j} \leqslant m$).

Формат выходных данных

Выведите k строк, состоящих из символов «+» и «-», где i-я строка задаёт для змейки с номером i способ прохождения каждого встреченного ей перекрёстка. Выходные данные должны быть корректными, то есть каждый перекрёсток должен быть пройден змейками один раз сверху и один раз снизу.

Если возможных ответов несколько, выведите любой.

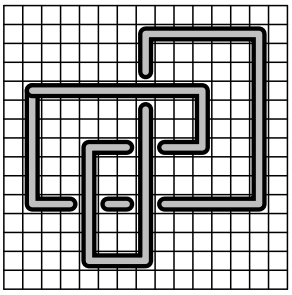
Примеры

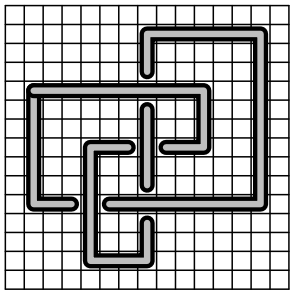
snakes2.in
1 72 2 5 3 5 4 5 5 5 6 5 7 5 8 5 9 5 10 5 11 5 11 6 11 7 11 8 10 8 9 8 8 8 7 8 6 8 5 8 5 9 5 10 5 11 5 12 5 13 5 14 6 14 7 14 8 14 8 13 8 12 8 11 8 10 8 9 8 8 8 7 8 6 8 5 8 4 8 3 8 2 9 2 10 2 11 2 12 2 13 2 14 2 14 3 14 4 14 5 14 6 14 7 14 8 14 9 14 10 14 11 13 11 12 11 11 11 10 11 9 11 8 11 7 11 6 11 5 11 4 11 3 11 2 11 2 10 2 9 2 8 2 7 2 6
snakes2.out
+-+++---

snakes2.in
2 36 5 2 5 3 5 4 5 5 5 6 5 7 5 8 5 9 5 10 5 11 5 12 5 13 5 14 5 15 5 16 5 17 6 17 7 17 8 17 8 16 8 15 8 14 8 13 8 12 8 11 8 10 8 9 8 8 8 7 8 6 8 5 8 4 8 3 8 2 7 2 6 2 36 2 5 2 6 2 7 2 8 2 9 2 10 2 11 2 12 2 13 2 14 3 14 4 14 5 14 6 14 7 14 8 14 9 14 10 14 11 14 11 13 11 12 11 11 11 10 11 9 11 8 11 7 11 6 11 5 10 5 9 5 8 5 7 5 6 5 5 5 4 5 3 5
snakes2.out
--++ +--+

Пояснения к примерам

Длина и координаты всех клеток каждой змейки расположены в одной строке. В условии эта строка разбита на несколько строк для удобства чтения.

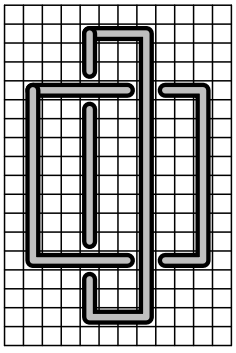


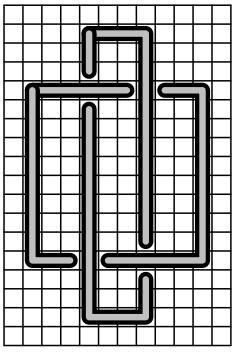


В первом примере дана одна змейка. Если поставить плюсы и минусы как в ответе, то она будет выглядеть как на левом рисунке и легко распутается. А если поставить, например,

+-+-+-+-+-

то она будет выглядеть как на правом рисунке, и такой ответ является неверным.





Во втором примере, если поставить плюсы и минусы как в ответе, то змейки будут выглядеть как на левом рисунке. Получившиеся прямоугольники вынимаются друг из друга и змейки могут расцепиться.

А если поставить, например,

-+-+

-+-+

то змейки будут выглядеть как на правом рисунке, и такой ответ является неверным.

Задача K. Зависимое подмножество

Имя входного файла:	`subset.in`
Имя выходного файла:	`subset.out`
Ограничение по времени:	2 секунды (*3 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Даны n векторов в d-мерном пространстве. Нужно выбрать максимальное по размеру подмножество векторов A такое, что:

$$\forall a, b, c \in A \quad \exists x, y, z \in \mathbb{Z}\colon \begin{cases} ax + by + cz = 0 \\ x^2 + y^2 + z^2 > 0 \end{cases}$$

Формат входных данных

В первой строке ввода задано два целых числа n и d ($1 \leqslant n \leqslant 1000$, $1 \leqslant d \leqslant 10$). Следующие n строк содержат по d целых чисел $a_{i,1}, \ldots, a_{i,d}$ — сами векторы. Гарантируется, что $|a_{i,j}| \leqslant 10^4$ и $\forall i \sum_{j=1}^{d} a_{i,j}^2 > 0$. Векторы пронумерованы целыми числами от 1 до n в том порядке, в котором они заданы.

Формат выходных данных

В первой строке выведите k — количество векторов в найденном множестве. Во второй строке выведите k различных чисел от 1 до n — номера векторов в исходном множестве. Если наборов с максимальным k несколько, выведите любой.

Пример

subset.in	subset.out
6 3 3 -2 0 9 0 0 0 9 0 0 0 9 3 3 0 0 1 1	4 2 3 1 5