Problem A. Bracket Expression

Input file:	`bracket-expression.in`
Output file:	`bracket-expression.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to find the number corresponding to a given bracket expression.

As you may already know, the set of *regular bracket sequences* $\mathcal{S}$ is recursively defined as follows:

1. $\varepsilon \in \mathcal{S}$ (empty sequence)
2. $A \in \mathcal{S} \Rightarrow \texttt{(}A\texttt{)} \in \mathcal{S}$ (bracketing)
3. $A, B \in \mathcal{S} \Rightarrow AB \in \mathcal{S}$ (concatenation)

For example, “`(())`” $\in \mathcal{S}$ by rule 2 and the fact that “`()`” $\in \mathcal{S}$; this, in turn, follows by rule 2 from the fact that “” $\in \mathcal{S}$; the latter is simply rule 1. Similarly, “`()()`” $\in \mathcal{S}$ by rule 3 and the fact that “`()`” $\in \mathcal{S}$; the latter is proven above.

We assign a number $f(X)$ to every regular bracket sequence $X \in \mathcal{S}$ as follows:

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

It can be shown that this assignment is proper because it does not depend on the order of calculation. Note that the same number can be assigned to different sequences.

Given a regular bracket sequence, find the number assigned to it.

Input

The first line of input contains an expression which consists of opening (“`(`”) and closing (“`)`”) brackets and has length from 0 to 40 characters, inclusive. It is guaranteed that this expression represents a regular bracket sequence.

Output

On the only line, print the number assigned to the given expression.

Examples

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

Explanations

In the first example, the expression «`(())`» is assigned the number $(1 + 1) + 1 = 3$.

In the second example, the expression «`()()`» is assigned the number $(1 + 1) \cdot (1 + 1) = 2 \cdot 2 = 4$.

Problem B. Checkers

Input file:	`checkers.in`
Output file:	`checkers.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

A checkers-playing server hosts N different playing engines. Vasya wants to play exactly K games against these engines. Vasya likes to play white very much. So he wants to maximize the number of games in which he plays white.

Vasya plays sets of games against each engine sequentially: first, he plays against the first engine, then against the second, and so on. After playing against the last one, Vasya can once again play against the first engine and then continue the cycle. The number of games in each set is chosen by Vasya, but each set must consist of one or two games.

Let the *color history* of a player be the following sequence of colors: the color he played in the previous game, the color in the pre-previous game, and so on. In each new game, the colors are distributed by the following algorithm. Consider the color histories of the engine and Vasya and compare them as sequences: as long as the previous colors are the same, look one game backwards. If the difference is found, the two colors are "swapped". If the history is too short to distribute colors by the above algorithm, Vasya will play white.

For example, suppose that Vasya played black in the previous game but white in the pre-previous one. Next, suppose that an engine played black in the previous two games. First, we compare black and black (previous games). After that, we compare Vasya's white and engine's black (pre-previous games) and stop as we found the difference. The colors are then "swapped": it means that now Vasya will play black and the engine will play white.

Help Vasya maximize the number of games he plays as white.

Input

The first line of input contains two positive integers N and K not exceeding 20. Next N lines contain the colors which the engines played before meeting Vasya. The colors are listed in historical order; in particular, the last color mentioned is the color which the engine played in its previous game. The length of each such line is at least one and at most 100 characters. Each line consists only of characters "`W`" and "`B`".

Output

On the only line of output, print a single integer: the maximal number of games Vasya can play as white.

Examples

`checkers.in`	`checkers.out`

`2 2`
`W`
`W`
	`2`

`2 5`
`W`
`B`
	`3`

Problem C. Convex and Compact

Input file:	`convexset.in`
Output file:	`convexset.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

You are given n points on the plane.

It is guaranteed that the points were obtained in the following way:

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

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

Here, the function `random [L..R]` returns a random integer from L to R inclusive, and all integers from L to R have the same probability.

You have to choose a set of k points in such a way that the perimeter of the convex hull of the chosen set will be minimal possible. When $k = 2$, the perimeter is twice the length of the segment between two points.

Input

The first line of input contains the number of points n ($3 \le n \le 60$) and the number k ($1 \le k \le 15$, $k \le n$). The next n lines contain coordinates of points x_i y_i (integers from 0 to 1000). The points are numbered by integers from 1 to n. It is guaranteed that you are lucky, so the n randomly generated points satisfy the following property: no two points coincide and no three points lie on the same line.

Output

On the first line, print a real number: the perimeter of the convex hull of the chosen set. On the second line, print k different integers from 1 to n: the numbers of the chosen points. If there are several optimal answers, print any one of them. You may print points in any order.

Consider three real numbers: the right answer, the perimeter of the convex hull of the chosen set and the real number which you printed on the first line. Your answer will be accepted only if the difference between the greatest and the least of these three numbers will not exceed 10^{-6}.

Example

convexset.in	convexset.out

5 4
0 0
0 1
4 4
2 0
2 1
	6
	4 5 1 2

Note

In the example, the points are not random for simplicity. In the testing system, the first test coincides with this example, and all other tests are truly random.

Problem D. Forbidden Words

Input file:	`forbidden-words.in`
Output file:	`forbidden-words.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to determine who wins in a game with a dictionary.

You are given a dictionary which is a collection of n different non-empty words consisting of lowercase English letters.

Two players take turns. A turn consists in mentioning a letter. One can not mention a letter which was already mentioned by one of the players. A player loses the game when, after she mentions a letter, it is possible to compose a word from the dictionary using only the mentioned letters. Each mentioned letter can be used several times if needed.

Which player wins the game if both players play optimally?

Input

The first line of input contains an integer n: the number of words. The next n lines contain the words themselves, one word per line. Each word is non-empty and consists of lowercase English letters. It is guaranteed that all the given words are distinct and their total length does not exceed 3 000 000.

Output

If the first player wins, print "`First`". Otherwise, print "`Second`".

Examples

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

Explanations

In the first example, the player who mentions "`a`" loses the game. If both play optimally, all the other letters will be mentioned before that, and when there are no more of them, it will be the second player's turn.

In the second example, the player who mentions the second element of the set $\{\mathtt{a}, \mathtt{d}, \mathtt{e}\}$ loses the game. Until that moment, 24 letters will be mentioned if both play optimally, so it will be the first player's turn.

Problem E. Four Prime Numbers

Input file:	`fourprimes.in`
Output file:	`fourprimes.out`
Time limit:	1 second (*2 seconds for Java*)
Memory limit:	256 mebibytes

You are given an integer n.

How many ways there are to represent it as a sum of four prime numbers?

Input

The only line of input contains a single integer n ($1 \le n \le 10^5$).

Output

Print one number: the number of ways to represent n as a sum of four prime numbers.

Example

fourprimes.in	fourprimes.out
9	4

Note

Number $9 = 2 + 2 + 2 + 3 = 2 + 2 + 3 + 2 = 2 + 3 + 2 + 2 = 3 + 2 + 2 + 2$.

Problem F. Set Intersection

Input file:	`intset.in`
Output file:	`intset.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

Vasya and Petya have two identical sets of cards. Each set consists of N cards. Each card contains one integer written on it. All integers from 1 to N are present in the set.

Vasya randomly chooses L cards from the first set. Petya randomly chooses M cards from the second set. Each choice is equally probable, and Vasya's and Petya's choices are independent. After choosing the cards, Vasya and Petya count the number of integers which are present in both of their selections. If this number is strictly less than K, Petya is declared the winner. Otherwise, Vasya wins.

Your task is to choose the number K in such a way that the game is the most fair. Formally, you must find the maximal K not exceeding N such that the probability of Vasya's victory is at least 0.5.

Input

The first line contains three positive integers N, L and M $(1 \le L, M \le N \le 1\,000\,000)$.

Output

Print the non-negative integer K. You are allowed to solve the problem approximately: your solution will be considered correct if the absolute value of the difference between your answer and the jury's answer does not exceed 10.

Example

intset.in	intset.out
4 1 2	1

Problem G. Medals

Input file:	`medals.in`
Output file:	`medals.out`
Time limit:	3 seconds (*5 seconds for Java*)
Memory limit:	256 mebibytes

One small (but very proud) African nation is preparing to take part in Winter Olympic Games 2014. The best national sportsmen are chosen, now it is the time to choose the best of the best, and to decide who will travel to the sunny city of Sochi and assign an olympic discipline to each sportsman.

It is well known that this will be the first Games with sets of ten medals. Instead of the regular gold, silver and bronze, best ten sportsmen in each discipline will be awarded. The first place winner will receive platinum medal, gold goes for the second place, palladium for third, cesium for forth, next are silver, wolfram, nickel, copper, magnesium, and lastly, for tenth place one receives a medal made from aluminum.

After a long set of trainings, we know what disciplines is each sportsman best in and what type of medal he (or she) will get in each of these disciplines. Your task is to distribute sportsmen among disciplines so that they will receive as many platinum medals as possible. From the distributions with equal amount of platinum medals, select one with the maximum number of gold medals, and so on down to aluminum. Rules of the Games don't allow you to send two or more sportsmen to one discipline and you cannot send one sportsman to two or more disciplines.

Input

The first line of input contains one integer n: number of sportsmen. The next n lines contain descriptions of sportsmen, one per line. Description of i-th sportsmen consists of an integer k_i (number of disciplines i-th sportsman can take part in) followed by k_i pairs of integers: number of discipline and type of medal this sportsman will receive in this discipline. Type 1 means platinum medal, 10 means aluminum.

Limits: $1 \le n \le 1000$, $k_i \ge 1$, $\sum_{i=1}^{n} k_i \le 10\,000$, discipline numbers are from 1 to 1000, medal types from 1 to 10.

Output

On the first line of output, print ten numbers: the amount of medals of each type the nation can receive, starting from platinum. On the second line, print n numbers: for each sportsman print the number of discipline he (or she) should take part in, or zero if corresponding sportsman shouldn't take part in any. If there is more than one optimal solution, print any one of them.

Example

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

Problem H. Reachability

Input file:	`reachability.in`
Output file:	`reachability.out`
Time limit:	3 seconds (*8 seconds for Java*)
Memory limit:	256 mebibytes

You are given an oriented graph of n vertices. Initially there are no edges.

We will denote an edge from a to b as (a, b).

You have to process the following four types of queries:

1. "`+ o` v k a_1 ... a_k": add edges $(v, a_1), \ldots, (v, a_k)$
2. "`+ i` v k a_1 ... a_k": add edges $(a_1, v), \ldots, (a_k, v)$
3. "`- o` v k a_1 ... a_k": delete edges $(v, a_1), \ldots, (v, a_k)$
4. "`- i` v k a_1 ... a_k": delete edges $(a_1, v), \ldots, (a_k, v)$

In each query, all a_i are different.

If you have to add an edge, it is guaranteed that there was no such edge in the graph.

If you have to delete an edge, it is guaranteed that there was such edge in the graph.

Also it is guaranteed that at each moment of time, the graph has no directed cycles.

You task is to maintain the matrix of reachability:

$$a_{ij} = \begin{cases} 1, \text{ if } i \neq j \text{ and from } i \text{ there is a path to } j \text{ using edges of the graph} \\ 0, \text{ otherwise} \end{cases}$$

Input

The first line of input contains integers n, the number of vertices in the graph, q, the number of queries, and also A and B which will be useful for output ($1 \le n \le 400$, $1 \le q \le 800$, $1 \le A, B \le 10^9$). Next q lines contain queries in the format described above, one per line.

Output

After each query, output the value of expression

$$\left(\sum_{i,j=0}^{n-1} a_{ij} A^i B^j \right) \bmod 2^{32}$$

on a separate line.

Example

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

Problem I. Revolving Lasers

Input file:	`revolving-lasers.in`
Output file:	`revolving-lasers.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

In this problem, you have to move the robot around the plane so as to evade lasers for an arbitrary amount of time.

There are n lasers on the two-dimensional plane. Each laser consists of a source installed at a point with even integer coordinates, and an infinite straight laser beam starting at that point. A laser number i turns uniformly by t_i degrees per second: a *negative* number means turning clockwise, a *positive* one means turning counter-clockwise. Initially, all lasers are directed upwards, collinear to Oy axis. No two sources are installed at a common point. Lasers do not interfere with each other, so that the beams and sources do not prevent propagation of other beams.

We control a robot which is initially located at the origin. During each second, the robot either stays still or moves uniformly and rectilinearly towards one of the eight neighbor points with integer coordinates. Two distinct points with integer coordinates (x_1, y_1) and (x_2, y_2) are considered neighbors if both their coordinates differ by at most one: $|x_2 - x_1| \le 1$ and $|y_2 - y_1| \le 1$.

Your task is to construct a route such that the robot can move along that route and evade the lasers for an arbitrary amount of time, or determine that there is no such route. Evading the lasers means that the robot can not be located on a laser source and can not touch or cross a laser beam. The sizes of objects and the thickness of the beam should be neglected.

Input

The first line of input contains an integer n: the number of lasers ($0 \le n \le 3$). The next n lines contain descriptions of lasers, one per line. Each laser i is defined by three space-separated integers x_i, y_i and t_i: the coordinates of the laser source and the turning speed in degrees per second ($2 \le x_i, y_i \le 10$, x_i and y_i are even, $-3 \le t_i \le 3$). It is guaranteed that no two laser sources coincide.

Output

Print a non-empty string S consisting of no more than 10 000 characters which corresponds to the route you found. Each character S_i of the string specifies the action of the robot during i-th second. After performing all actions in the route, the positions of lasers and the robot must correspond to their initial positions. The characters used to encode the direction of movement correspond to the positions of arrows on a numeric keypad: when the robot is at point (x, y),

digits

7	8	9
4	5	6
1	2	3

mean moving to points

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

.

If there are several possible answers, print any one of them. If it is impossible to evade the lasers for an arbitrary amount of time, print the word “`None`” instead of a route.

Examples

`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`

Explanations

In the first example, the route consisting of 360 steps is shortened for readability: the fragments of the form “*ddd..(X)..dd*” mean repeating the digit d exactly X times. The robot walks around the single laser source in counter-clockwise direction and returns to the origin.

In the second example, it is impossible to evade the lasers for an arbitrary amount of time.

Problem J. Snakes on the Stone

Input file:	`snakes2.in`
Output file:	`snakes2.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

Several three-dimensional snakes lie on a large stone. Each of them holds its tail in its mouth. The stone is flat, it has Cartesian coordinate system, so it's divided into square cells with side of 1.

The projection of each snake on the stone is a sequence of cells. Any two neighbouring cells in this sequence are distinct and share a side, and it holds also for the last cell and the first cell (the tail and the head of a snake).

We say a snake is *passing a cell t* times if this cell is repeated t times in the sequence of cells into which this snake is projected.

Let's call a cell an *intersection* if it is passed by more than one snake, or if it is passed by one snake more than once. A snake is passing an intersection *from above* if the corresponding part of the snake is farther from the stone than other parts of snakes that are projected into that intersection. By analogy, a snake is passing an intersection *from below* if the corresponding part of the snake is closer to the stone.

The snakes lie on the stone in such a way that there exist only two types of intersections. The first type is an intersection that is passed by the only snake for two times: once from above and once from below. The second type is an intersection that is passed by two snakes, one from above and one from below. Besides, neither pair of intersections is located in adjacent cells, and all the snakes cross intersections straight.

Let i-th snake pass intersections r_i times in total. It may pass some of them once, and some of them twice. Each time it passes an intersection either from above (denote it "+") or from below (denote it "-"). We will specify a way of passing intersections by the snake with the line that consists of symbols "+" and "-", recorded for all the intersections met traversing the snake from head to tail.

Consider k sequences of cells corresponding to k snakes ($1 \le k \le 3$). The total number of intersections for these sequences does not exceed 7, and each snake passes an intersection at least once. The coordinates of cells lie within range from 1 to $m = 25$.

You have to find such a way of passing intersections that the snakes holding their tails in their mouths can take such positions in space that their projection to the stone would be k non-intersecting circumferences (neglecting snakes' width).

Input

The first line of input contains one integer k, the number of snakes ($1 \le k \le 3$). Each of the following k lines describes one snake. The description of snake i starts with an integer l_i: the length of the snake ($4 \le l_i \le m^2$). It is followed by l_i pairs of integers $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$, …, x_{i,l_i}, y_{i,l_i}: the coordinates of cells that are passed by the snake, given in the order of traversal ($1 \le x_{i,j}, y_{i,j} \le m$).

Output

Output k lines containing symbols "+" and "-"; i-th line should specify a way to pass intersections for the i-th snake. Output has to be correct, so that each intersection is passed by snakes once from above and once from below.

If there exist multiple answers, output any of them.

Examples

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
+-+++---

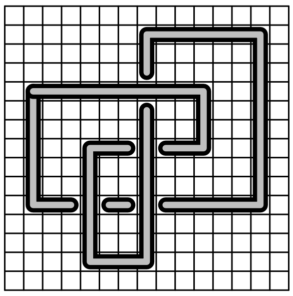
snakes2.in

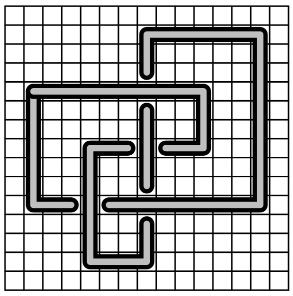
2
36 5 2 5 3 5 4 5 5 5 6 5 7 5 8 5 9 5 10 5 11 5 12 5 13 5 14 5 15 5 16 5 17 6 17 7 17 8 17 8 16 8 15 8 14
8 13 8 12 8 11 8 10 8 9 8 8 8 7 8 6 8 5 8 4 8 3 8 2 7 2 6 2
36 2 5 2 6 2 7 2 8 2 9 2 10 2 11 2 12 2 13 2 14 3 14 4 14 5 14 6 14 7 14 8 14 9 14 10 14 11 14 11 13 11
12 11 11 11 10 11 9 11 8 11 7 11 6 11 5 10 5 9 5 8 5 7 5 6 5 5 5 4 5 3 5

snakes2.out
--++ +--+

Explanations

The length and coordinates of the cells of each snake are placed in one line. In the problem statement, that line is broken into several lines for readability.

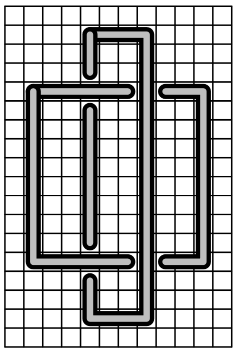


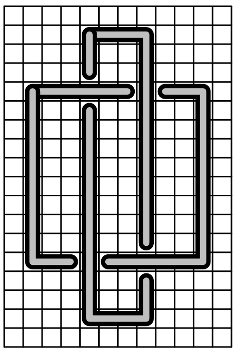


In the first example, there is the only snake. The answer to the example corresponds to the left picture, where the snake is easily untangled. And, for example, the path

```
+-+-+-+-
```

corresponds to the right picture, and this answer would be incorrect.





In the second example, the answer also corresponds to the left picture. One rectangle can be pulled out from another and so the snakes are untangled.

And, for example, the path

```
-+-+
-+-+
```

corresponds to the right picture, and this answer would be incorrect.

Problem K. Dependent Subsets

Input file:	`subset.in`
Output file:	`subset.out`
Time limit:	2 seconds (*3 seconds for Java*)
Memory limit:	256 mebibytes

You are given n vectors in d-dimensional space. You have to choose a set of vectors A of maximal size in such a way that:

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

Input

The first line contains integers n and d ($1 \le n \le 1000$, $1 \le d \le 10$). Each of the following n lines contains d integers $a_{i,1}, \ldots, a_{i,d}$: the vectors themselves. It is guaranteed that $|a_{i,j}| \le 10^4$ and $\forall i \sum\limits_{j=1}^{d} a_{i,j}^2 > 0$. The vectors are numbered by integers from 1 to n in the order they are given.

Output

On the first line, print k: the number of vectors in the set you found. On the second line, print k different integers from 1 to n: the numbers of vectors in the initial set. If there are several optimal answers, output any one.

Example

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