Problem A. Advanced Compass

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

Mars lacks global magnetic field. Advanced gadgets can detect weak local magnetic fields and serve as compasses for navigation.

A compass of this kind depends of movement of a highly magnetized diamond crystal in an intricate maze inside the gadget.

A compass maze is built on a system of concentric rings located on a flat disk with a single so-called base point on its perimeter. On each ring, there are located 360 so-called principal points in regular distances from each other. The angle of a principal point is the angle between two specific rays, measured in degrees. The first ray points from the maze center to the base point, the second ray points from the center of the maze to the principal point. The angle of the first principal point on a ring is 0 degrees, the angle of the next principal point in the clockwise direction is 1 degree, and so on.

The maze itself consists of circular arc segments and straight radial segments. A circular arc segment is always completely embedded in one of the rings. It begins in a principal point and it ends also in a principal point. Each radial segment connects two neighbour rings and it is perpendicular to the tangents of the rings at the respective points of connection. Often, at least one endpoint of a radial segment belongs to a circular arc segment too, but that is not a strict necessity. Any two segments can intersect only at a principal point. No two segments overlap, not even partially.

Before calibration of a compass is started, two points in the maze are given, a principal start point and a principal end point. The diamond is located in the principal start point. The diamond has to travel from the principal start point, move along the segments in the maze, and finally stop at the principal end point.

During calibration, the compass is hanged vertically on a wall, with the base point initially at the top of the compass. Calibration proceeds in phases. In a phase, the compass remains fixed, and the diamond falls through the maze, from its previous position to the lowest currently reachable point in the maze. The phase is terminated when the diamond stops. The diamond may also remain still during a phase, if its position in the maze prevents it from movement. Between each two consecutive phases, the compass is rotated clockwise or counterclockwise by 1 degree.

Most of the time, due to the maze shape, the diamond cannot move exactly vertically. At any point of time, the vector of diamond movement can deviate from vertical vector by at most 89 degrees.

Note that at the end of each phase the diamond is in some principal point. When the diamond arrives at a principal point from which two segments continue currently downwards, it always checks the radial segment. If the current vector of diamond movement through the radial segment would deviate from the vertical by at most 45 degrees, the diamond falls into the segment and continues its movement down through it. Otherwise, the diamond continues its present movement through the circular segment.

Your task is to find the minimum number of the compass rotations between the phases to bring the diamond from the principal start point to the principal end point.

Input

The first line contains a single number N ($1 \le N \le 20$) — the number of rings in the maze.

Then, N pairs of lines follow. Each pair describes one ring, in the order from the center of the maze (the rings are numbered 0, 1, …, $N-1$). The first line in a pair starts with integer K, the number of maze circular arc segments on the current ring. It is followed by K pairs of integers (X and Y) representing an arc starting in a principal point whose angle is X and ending in principal point whose angle is Y. The second line in the pair contains integer L, the number of radial segments leading from the current circle out to the next ring. This number is followed by L values, the angles of principal points on which the radial segments are located.

Last two lines of input represent the principal start point and the principal end point, respectively. Each point is described by the number of ring it lies on and the angle of the principal point it coincides with.

Output

Print the minimum number of compass rotations between the phases which bring the diamond from the principal start point to the principal end point. The diamond must be standing still in the principal end point at the end of the last phase. Print the string "`Impossible`" if the diamond cannot be brought to the principal end point at the end of any phase.

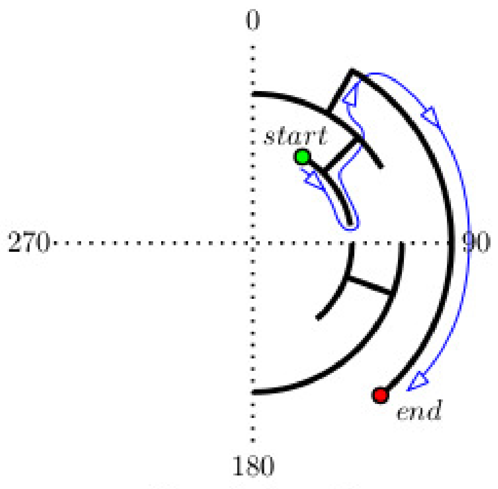
Examples

standard input	*standard output*

```
3
2 30 80 90 140
2 45 110
2 0 60 90 180
1 30
1 30 140
0
0 30
2 140
```
	260

```
3
2 30 80 90 10
2 45 110
2 0 60 90 180
2 30 100
1 30 140
0
0 30
0 10
```
	594

Note



Sample input 1

Problem B. Breaking Bars

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

Selma is visited by her two grandchildren Elsa and Asle who love chocolate. To be precise, they are especially fond of the brand Nut Cream Puffed Chocolate that comes in bars made up by 6×6 squares. The bars can be broken along the valleys between squares into smaller rectangular bars of integer dimensions. Due to the fragile nature of this type of chocolate, the bars often break into smaller rectangular bars even before you unpack them (but still only of integer dimensions).

Thus Selma finds herself with a set of rectangular bars of various dimensions in her candy stash. She knows how important it is to be fair to children, so not only does she want to give Elsa and Asle the same amount of chocolate, but also identical *collections* of rectangular bars (where an $a \times b$ bar is considered identical to a $b \times a$ bar). To do this, Selma can break her bars into smaller pieces. A *break* is the operation of taking an $a \times b$ bar and breaking it along a valley to produce two bars of dimensions $c \times b$ and $(a-c) \times b$, for some integer $c \in [1, a-1]$, or two bars of dimensions $a \times d$ and $a \times (b-d)$, for some integer $d \in [1, b-1]$. See the figure for an example.

Selma would like to give her two grandchildren identical collections of bars, each collection consisting of at least t squares of chocolate. What is the minimum number of breaks she needs to make to be able to do this?

Input

The first line of input contains two integers n and t ($1 \le n \le 50$, $1 \le t \le 900$), where n is the number of bars Selma has, and t is the least number of squares she wants each grandchild to receive. Then follows a line containing n bar descriptions. A bar description is on the format "axb" for two integers $1 \le a, b \le 6$.

You may assume that the total amount of chocolate squares among the n bars is at least $2t$.

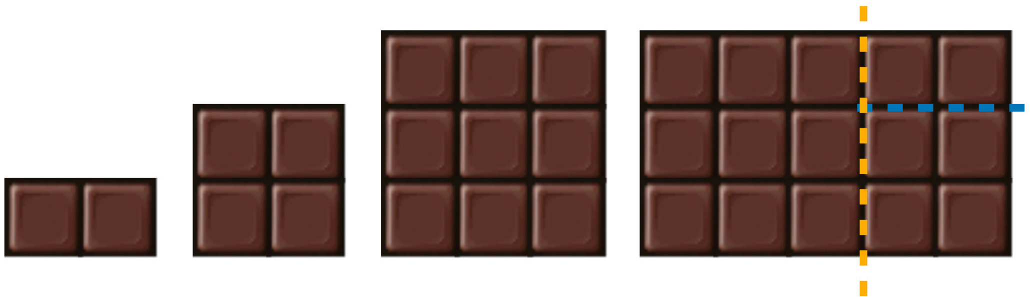
Output

Output the minimum number of breaks needed to obtain two identical collections of bars, each having a total of at least t squares.

Examples

standard input	*standard output*
4 15 1x2 2x2 3x3 3x5	2
6 7 1x2 2x3 1x4 3x2 4x1 6x6	0
5 3 1x1 1x1 1x1 1x1 1x4	1

Note



Explanation of Sample Input 1. First make a vertical break as shown on the 3×5 bar (orange), then make a horizontal break on the newly created 3×2 bar (blue). This way Elsa and Asle can each get one 1×2, one 2×2, and one 3×3 bar, in total 15 squares each.

Problem C. Customs Control

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

With lifted restrictions, the border trade between Norway and Sweden will surely be back to its former glory. But the authorities are worried that this will also mean an increase of illegal smuggling of goods. The customs authorities of Norway and Sweden must cooperate to prevent this from becoming too big of a problem.

To pass through the customs, one must visit a series of checkpoints, the Nordic Customs and Passport Control. There are n checkpoints in total, numbered from 1 to n, where 1 is the entrance and n is the exit. There are m pairs of bidirectional roads that connect distinct checkpoints. The ith checkpoint takes some amount of time t_i to pass through, and this is the bottleneck in crossing the border (the time it takes to walk the roads is negligible).

Each checkpoint can be watched by one customs unit, either a Norwegian one or a Swedish one. There are k Norwegian customs units available, and $n-k$ Swedish units. When a road has both of its endpoints watched by customs units from the same country, any smugglers using that road will be caught. Smugglers are of course always in a hurry, and will always attempt to go from 1 to n in as short amount of time as possible.

Your task is to decide where to put the n customs units, so that any smugglers who take a fastest possible route from 1 to n will be caught.

Input

The first line of input contains three integers n, m, and k ($2 \le n \le 10^5$, $1 \le m \le 2 \cdot 10^5$, $0 \le k \le n$), the number of checkpoints, roads, and Norwegian customs units. The second line of input contains n positive integers $t_1, \ldots, t_n$ ($1 \le t_i \le 10^4$), the time it takes to pass through each checkpoint. Then follow m lines of input each containing two integers u and v ($1 \le u, v \le n$), meaning that there is a road between checkpoints u and v.

It is guaranteed that it is possible to go from any checkpoint to any other checkpoint using the roads. There is also at most one road between each pair of checkpoints, and no road connects a checkpoint to itself.

Output

If there is a way to place the customs units so that every smuggler is caught, output a string of length n, where the ith character indicates which type of customs unit to put at the ith checkpoint (an ‘`N`’ for a Norwegian customs unit, and an ‘`S`’ for a Swedish customs unit). Otherwise, if there if there is no way to catch every smuggler, output “`impossible`”.

Examples

standard input	*standard output*

3 2 0
1 1 1
1 2
2 3
	SSS

2 1 1
1 1
1 2
	impossible

8 9 4
3 3 1 2 2 3 2 1
1 2
1 3
1 4
2 5
3 6
4 7
5 8
6 8
7 8
	NNNNSSSS

Problem D. Dog

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 mebibytes

Let us talk about the big elephant in the room: you've had a enormous dog in your room for a while now and you need to hide it without raising suspicion, since you want to keep the animal. Hiding an animal of this size is difficult: if you use a lot of space, it is obvious that you are hiding something from your landlord. Hence, you want to use as little space as possible to hide the dog.

When the dog was placed against the wall, you took a black and white picture of the animal. Looking around in the house, the only tools you found to hide the dog with were empty tin cans. The dimensions of the tin cans correspond with 2×2 pixels in the picture and these cans cannot overlap. So, you can make a *dog* and if landlord asks why you have cans in the shape of a dog, you simply say it is a bad joke of yours.

The position of each can has to exactly correspond to a block of 2×2 pixels in the picture, and they cannot be shifted or rotated to only partially cover some pixels. Furthermore, cans cannot float in the air, so every can has to be supported either by the floor, which is just below the bottom row of the picture, or by another can, for which at least one of the left and right half must directly rest on another can. The structure does not otherwise need to be balanced.

What is the minimum number of cans needed to hide the dog?

Input

The input consists of:

- One line containing two integers n ($2 \leq n \leq 100$) and m ($2 \leq m \leq 10$), the height and width of your room. Both n and m are even.
- n lines, each containing m characters that are either '`.`' or '`#`', where '`#`' marks a position that needs to be hidden by a can.

Output

Output the minimal number of 2×2 cans that is required to hide the dog in the room.

Examples

standard input	*standard output*
4 4	3
....	
..#.	
.##.	
##..	
4 4	4
#.#.	
....	
#...	
....	
14 8	15
....	
....##..	
...###..	
....##..	
.....#..	
...####.	
..#####.	
.######.	
.#####..	
.###....	
.##.....	
..##....	
...#....	
.###....	

Problem E. Eavesdropper Evasion

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	1024 mebibytes

Alice wants to send n messages to Bob over a communication channel. The ith message takes t_i time steps to send. At each *integer* time step, Alice can start sending any number of her messages. Once started, a message must be transmitted in its entirety (it cannot be paused and resumed later). Any number of messages can be sent in parallel over the channel without affecting the transmission time of individual messages.

An attacker has the capability to disable the security protocols of the channel for an interval of x continuous time steps, but only once (i.e., after doing this, they cannot wait a while and then disable it for another x time steps). While the security is disabled, the attacker is able to listen in, and any message that is sent *in its entirety* during those x time steps is considered exposed.

What is the minimum time needed for Alice to send all n messages to Bob so that *at most* two messages are exposed, no matter when the attacker chooses to disable the security?

Input

The first line of input contains the two integers n and x ($1 \le n \le 20\,000$, $1 \le x \le 10\,000$), the number of messages Alice wants to send and the number of time steps someone may listen in. This is followed by a line containing n integers $t_1, \ldots, t_n$ ($1 \le t_i \le 10\,000$), the number of time steps it takes to transmit each message.

Output

Output the minimum number of time steps to complete transmission of all n messages so that at most two of them can be exposed.

Examples

standard input	*standard output*
6 10 2 3 4 5 6 7	16
7 6 9 3 2 3 8 3 3	11

Note

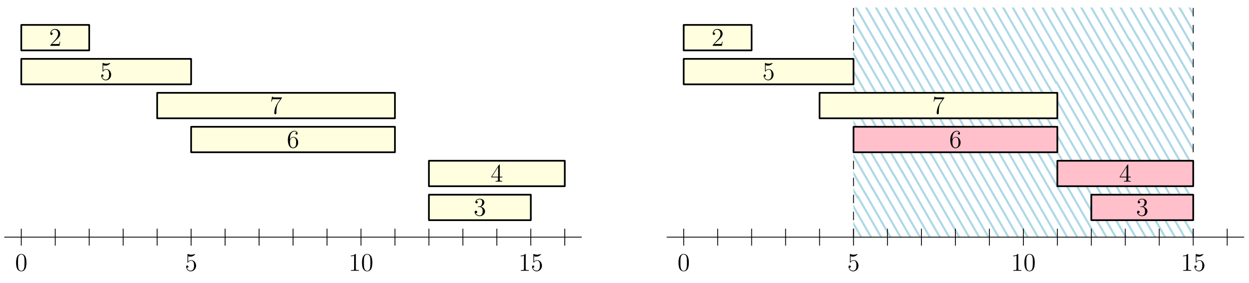


Illustration of a solution to Sample Input 1. Right: sending the message of length 4 a time step earlier would not be a solution, because the three messages of length 6, 4, and 3 would then be exposed to an eavesdropper listening in from time step 5 to time step 15.

Problem F. Fortune From Folly

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

Your friend Ómar's favourite video game is *Striker-Count*. But he has now grown tired of actually playing the game and is more interested in the lootboxes found in the game.

Inside each lootbox there is an item of some level of rarity. Ómar is only interested in acquiring the rarest items in the game. When he starts the game, he chooses two numbers n and k, such that $k \leq n$. He then opens lootboxes in the game until k of the last n lootboxes included an item of the highest rarity. However, if he will find k lootboxes before he will open first n, he stops opening the lootboxes immediately.

This activity amuses Ómar, but does not interest you in the slightest. You are more interested in the numbers: you know that each lootbox Ómar opens has probability p of containing an item of highest rarity, independently for each lootbox.

You want to find the expected number of lootboxes Ómar will open before concluding his process.

Input

The only line of the input contains the two integers n and k ($1 \leq k \leq n \leq 6$), and the real number p ($0 < p \leq 1$ and p has at most four decimals after the decimal point), with meanings as described above.

Output

Output the expected number of lootboxes Ómar must open, with an absolute or relative error of at most 10^{-6}. It is guaranteed that the input is such that this expected number does not exceed 10^9.

Examples

standard input	*standard output*
`3 2 0.0026`	`74445.39143490`
`6 1 0.0026`	`384.61538462`

Problem G. Gyrating Glyphs

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

This is an interactive problem

You are rocking the latest breakthrough in Computer Science: animated fonts. Suddenly, all of your colleagues' code looks amazing, and you are finally motivated to review it. Unfortunately, due to the constant rotations, it is hard to distinguish between the $+$ (plus) and the $\times$ (multiply) operators (all the other characters are still readable). The function you are reviewing takes as input $n+1$ integers $a_0, a_1, \ldots, a_n$ and returns the value

$$\left(\ldots\left(\left(\left(a_0 \text{ op}_1 \ a_1\right) \text{ op}_2 \ a_2\right) \text{ op}_3 \ a_3\right) \ldots \text{ op}_n \ a_n\right) \quad \bmod 10^9+7,$$

where the n operators $\text{op}_1, \text{op}_2, \ldots, \text{op}_n$ are either $+$ or $\times$. For example when given input $(a_0, a_1, a_2) = (1, 1, 2)$ with hidden operators $(\text{op}_1, \text{op}_2) = (+, \times)$, then the function returns $((1+1) \times 2) = 4 \bmod 10^9+7$.

You can still execute the function a few times on some input and read the returned value. Use this to recover the operators.

Interaction Protocol

This is an interactive problem. Your submission will be run against an *interactor*, which reads the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

The interactor first sends one line containing one integer n ($1 \le n \le 4000$), the number of hidden operators.

Then, your program should make at most 275 queries to determine the operators. Each query is made by printing one line of the form "`?` a_0 a_1 `...` a_n" ($0 \le a_i < 10^9+7$). The interactor will respond by printing one line with an integer, the value of

$$\left(\ldots\left(\left(\left(a_0 \text{ op}_1 \ a_1\right) \text{ op}_2 \ a_2\right) \text{ op}_3 \ a_3\right) \ldots \text{ op}_n \ a_n\right) \quad \bmod 10^9+7.$$

Make sure you flush the buffer after each write.

When you have determined the operators, print a single line of the form "`!` s", where s is a string consisting of exactly n characters, which are all "`+`" (plus) or "`x`" (multiply)[1]. The ith character of this string should be op_i. This line does not count as one of your queries.

Using more than 275 queries will result in a wrong answer verdict.

[1]This is the lowercase letter "`x`", not the Unicode "$\times$" symbol.

Examples

standard input	*standard output*
2	
	`? 1 1 2`
4	
	`? 1 1 3`
6	
	`! +x`
10	
	`? 1 1 1 1 1 1 1 1 1 1 1`
5	
	`? 0 4 2 4 2 4 2 4 2 4 2`
6224	
	`? 1 2 3 4 5 6 7 8 9 10 11`
640750	
	`! ++xxx+x+xx`

Problem H. Hiring Help

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

A certain large unnamed software development company has n developers. The productivity of each coder working for the company has been rigorously measured in terms of two key performance indicators: the number of lines of code they write per hour, and the number of bugs they fix per hour.

When a project needs to be done, the manager in charge of the project is allocated some budget of t man-hours of programmer time. The manager can then staff different coders on the project, up to a total of t hours. For instance if there are three programmers, the manager can allocate any non-negative real numbers t_1, t_2, and t_3 hours of their respective work hours, as long as $t_1 + t_2 + t_3 \leq t$. If the three programmers write l_1, l_2, and l_3 lines of code per hour, a total amount of $t_1 \cdot l_1 + t_2 \cdot l_2 + t_3 \cdot l_3$ lines of code will then be written for the project. Similarly if they fix b_1, b_2, and b_3 bugs per hour, a total of $t_1 \cdot b_1 + t_2 \cdot b_2 + t_3 \cdot b_3$ bugs will be fixed.

Due to the uncertain economy, the company has a hiring freeze, meaning that no new coders are hired to the company. However, under certain conditions, a manager is allowed to bring in outside help by outsourcing a project to an external consultant rather than doing it in-house. But this is only allowed if it is not possible to do the project equally efficiently in-house. In particular, if the consultant writes ℓ lines of code and fixes b bugs in t hours, and there exists some allocation of our existing coders which would write at least ℓ lines of code *and* fix at least b bugs in at most t hours, then a manager is *not* allowed to hire this consultant (regardless of whether those existing coders would actually have time to work on the project or whether they are already too busy with other projects).

While no new coders are hired, employees do sometimes decide to leave the company. Given a chronological list of events – requests to use a consultant, and employees quitting – find out which of the requests will be approved.

Input

The first line of input consists of a single integer n ($0 \leq n \leq 2 \cdot 10^5$), the number of coders (initially) at the company. The employees are numbered from 1 to n (names are too personal). Then follow n lines, the ith of which contains two integers ℓ_i and f_i ($1 \leq \ell_i, f_i \leq 10^8$), the number of lines of code and the number of bugs fixed per hour by coder i.

Next follows a line with a single integer e ($1 \leq e \leq 10^5$), the number of events. This is followed by e lines, describing the events in chronological order. An event is a line in one of the following two forms:

- "`c` t ℓ f", for three integers t, ℓ and f ($1 \leq t \leq 100$, $1 \leq \ell, f \leq 10^8$): a request to take in a consultant for a project of t hours, where the consultant would write ℓ lines of code and fix f bugs in those t hours.
- "`q` i", for an integer i ($1 \leq i \leq n$): coder i quit the company.

You may assume that no coder quits more than once.

Output

For each request to take in a consultant, output "`yes`" if the request is approved, and "`no`" if it is not approved.

Examples

standard input	*standard output*
4 200 100 100 200 100 100 200 200 5 c 10 2000 2000 c 5 750 750 q 4 c 3 600 600 c 10 1500 1500	no no yes no
8 400 300 300 200 300 400 200 300 500 500 100 500 100 100 500 100 12 c 4 1611 1601 c 3 602 601 c 2 399 795 c 1 395 206 q 7 q 6 q 5 q 4 c 4 1611 1601 c 3 602 601 c 2 399 795 c 1 395 206	no no no no yes no no no

Problem I. Intact Intervals

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	1024 mebibytes

Gustav is an astronaut on the Nordic Celestial Planetary Craft, a large space station in orbit around Mars. Today, one of Gustav's tasks is to look over the satefy routines on board.

The space station consists of n modules arranged in a circle, so that module i is connected to module $i+1$ for $i = 1 \ldots n-1$, and module n is connected to module 1. Each module i has a non-negative integer type a_i, representing the kind of equipment that can be found there. Different modules can have the same type. In case of emergency, the equipment must be rearranged so that each module i instead gets type b_i, for some list $b_1, b_2, \cdots, b_n$. Here, the list b is a rearrangement of the list a.

Gustav has noticed that if some module connections are severed, causing the space station to split into separate parts, it may become impossible to perform this rearrangement of the equipment. He decides to estimate how likely it is that the safety routines can be followed, by calculating in how many ways the space station can be separated into two or more parts such that it is still possible to rearrange the equipment according to the emergency procedures.

In other words, your task is to count in how many ways the circular list a can be partitioned into *at least two* non-empty contiguous intervals, in such a way that the circular list b can be obtained by rearranging elements within each interval. Since this number can be quite big, you should find its remainder modulo $10^9 + 7$.

For example, consider Sample Input 1 below. Here the list a could be split into [1|223|4], indicating that the connection between modules 1 and 2, and the connection between modules 4 and 5, are severed. Note that the connection between module 5 and 1 remains in this split. The second possible way in which a could be split is [12|2|34].

In Sample Input 2 below, the only possible way to split the list a into at least two non-empty parts is to separate the two modules. But then it is impossible to rearrange the parts to create the list b.

Input

The first line of input contains a single integer n ($2 \le n \le 10^6$), the number of modules. The second line contains the n integers $a_1, \ldots, a_n$ ($0 \le a_i \le 10^9$). The third and final line contains the n integers $b_1, \ldots, b_n$ ($0 \le b_i \le 10^9$).

The list b is guaranteed to be a rearrangement of the list a.

Output

Print one integer, the number of safe separations modulo $10^9 + 7$.

Examples

standard input	*standard output*

5
1 2 2 3 4
4 3 2 2 1
	2

2
1 2
2 1
	0

Problem J. Jail or Joyride

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

A group of criminals has stolen a fast sports car for a Saturday night joyride. The local police department has only one car available to catch the criminals red handed and put them in a jail.

The city consists of a set of junctions and bidirectional roads, each of a certain length. The criminals stay at a certain junction until just before the police car arrives at this junction.

At that moment, the criminals want to get to a junction as far as possible from their current location, without using the road the police car is on. They quickly look at a map to determine all junctions within the city which are reachable without using the road with the police car.

Then the criminals determine the distance to each of these junctions using their satellite system and randomly pick one of the furthest located junctions.

Note that the satellite system does not know about the location of the police car, and will not take it into account when computing the distance.

The sports car then drives instantly to that junction using any route which does not pass by the police car, while the police is left behind dumbfounded.

The gangsters will wait there until the police car makes a new approach. The only way for the police to catch the criminals is by approaching them while they are in a dead end (a junction with only one incoming road).

Since time is precious for the police, they need you to find out if it is possible to catch the joyriders with absolute certainty. And if so, what is the minimal distance they need to drive to be guaranteed to catch the criminals, assuming the police uses an optimal strategy?

Input

The input consists of:

- One line containing four integers: n ($2 \le n \le 300$), the number of junctions, m ($1 \le m \le \frac{n(n-1)}{2}$), the number of roads, p ($1 \le p \le n$) the initial position of the police car, and t ($1 \le t \le n$, $t \ne p$) the initial position of the group of criminals.
- Then follow m lines, each containing three integers a, b and ℓ ($1 \le a, b \le n$, $a \ne b$, and $1 \le \ell \le 10^9$), indicating a road between junctions a and b with a length of ℓ.

There is at most one road between every pair of junctions and you can reach any junction from any other junction by making use of the roads.

Output

If it is possible to catch the criminals with absolute certainty, output the minimal distance that the police car needs to cover to achieve this. Otherwise, output “`impossible`”.

Examples

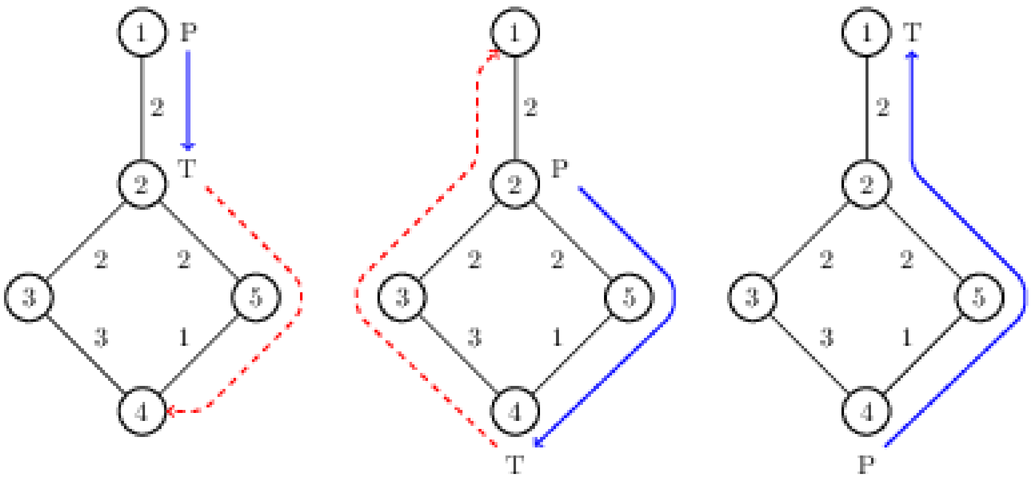
standard input	*standard output*

`5 5 1 2`
`1 2 2`
`2 3 2`
`3 4 3`
`4 5 1`
`2 5 2`
	`10`

`5 5 1 3`
`1 2 2`
`2 3 2`
`3 4 3`
`4 5 1`
`2 5 2`
	`impossible`

Note

Figure shows how the police can capture the criminals in the first sample case.



Possible movements of the police (P) and criminals (T) in the first sample case. The movement of the police (solid blue arrows) takes time according to the length of the edges, while the movement of the criminals (dashed red arrows) is instant.

Problem K. Kinky Cables

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You need to lay a cable to connect two computers with each other. This cable however has a very specific length and you need to use exactly the full length of the cable. Moreover, the cable cannot intersect itself and one part of the cable cannot be too close to another part. Can you connect the two computers with each other using the full length of the cable?

The two computers are standing in an $n \times m$ rectangular two-dimensional room. Computer 1 is always positioned at $(0, 0)$ (the upper left corner) and Computer 2 at (n, m) (the lower right corner). The cable is specified by a sequence of marked points $p_1, p_2, \ldots, p_s$. The path of the cable is then obtained by connecting the consecutive points of this sequence with (straight) line segments. The cable path should satisfy the following constraints:

- None of the line segments within the cable path should intersect.
- The marked points of the path should not be too close to each other: given a point p_i there should be no other marked points strictly within a radius of 1 of p_i, except possibly p_{i-1} and p_{i+1} (the two consecutive points).
- The path should always start at $(0, 0)$ and end at (n, m).
- All points should lie somewhere in the $n \times m$ room.

Input

The input consists of:

- One line with two integers n and m $(2 \leq n, m \leq 100)$, the width and height of the room.
- One line with a floating-point number ℓ $(\sqrt{n^2 + m^2} \leq \ell \leq n \cdot m)$, the length that the cable should have.

Output

Output the number of points k $(2 \leq k \leq 500)$ that the cable path contains, followed by the k points of the path, in their respective order. Each point consists of two floating-point numbers x and y $(0 \leq x, y \leq 100)$, the x- and y-coordinates of this point in the path.

The total length of the path should be exactly ℓ, up to a relative or absolute error of 10^{-4}.

If there are multiple valid solutions, you may output any one of them.

Examples

standard input	*standard output*

`3 4`
`5.0`
	`3`
	`0 0`
	`1.49979 2.00028`
	`3 4`

`3 4`
`7.0`
	`4`
	`0 0`
	`0 4`
	`1 4`
	`3 4`

`5 5`
`11.5`
	`4`
	`0 0`
	`0 5`
	`1 3.15776`
	`5 5`

Problem L. Lopsided Lineup

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 mebibytes

Together with your coworker, Sergey, you are organizing the exciting Billiards and Pool Competition for your coworkers in your small company. However, communication has not been great between you two. You are not sure you and Sergey think alike, but as far as you are concerned, this would be a great opportunity to do some team building. The actual prizes are meaningless, but there is possibly a lot to be gained from this in team bonding. You want to maximise result.

You start reading some pseudo-scientific books on team management, and after some research, you conclude that there are two good ways of team bonding: people feel more connected after either a triumphant victory or a crushing defeat. This gives you a great idea: if you divide your coworkers into two groups that are as far apart in skill level as possible, both teams will experience improved bonding! You therefore think it is optimal to try to make the teams as unbalanced as possible. Make sure, however, that the teams are of equal size.

With a bit of work you come up with a nice model for the strength of a team. You think team strength is mainly determined by how well two players play together, whether they encourage one another and complement each other's weaknesses. Whenever two players i and j are in the same team, they increase the team score by an integer $c_{i,j}$. The total score of a team is thus equal to the sum of $c_{i,j}$, over all unordered pairs of players i and j in the team.

Input

The input consists of:

- One line with an even integer n ($2 \le n \le 1000$), the total number of players.
- n lines, the ith of which contains n integers $c_{i,1}, c_{i,2}, \ldots, c_{i,n}$ ($-10^6 \le c_{i,j} \le 10^6$). For any i and j, it is guaranteed that $c_{i,i} = 0$ and $c_{i,j} = c_{j,i}$.

Output

Output the maximum possible difference in strength between two teams of equal size.

Examples

standard input	standard output

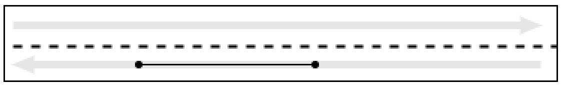
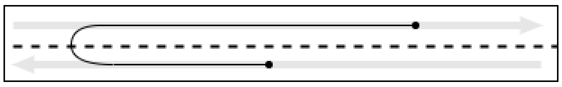
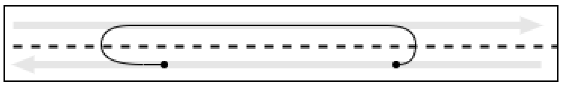
6
0 4 -6 2 3 -3
4 0 2 -6 0 0
-6 2 0 0 2 2
2 -6 0 0 -1 5
3 0 2 -1 0 -4
-3 0 2 5 -4 0
	0

4
0 1 2 2
1 0 8 -3
2 8 0 5
2 -3 5 0
	6

Problem M. Marvelous Marathon

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	1024 mebibytes

A marathon race is being planned in the beautiful countryside. The course will be somewhere along a long, bidirectional, road. The organizers want to determine exactly where along this road the race should be in order to maximize the experience for the runners, so that they get to enjoy as much beautiful scenery as possible and are distracted from their tired limbs. The scenery varies in beauty depending on where on the road you are, and also in which direction you are running. Because of this, the organizers are fine with having the runners make up to two U-turns, as long as no part of the road is used more than once in each direction.

The three different shapes of a valid course: zero, one or two U-turns, respectively.

We model the road of length m meters as a rectangular grid of size $2 \times m$, where each cell has a non-negative “beauty” value associated with it. The columns represent each meter of the road ordered from start to to end. The top row in a column represents the beauty for this part of the road when running in the direction towards the end of the road, and the bottom row in a column represents the beauty when running towards the start of the road. A race of length x is then some set of exactly x of the cells in the grid. Those x cells must form a path in the grid where no cell is visited more than once, we only move to the right or down from cells in the top row, and we only move to the left or up from cells in the bottom row. The sample race you may see in the note. The “total beauty” of a race is the sum of the beauty values of the included cells.

The road is long, so rather than providing a list of all of the $2m$ beauty values, each side of the road is divided into a small number of segments, where the cells within a segment have some constant beauty value (and cells with beauty 0 are simply omitted).

Input

The first line of input contains the three integers m, x and n ($1 \le m \le 10^9$, $1 \le x \le 2m$, $0 \le n \le 200$), the length of the road, the length of the race and the number of segments.

This is followed by n lines describing the segments. Each such line contains three integer a, b, v ($0 \le a, b \le m$, $1 \le v \le 10^9$, and $a \ne b$), describing a segment with endpoints a and b having beauty value v. If $a < b$, this is the segments of cells in the top row of the grid in the range $[a, b)$, and if $a > b$, this is the segments of cells in the bottom row of the grid in the range $[b, a)$.

The parts of the road that are not covered by any segments have beauty value 0. Each cell in the grid is covered at most once (that is, there are no overlapping segments in the same direction).

Output

Output the maximum possible total beauty the race can have.

Examples

standard input	*standard output*

19 14 6
14 5 7
11 15 6
3 7 4
16 15 5
19 17 8
0 3 9
	89

100000 42195 2
30000 60000 500000000
40000 10000 1000000000
	35548500000000

Note

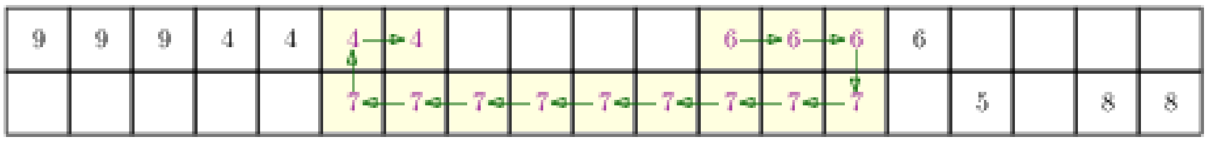


Illustration of Sample Input 1. The numbers in the cells indicate the beauty value for each meter of the road (with omitted values being 0). The highlighted cells and arrows mark the optimal race, involving two U-turns.

Problem N. New Strategy

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are playing your favorite game, Basements and Pigeonlike Creatures, for the umpteenth time. You know the game pretty well, but you have never spent enough time on it to figure out the best strategy. That is, until now. The game consists of a certain sequence of events, such as battling a monster or saving a cat from a tree, and you need to complete all events to win. Attached to each event is an attribute, such as strength, and a threshold, some positive integer. If your attribute score matches or exceeds the threshold, you successfully complete the event! If not, it is unfortunately game over and your total score will be zero.

If you complete all the events successfully, your score depends on how well you did during these events. If your attribute score matches the threshold of an event exactly, you get 0 points, barely scraping by that event. If you exceed the threshold, you get points equal to your attribute score that was used for that event.

You are now at the final part of the game, but first you have some attribute points to spend to increase your attribute scores. You know what events will happen during the final part of the game, so all that is left is to figure out what attributes to increase.

Input

The input consists of:

- One line containing an integer n ($1 \le n \le 10^5$) and an integer k ($1 \le k \le 10^9$), the number of attributes and the number of attribute points you can still spend.
- n lines, each containing a distinct attribute name, and an integer s ($0 \le s \le 10^9$), the current score you have in that attribute.
- One line containing an integer l ($1 \le l \le 10^5$), the number of events.
- l lines each describing one event, containing the name of the attribute that is used, and an integer t ($0 \le t \le 10^9$), the threshold for this event.

Attribute names consist of upper case English letters (`A-Z`), and have a length between 1 and 20 characters inclusive.

Output

Output the maximum score you can get from the events.

Examples

standard input	*standard output*
2 3 STR 15 CON 12 2 STR 17 CON 14	0
3 7 JUMP 5 RUN 7 FLY 0 4 FLY 0 JUMP 6 RUN 10 RUN 8	31