Problem 2. Decorating the Tree

Input file:	`standard input`
Output file:	`standard output`
Time limit:	6 seconds
Memory limit:	512 mebibytes

Every year Vasya has winter holidays from January, 1 for a week or more. He considers it as the most boring time of the year. His university is closed, exams still not started, and TV shows only the oldest movies. Luckily, Vasya has a New Year tree at home and he constantly invents more complicated ways to decorate his tree.

Vasya wants to use n sets and put them to n different places of his tree. More formally, he represnted his New Year tree as a graph theory tree with n nodes. Vasya wants to put exactly one decoration set to every node of the tree. Decoration set for node v_i must be of size c_i.

Vasya can buy decoration sets in the shop next to his house. Any decoration set is defined by two parameters: its *size* and its *base*. Such a set consists of *size* balls with radii *base*, *base*+1, ..., *base*+*size*−1. The set costs $base + size - 1$ rubles, i. e. the price only depends on the radius of the largest ball. For any given positive integers *size* and *base* the shop can quickly deliver any amount of decoration sets.

Vasya has a special requirement to the decoration. He does not want two decoration sets on adjacent nodes to share a ball of the same size. At the same time, he has nothing against having same balls or even same decoration sets on nodes which are not connected with an edge.

Please note, that Vasya has to buy exactly n decoration sets. No decoration set can be split, and no two decoration sets can be merged together.

Please help Vasya to decorate his tree at a minimum cost.

Input

First line of the input contains a single integer, n ($1 \le n \le 2000$). Next line contains n integers, c_i describes required size of decoration set for i-th node ($1 \le c_i \le 10^9$). Next $n-1$ lines describe edges. Each line has format "`u v`" ($0 \le u, v \le n-1$) and means that nodes u and v are connected with the edge.

Output

Print minimum decorating cost on the single line.

Examples

standard input	standard output

5
5 1 5 1 2
0 1
1 2
2 3
2 4
	17

2
1 2
0 1
	4

Problem 3. Palette

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

One not so famous artist Palevich is trying to use up-to-date technologies in his paintings. To make the process of creating his masterpieces easier, he decided to create a palette which contains all combinations of basic colors. In case you are new to this, there are exactly n basic colors. In this problem we will denote them by capital English letters `‘A’`, `‘B’`, `‘C’`, $\ldots$, n-th letter of the alphabet.

He has already decided that his palette will have the form of table $r \times c$, $rc = 2^n$. Each cell of the table should contain unique combination of basic colors. By combination we mean a subset of colors. So there are 2^n combinations in total, including one special combination of zero colors.

Before making a real palette, Palevich is going to see how it will look. He will draw it in his favourite graphics editor. The editor provides some simple tools that Palevich will use to draw the palette. First, he creates n layers, one layer per basic color. In the first layer, he selects simple (without self-crossings and self-intersection) polygon. The edges of the polygon should be parts of grid lines of the table. Then he fills all cells of the table that are inside the selected polygon with color `‘A’`, creating solid connected figure of the color ‘A’. He repeats this process for every other color (in a separate layer), and then merges the layers. This causes colors to mix and create different combinations.

Consider the following example: there are three basic colors `‘A’`, `‘B’` and `‘C’`, and Palevich wants to create palette in 2×4 table. First, he creates three layers and paints them like in the picture.

A	
A	
A	
A	

B	B
B	B

C	C
C	C

Then he merges all layers into one, producing the following palette. It is easy to see that each cell of the table contains unique combination of colors.

A	
A,C	C
A,B,C	B,C
A,B	B

As you see, Palevich has managed to create 2×4 palette, and even a 4×4 one, but he is completely unable to create larger palettes. Your task is to help him.

Input

Input contains two integer numbers r and c ($4 \le r, c \le 256$, $rc = 2^n$ for some integer n).

Output

Print n layers, the first one should correspond to the color `‘A’`, the second layer — to the color `‘B’`, and so on. Print each layer as r lines of c characters each. Print the cells that are inside the filled polygon as letter of corresponding color, all other cells print as `‘.’` (dot). Print a blank line between consecutive layers. If there are multiple solutions, output any of them.

Examples

standard input	standard output
4 4	AAAA AAAA .... BBB. B... BBBB .... .CCC .C.. CC.. CC.. .D.. DDD. D.D. D.D.

Problem 5. Garbage problem

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

The inhabitants of San Francisco have realized that there is a huge garbage problem and someone should immediately start thinking about the solution. But first let's understand what is going on. Scientists concluded that there were several dirty regions in the city and these regions were growing every second. Now the problem that the scientists need to solve is to find out when the garbage areas would separate the city into non-connected components.

In this problem we can assume that the city is a polygon without holes, self intersections and touching. In the beginning each dirty region is a circle with zero radius. All dirty regions are growing with the speed 1, that is at time t radius of each region is t.

You need to figure out the first moment of time when there would be more than one clean region in the city and they would not be connected to each other. And you need to find out the number of components as well.

Input

The first line of the input contains integer n ($3 \leq n \leq 100$) — the number of vertices of the polygon that represents the city. Next n lines contain the points of the polygon in the traverse order.

Next line contains integer k ($1 \leq k \leq 20$) — the number of dirty regions. Next k lines represent the centers of dirty regions, one point on each line. Center of each dirty region is either inside the polygon or on the border of the polygon.

All coordinates are integers in range $[-10^5, 10^5]$.

Output

On the first line of the output write "`Never`" if the event never happens, or "`Yes`" if it is possible.

For the "`Yes`" output, second line should contain first moment of time when the city becomes disconnected. Absolute or relative error of the answer must not exceed 10^{-5}. The third line should contain the number of disconnected components.

Examples

standard input	standard output

`4`
`0 0`
`0 5`
`9 5`
`8 0`
`1`
`2 4`
	`Yes`
	`2.0`
	`2`

`4`
`0 0`
`8 0`
`8 4`
`0 4`
`2`
`4 1`
`4 3`
	`Yes`
	`1.0`
	`2`

`3`
`0 0`
`7 2`
`2 4`
`1`
`2 4`
	`Yes`
	`3.2966535347685366`
	`2`

Problem 7. Reverse engineering

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

As you know, San Francisco is the center of Software Engineering. Most difficult problems are solved there. One of the companies found the problem which they still cannot solve. Help them, please.

The description is simple: "Black box". There is some program that gets something as input and produces something to the output. You are given this black box (in the supplementary problem **97**) and some sample tests just to start with.

The only requirement that you know is that the "Black box" accepts only strings no longer than 1000 characters. You need to repeat the logic of the "Black box".

Input

A string with the length no more than 1000 and valid ASCII code characters from 32 to 127.

Output

Your output for each test cases should be the same as the "Black box".

Examples

standard input	standard output
`int foo;`	`Declare foo as int`
`double *arr[23];`	`Declare arr as array 23 of pointer to double`
`double (*a)(int, double (*)()))(int);`	`Incorrect input`

Note

Start with simple tests to find out what is valid and what is not.

Problem 11. Cover WiFi

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

San Francisco is amazing city! It is located near the ocean and you can easily go for a walk on the beach. All important locations are connected with bridges. And as you can assume, there are lots of cars on the bridges during the business hours. The traffic is awful.

Government of California decided that it is not perfect that people in their state are upset about the traffic. So the idea came suddenly: WiFi! If people could serf in the web while stuck in the traffic it would make them happy! But nobody likes spending more money than it's needed. So Government has decided to set up only one hotspot and now they need to determine the location.

You could imagine that all the problematic locations are just points on the plane. Government has a plan to set up the hotspot in the way to cover not less than k problematic points. Of course, they want to use the least powerful hotspot as possible (power of hotspot would be measured by the radius of coverage). So your task is to find the minimal possible radius to cover at least k problematic locations, and the location of the hotspot as well.

Input

The first line contains two integers n and k ($2 \le n \le 10000, 2 \le k \le min(50, n)$) — the number of problematic locations and minimum number of locations you need to cover. Next n lines contain coordinates of the locations — integers x, y ($-10000 \le x, y \le 10000$). There are no two equal locations.

Output

On the first line output the minimum radius of the hotspot with relative or absolute error no more than 10^{-5}. Next line should contain the coordinates of the location with the same acceptable error.

Examples

standard input	standard output

5 4
0 0
1 1
0 1
1 0
10 10
	0.7071067813839326
	0.5000000000000001 0.4999999997208554

3 3
0 0
1 1
0 1
	0.7071067813839326
	0.49998818593030947
	0.5000118140696904

Problem 13. Anti-Lines

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	256 mebibytes

In this task, we will tell about the classic version of the game “Lines”. There is a square field of 9×9 cells. Each cell can be empty or contain a single ball of one of 7 colors. Each turn the computer puts three balls of random colors in random empty cells. After that player can move one ball in any empty cell. If 5 or more balls of the same color arranged in a vertical , horizontal or diagonal lines, these balls are destroyed, and the player gets points and an extra turn.

In this task, everything is a bit different: we hacked the computer and are now able to select empty cells on each turn, which will be occupied by new balls, and colors of these new balls. Two new balls appear on each turn. Player doesn’t move anything (because we are directing the actions of the computer). Our goal — to make such moves, that allow us on the first destruction to destroy as many balls as possible. That is, you can make a few moves to build some successful design, and then destroy it in one move . You cannot make additional destructions to clear map.

Two balls are put in the field simultaneously. Thereafter, any ball that belongs to the vertical, horizontal or diagonal line of at least 5 balls of the same color, instantly collapses. If there are less than two empty cells, the next turn is not available.

Input

Input contains exactly 9 rows by 9 characters each — the current position in the game. Each character is either a digit from 1 to 7 , meaning the color of the ball in the current cell, or character ‘.‘ if the cell is empty. It is guaranteed that there is no line with 5 or more balls of the same color.

Output

The first line of output should contain two numbers D and M — maximum number of balls that can be destroyed by one stroke and the number of moves that lead to this destruction. $2M$ lines follow(2 lines on each turn). Each line should contain three numbers r_i, c_i and $color_i$ ($1 \le r_i, c_i \le 9$ $1 \le color_i \le 7$) — number of row and column of empty cells, where ball of $color_i$ color was placed. Both ball of one turn were placed at the same time. At the time of first destruction you must destroy exactly D balls (theoretically its possible to destroy them in a few times and make extra moves, but this is not recommended).

If no one ball can be destroyed, D must be 0. M does not exceed 100.

Examples

standard input	standard output
723134552	25 9
2421.2456	6 7 2
353...442	3 4 2
14...2..4	5 8 2
23.2....4	5 7 2
4....1..1	5 3 2
1222..112	6 3 2
3621..124	3 6 2
631211777	8 5 2
	7 5 2
	6 5 2
	5 5 2
	2 5 2
	3 5 2
	4 8 2
	4 7 2
	4 4 2
	5 6 2
	4 5 2

Problem 17. Building a Cube

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	64 mebibytes

A 3D plane is defined by an equation $Ax + By + Cz + D = 0$. You have to build a cube with vertices located on given distances $d_1, d_2, \ldots, d_8$ from this plane.

Input

The first line contains coefficients of the plane equation of the plane A, B, C and D, and the second line contains distances — d_1, d_2, $\ldots$, d_8.

All values are real, and do not exceed 10^4 in absolute value and have no more than two digits after the comma. At least one number of A, B and C are nonzero, all the d_i values — nonnegative.

Output

Output 8 lines, each of them contains three numbers — coordinates x, y and z of the corresponding cube vertices. The first printed point should be at the distance d_1 from the plane (with accuracy up to 10^{-5}), the second one — at the distance d_2 and so on.

Examples

standard input	standard output

1.0 1.0 1.0 1.0
1.0 3.0 2.0 4.0 2.0 3.0 3.0 2.0
	0.244017 0.244017 0.244017
	1.976068 1.976068 0.244017
	1.976068 0.244017 0.244017
	1.976068 1.976068 1.976068
	0.244017 1.976068 0.244017
	1.976068 0.244017 1.976068
	0.244017 1.976068 1.976068
	0.244017 0.244017 1.976068

-1 2 -3 6 0 0 0 0 0 0 0 0	Impossible

Problem 19. Pizza

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	512 mebibytes

Little Vitechka got a job at pizza delivery. When he started his work, he found a problem: the company in which he works produces pizzas and boxes for them of very unusual form. Their forms can be arbitrary convex polygons. To start the delivery, Vitechka has to fit all of the pizzas into their boxes. While doing so, he can't flip the pizzas (but he can rotate them).

Help Vitechka to solve this problem. For every pizza, determine if it is possible to fit it in its box, and if yes, find out how to do that.

Input

The first line contains an integer t ($1 \le t$), the number of pizzas. It is followed by t blocks. Each block describes a pizza and its box.

The first line of a block contains an integer n ($3 \le n \le 30$), the number of vertices in the polygon describing the shape of the pizza. Each of the next n lines contains two real numbers not exceeding 10^6 by the absolute value, which are coordinates of a single vertex of the polygon. The vertices are given in the counter-clockwise order. The next line contains an integer m ($3 \le m \le 30$), the number of vertices in the polygon describing the shape of the box. Then follows the description of the box shape in the same format.

All real numbers in the input contain no more than twenty decimal digits after the point.

The total number of vertices of all polygons does not exceed 10^3.

It is guaranteed that, if we move each point of a block by distance at most 10^{-3} in any direction, the test will still be correct, and the possibility to fit the pizza into its box will not change.

Output

For each block, if it is impossible to fit the pizza into the box, print "`NO`" on a single line. Otherwise, on the first line, print "`YES`", and on the second line, print three real numbers describing the motion of the pizza. The first number must be angle by which the pizza must be rotated around the origin. The second and the third numbers are the coordinates of the vector, by which the pizza should be moved afterwards. After applying the described motion, the pizza should fit into the box. The pizza is considered inside the box if each of its vertices is located at distance no more than 10^{-3} from the box.

Example

standard input	standard output

2
3
0 0
2 0
0 2
3
3 0
3 3
0 3
3
3 0
3 3
0 3
3
0 0
2 0
0 2
	NO
	YES
	3.1415926 3 3

Problem 23. Teams

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	128 mebibytes

Mr Byteotish is the most popular P.E. teacher at No. 64 Primary School named after Byteasar, the Travelling Salesman of Byteotia. During each lesson, after a brief warm-up, he ask students to choose the team game they would like to play, and then helps them to get divided into teams.

During the line-up the students number themselves up using consecutive numbers from 1 to n. Mr Byteotish creates the teams in such a way that each of them is formed from students with consecutive numbers. Each student must belong to one of the teams.

The teacher knows his students well and knows that the student number i will be satisfied with his assignation only in case the number of the players in his team will not be less than c_i and not exceed d_i.

Mr Byteotish wonders whether it is possible to divide students into teams in such a way, that every student would be satisfied. In case it is possible, he would like to know the maximum possible number of teams that could be formed, as well as the number of divisions that are pursuing this maximum.

Input

The first line of input contains one integer n $1 \leq n \leq 1\,000\,000$), denoting the number of students. The following n lines describe the preferences of students: i-th of these lines contains two integers c_i, d_i ($1 \leq c_i \leq d_i \leq n$), indicating that the student bearing number i is satisfied, providing the number of players in his team will belong to the interval $[c_i, d_i]$.

Output

In case it is possible to divide students according to Mr Byteotish's procedure in such a way, that each of them would be satisfied, the output should contain two integers separated by a single space—the maximum number of teams and number of divisions pursuing this maximum. The second of these numbers should be produced modulo $10^9 + 7$.

If students cannot be divided in accordance with the above requirements, output should contain only the word `NIE` (Polish for *no*).

Examples

standard input	standard output

9
1 4
2 5
3 4
1 5
1 1
2 5
3 5
1 3
1 1
	5 2

2
1 1
2 2
	NIE

Problem 29. Killing The Time

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	512 mebibytes

Roma has downloaded a modern strategic game with RPG elements on his smartphone. In the game, the player is using a turn-based mechanism to command three characters and fight the hordes of enemies. Since the game is a shareware and Roma is a serious guy that does not spend money on games, this game is available to him in a very limited mode.

The number of moves is severely limited, enemies do not move, enemies and characters do not die, but the score at the end of the game is still displayed. As Roma has figured out from his attempts to play this game, the score consists of the sum of damage dealt by characters to the enemies and life healed to the characters minus the sum of damage dealt by the enemies to the characters and life healed by the characters to the enemies. Roma is about to set the absolute record for each of the levels. And you are about to help him!

The game field is a rectangular tiled grid of size $N \times M$. Some tiles are occupied by enemies, some other by impassable obstacles, yet some other by three Roma's characters, and all other tiles are free. The game process is working in the following way. Roma moves first, then moves the almighty artificial intelligence. In his turn, Roma chooses the order in which the characters will move, and then makes those moves sequentially, character by character. During his turn, Roma can move the character and then apply one of its abilities. The characters move by steps (one tile by each step) either horizontally or vertically so that the total number of moves does not exceed that character's stamina. Characters can not pass through the impassable obstacles, enemies or other characters. Roma can also decide not to move the character at all, or not to apply any of its abilities, or not to do anything by that character at all. After Roma's turn, the enemies take action, and artificial intelligence tries to minimize the player's score. Then a new turn begins or the game ends.

Each of the characters' abilities is one of the following:

- "`TELEPORT` L": the character can teleport to any free tile of the game field with Manhattan distance not more than L from the current tile.
- "`SWAP` L: the character can exchange places with any of the other characters with Manhattan distance not more than L from the current tile.
- "`SINGLE ATTACK` r R d": the character deals d base damage to any one selected tile with Manhattan distance not less than r and not greater than R from his current position.
- "`SPLASH ATTACK` r R d": the character deals d base damage to all tiles with Manhattan distance not less than r and not greater than R from his current position.
- "`SINGLE HEAL` r R d": the character heals d life to any one selected tile with Manhattan distance not less than r and not greater than R from his current position.
- "`SPLASH HEAL` r R d": the character heals d life to all tiles with Manhattan distance not less than r and not greater than R from his current position.

Each type of enemy is one of the following:

- "`SINGLE` r R d": the enemy deals d base damage to any one selected tile with Manhattan distance not less than r and not greater than R from its current position.
- "`SPLASH` r R d": the enemy deals d base damage to all tiles with the Manhattan distance not less than r and not greater than R from its current position.

Each enemy has three multipliers for the dealt damage: it deals $d \cdot M_1$ damage to the first character, $d \cdot M_2$ to the second character and $d \cdot M_3$ to the third character. Additionally, each enemy has three multipliers for the damage received: damage from the first character is multiplied by S_1, from the second by S_2, and from the third by S_3.

Since the game is played on hardcore level, the characters can deal damage to each other (base damage value) and also can heal enemies, but the enemies do not deal damage to each other.

The Manhattan distance between two positions is the sum of the absolute difference between row numbers of these two positions and of the absolute difference between column numbers of these two positions.

You are to calculate the maximum score that Roma can earn in the given game.

Input

The first line of input contains integers N, M, D and K: the dimensions of the game field, the length of the game and the number of types of enemies on the level K ($1 \le N, M, D \le 8$, $0 \le K \le 26$).

On the next N lines, the game field is given. Each of them contains exactly M characters: "`.`" denotes the field is empty; "`#`" denotes the field is not passable; "`1`", "`2`" or "`3`" denotes the positions of the characters; "`A`".."α" (where α is the K-th letter of English alphabet) denotes the type of the enemy on the field. It is guaranteed that there is exactly one of each character "`1`", "`2`" and "`3`" on the field.

On the next K lines, the descriptions of the enemy types are given. First goes the type of damage, and then follow nine numbers r, R, d, M_1, M_2, M_3, S_1, S_2, S_3 ($0 \le r \le R \le 20$, $1 \le d, M_1, M_2, M_3, S_1, S_2, S_3 \le 1000$). The first description is for type "`A`", the second one for type "`B`" and so on.

Then the descriptions of three characters are given. On the first line of each of these descriptions, two numbers are given: V, the stamina of the character and A, the number of his abilities ($0 \le V \le 5$, $0 \le A \le 100$). Then A lines describe the abilities in the format "`name of ability` L" for movement abilities ($0 \le L \le 5$) and "`name of ability` r R d" for the other abilities ($0 \le r \le R \le 20$, $1 \le d \le 1000$).

Output

Output one integer: the maximum score that Roma can earn in the given game.

Examples

standard input	standard output
4 4 1 1 .... .12. .... AA3A SINGLE 0 1 10 1 1 100 10 10 1 1 2 SPLASH HEAL 0 3 4 TELEPORT 1 1 2 SINGLE ATTACK 0 10 10 SWAP 2 1 1 SPLASH ATTACK 1 1 1000	1984
1 6 8 2 123#BA SINGLE 5 5 10 1000 100 1 1 1 1 SINGLE 2 2 5 300 1 100 1 1 1 5 1 SINGLE ATTACK 5 5 1 5 1 SPLASH HEAL 1 1 1 5 2 SPLASH ATTACK 0 1 100 SWAP 2	-5574

Problem 31. Linear Systems

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

After dealing with sociology problem last year, you now have some troubles with linear algebra.

An old professor likes to give his students huge systems of linear equations by prime modulo P to solve as a homework. He can dictate them for hours, but he found out that students still solve them quite fast and giving even bigger systems of linear equations is not an option due to limited duration of the seminar. He invented one feature though: he has his favorite number L, and if he has already dictated L coefficients of the equation, he can stop and instruct students to copy the other $(N - L)$ coefficients from the the beginning of one of the previous equations, and then dictates the right part of the equation.

OK, you'll have to deal with the task of automating the solving of a system of linear equations given in that form.

Input

On the first line of input, integers N, M, L and P are given, where N is the number of unknown variables, M is the number of equations, L is the favorite number of the professor and P is the number serving as a modulo ($1 \le N, M \le 4000$, $1 \le L < N$, $2 \le P \le 10^9$, P is prime).

The next M lines contain the equation descriptions. The first integer denotes the type of the equation: 1 if the equation is defined by professor in complete form, or 2 if he referenced the previous equation in its definition. In the former case, the line then contains $(N + 1)$ integers: the coefficients for the unknown variables and the right side of equation. In the latter case, the line then contains $(L + 2)$ integers: the L coefficients for the unknown variables, the index n_i of the equation referenced by the professor ($1 \le n_i < i$) and the right side of equation. All coefficients are non-negative and less than P.

The total amount of coefficients for the unknown variables dictated by the professor is not greater than 10 000.

Output

The first line of output must contain "`No solution`" if there is no solution for the given system of linear equations.

If a solution exists, then the first line of output must look like "`There are` P `^` k `solutions`", where k is the dimension of the solution space. The second line in this case must contain N integers x_i: the solution itself ($0 \le x_i < P$). If there are multiple solutions with these criteria, output the one with the minimal x_1. If there are still multiple solutions, output the one with the minimal x_2, and so on.

Examples

standard input	standard output
4 4 2 13	There are 13 ˆ 0 solutions
1 1 0 0 0 1	1 2 3 4
1 0 1 0 0 2	
1 0 0 1 0 3	
1 0 0 0 1 4	
4 3 1 19	There are 19 ˆ 1 solutions
1 1 0 0 0 1	1 1 1 0
2 1 1 2	
2 1 2 3	
2 2 1 11	No solution
1 1 1 1	
2 1 1 2	

Problem 37. Antichamber

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
Memory limit:	256 mebibytes

In the Antichamber video game the player has a special tool called the Brick Tool. You must model its work in a simple two-dimensional variant.

The game takes place in an infinite grid. Each cell is either white or black. Two cells are considered *adjacent* if they share a side. A maximum connected set of black cells is called a *component*. The size of a component is the number of cells in it.

The tool allows the player to repaint any cell to the opposite color. Straight afterwards checks are performed which can change the state of the grid.

If a cell has been repainted to white and the number of components has increased, then some component has split into several *subcomponents*. In this case the newly formed subcomponents are removed except for, possibly, the largest of them. The largest subcomponent is not removed only in the case when its size is strictly greater than the sum of the sizes of all the other subcomponents. Removing a component means repainting all its cells to white.

Next regardless of the color of the repainted cell, «holes» in components are removed: If any white cell is located inside any cycle of black cells, it is repainted to black. Every two neighboring cells in a cycle must be adjacent in the sense defined above.

Initially all cells are white. A sequence of operations is given. Each operation is either repainting of a cell or a query. All the operations must be executed in the given order.

There can be two types of queries. A query of type `Comp`: is it true that the given two cells are black and belong to the same component? A query of type `Size`: find the size of the component that contains the given cell. If this cell is white, the answer must be zero.

Input

The first line contains a single integer N — the number of repaintings and queries ($1 \leq N \leq 10^5$). Each of the following N lines contains a description of a single operation.

The operation formats are presented below:

- "`Draw` x y" — repaint a cell with the coordinates $(x; y)$ to the opposite color;
- "`Size` x y" — find the size of the component containing the cell with the coordinates $(x; y)$;
- "`Comp` x_1 y_1 x_2 y_2" — determine whether the cells with the coordinates $(x_1; y_1)$ and $(x_2; y_2)$ belong to the same component.

All coordinates are positive integers not exceeding 10^6.

Output

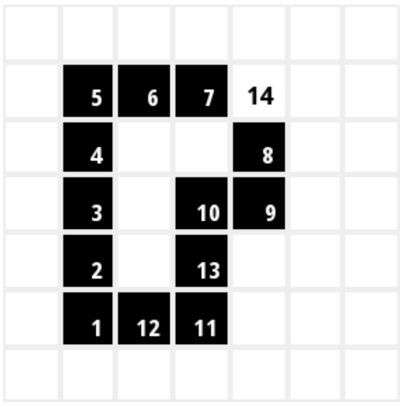
Print answers to the queries in order of their appearance in the input file, one answer per line. An answer to a query of the type `Size` must be an integer. An answer to a query of the type `Comp` must be either `YES` or `NO`.

Examples

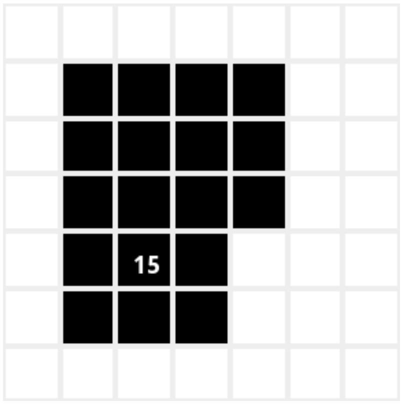
input.txt	output.txt
38	1
Draw 2 2	7
Draw 2 3	3
Draw 2 4	0
Draw 2 5	YES
Draw 2 6	NO
Draw 3 6	NO
Draw 4 6	13
Draw 5 5	18
Draw 5 4	18
Draw 4 4	YES
Draw 4 2	0
Size 4 2	9
Size 2 4	9
Size 4 4	0
Size 3 3	0
Comp 2 2 2 4	0
Comp 2 2 4 4	
Comp 3 3 4 4	
Draw 3 2	
Draw 4 3	
Size 2 2	
Draw 5 6	
Size 2 2	
Draw 3 3	
Size 3 3	
Comp 3 3 4 4	
Draw 2 4	
Draw 3 4	
Draw 4 4	
Size 3 2	
Size 3 6	
Draw 4 7	
Draw 3 5	
Size 2 5	
Draw 4 6	
Size 2 5	
Size 4 5	
Size 4 7	

Note

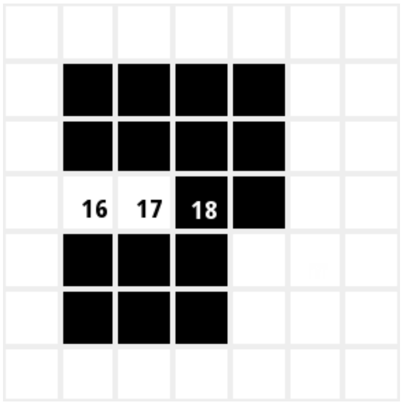
Numbers on the pictures below denote number of Brick tool usage for the corresponding number.



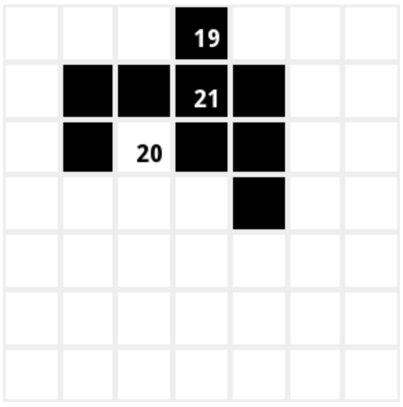
Playfield after 14'th usage of the Brick tool



15'th usage of the Brick tool does not cause changes in the playfield.



Playfield after 18'th usage of Brick tool.



After 21'th usage of Brick tool all the cells in the playfield will become white again.

Problem 41. Faster Than Light

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	6 seconds
Memory limit:	256 mebibytes

In the Faster Than Light video game each spaceship can be represented on a flat grid. All cells of the grid are unit squares. Some cells represent ship sections and they can be fired at. Other sections don't belong to the ship.

There is a beam weapon in the game which shoots in the following way. When fired the weapon draws a line segment of the fixed length L with the beam over the attacked ship. The position of the segment can be chosen arbitrarily given that one of its ends is positioned inside or on the boundary of one of the sections of the attacked ship. The other end of the segment can be anywhere, including outside the grid. The damage to the ship and its crew depends on the set of sections damaged by the beam. A ship section is considered damaged if the line segment and section cell (including its boundary) have at least one common point.

You are invited to develop a targeting program which should work in the following way. For each spaceship a positive number of points for hitting each of its sections is given. Your program must find the position of the segment which yields the maximum sum of points for all damaged sections.

Input

The first line of the input file contains two integers N and M – the numbers of rows and columns in the grid, respectively ($1 \le N, M \le 30$).

The second line contains the length of the segment — a real number L given with two or less digits after decimal point ($0.1 \le L \le 50$).

The next N lines describe the grid. Each of them contains M integers. Let the jth number in the ith of these lines equal a_{ij} ($0 \le a_{ij} \le 10^7$, $i = 1 \ldots N$, $j = 1 \ldots M$). If $a_{ij} = 0$, then the corresponding cell is empty. If $a_{ij} > 0$, then the cell contains a spaceship section, which yields a_{ij} points when hit.

It is guaranteed that at least one cell containing a spaceship section is present in the grid. Different spaceship sections may be disconnected from each other.

Output

The output file must contain a single integer – the maximum possible score for a single shot of the beam.

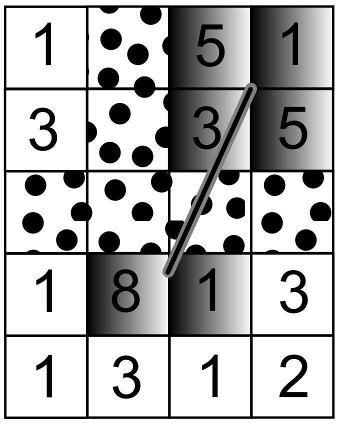
Example

input.txt	output.txt

5 4
2.5
1 0 5 1
3 0 3 5
0 0 0 0
1 8 1 3
1 3 1 2
	23

Note

The sample test allows to fire the weapon in such a way that it hits two 5-point cells, 1- and 3-point cells, as well as 8- and 1-point cells.


1
5
1
3
3
5
1
8
1
3
1
3
1
2

Problem 43. Nanobugs

Input file:	`nanobugs.in`
Output file:	`nanobugs.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

In this problem, for a collection of bugs each of which is either a spy or an agent, you have to show the true number of spy bugs while not disclosing any spy or agent.

Nanobugs are intelligent robots who obey the orders of their masters. They are usually busy spying, eavesdropping or monitoring other nanobugs. All nanobugs look exactly the same.

Bartosz and Vivek are engineers of two corporations: Eastern Cartel and Southern Trading Company. They have to examine the meeting room in the office of Eastern Cartel where representatives of their corporations will discuss a new contract. Together, they managed to uncover and catch n nanobugs. Each of the bugs is either an Eastern Cartel's security agent or a Southern Trading Company's spy.

Anders is a security expert from SpyTek. His job for today is to help Bartosz and Vivek determine how many spies and agents are among the nanobugs they caught. For each bug, Anders knows whether it is a spy or an agent, but no other person has that knowledge.

Anders sees that there are exactly a spies among the bugs. Bartosz and Vivek, on the other hand, have only learned by standard procedures that there are either a or b spies, and the numbers a and b differ by exactly one. They have no other information about the bugs.

Nanobugs are expensive intelligent machines which especially value their incognito. Sure, being an expert, Anders knows who is the master of each of the bugs. But if for some nanobug, its affiliation (which company the nanobug works for) became known to the engineers or other nanobugs, further functioning of that nanobug (and its very existence!) would be threatened.

As a security expert, Anders has an instrument which will help him in his work: a balance checker. It is a small box with two chambers and an indicator light. Using a balance checker is simple: Anders places some nanobugs in each of the chambers and presses a button. After that, for each corporation, the instrument checks that the number of nanobugs affiliated with it is the same in both chambers. If this is true for both corporations, the indicator light is green, otherwise, it is red.

Anders plans to perform a series of checks with the balance checker. For that, he will start by placing all nanobugs in a row in the order he likes. During each check, Anders will place some nanobugs in the first camera, some in the second camera, press the button, show the result to the engineers, and then return all nanobugs to their places in the row, restoring their original order. With such procedure, it can be said that, during the whole series of checks, the nanobugs are numbered by integers from 1 to n according to their places in the row.

Help Anders prove to the engineers that there are exactly a spies among the n nanobugs they caught, and do it in a way that does not disclose any nanobug's affiliation, or determine that this is not possible.

Input

The first line of input contains three integers n, a and b ($20 \le n \le 50$, $1 \le a, b \le 4$, $|a - b| = 1$). Here, n is the total number of nanobugs. Bartosz and Vivek know that there are either a or b spies among them, and Anders has to prove that the exact number of spies is a.

Output

On the first line, print one integer: the number of checks m ($0 \le m \le 1000$) or -1 if it is impossible to satisfy all conditions. In case of positive answer, print m lines, one for each check.

The description of each check contains exactly the information that Anders shows to the engineers, and consists of three parts. The first part determines which nanobugs are placed in the first chamber: it is denoted by the number of these bugs followed by their numbers in any order. Then follows a character which shows the result of the check: "=" (ASCII code 61) for positive result (green indicator light) or "^" (ASCII code 94) for negative result (red indicator light). After that, the bugs which are placed in the second chamber are listed in the same format as for the first chamber. All the above is printed in one line, and consecutive numbers or characters are separated by one or more spaces.

No bug can be mentioned twice during any one check.

Note

Note that it is not disclosed anywhere which of the bugs were actually spies and which were agents. This information is available only to Anders and, possibly, to your solution. It must not be disclosed to any other party.

Example

nanobugs.in	nanobugs.out

`22 2 1`	`4`
	`6 1 2 3 4 5 6 = 6 7 8 9 10 11 12`
	`5 13 14 15 16 21 = 5 17 18 19 22 20`
	`3 13 14 17 ^ 3 6 8 9`
	`1 1 ^ 2 2 3`

Note

In the example, the engineers know that among the 22 nanobugs they caught, there are either one or two spies. Actually, there are exactly two spies, and Anders' job is to prove that. In the presented answer, he decided to perform a series of four checks.

The first check shows that among the first six bugs, there are as much spies as there are among the next six bugs. Here and further on, the same statement is true for agents: if the number of nanobugs in both chambers is the same, the number of spies is the same if and only if the number of agents is the same.

After the second check, we may conclude that among the bugs numbered 13, 14, 15, 16 and 21, there are as much spies as there are among the bugs numbered 17, 18, 19, 22 and 20.

The third check shows that the number of spies among the bugs 13, 14 and 17 is necessarily different from the number of spies among the bugs 6, 8 and 9.

Finally, the fourth check does not provide any information since a check with different number of bugs in the chambers always gives a negative result.

Now, which bugs may have been spies?

Suppose, for example, that Anders initially placed the two spies in positions 2 and 8. Then the first check gave a positive result, as there were five agents and one spy in each of the two chambers. The second check gave a positive result since there were no spies in any of the chambers. The third check gave a negative result: there are no spies among the bugs 13, 14 and 17, but there is one spy among the bugs 6, 8 and 9. The fourth check does not depend on the positions of spies. To conclude, all checks gave the required results. So, the *possibility* that there are exactly two spies is shown.

Suppose now in place of the engineers that there was in fact one spy, and its number was 1. But then, the first check would have given a negative result. Similarly, it can be shown that if there is a single spy which has number 2, 3, ..., 22, at least one of the checks would have given a different result. So, Anders proved the *impossibility* of a situation in which there was exactly one spy among the nanobugs. Recall now that Bartosz and Vivek know there are either one or two spies. It turns out that Anders proved that the number of spies is exactly two.

What remains is to check that no agent and no spy is disclosed. For that, we can present a set of possible arrangements of spies such that each of these arrangements gives the required results during the checks, and furthermore, each nanobug is an agent in at least one of these arrangements, and each nanobug is a spy in at least one of these arrangements. Such a set is, for example, the set of arrangements where the spies have numbers $(1,8)$, $(2,9)$, $(3,8)$, $(4,9)$, $(5,8)$, $(6,7)$, $(6,10)$, $(6,11)$, $(6,12)$, $(13,17)$, $(13,18)$, $(13,19)$, $(13,20)$, $(13,22)$, $(14,18)$, $(15,17)$, $(16,17)$ and $(17,21)$.

Problem 97. Query the Black Box (supplementary for 7)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 megabytes

This problem is supplementary for the problem **7**. It cannot be solved and will return Presentation Error test 1 for any submit.

Here is the Black box description that helps you to understand the format and requirements.

First of all, we tried to make the black box consistent. That means that there is no insane things in the black box.

To make your life easier checker for this problem accepts multiple requests for Black box produced by your *program* (up to 10). So, for request you must submit a *code*, printing a request to standard output. To see result of your request, go «Submissions» tab in your ejudge client, then open report for this problem («View» in the «View report» column). In the end of report, after text "`Black box says:`" and empty line will be answer of black box to your requests, in same order as it was sent by your code.

You have limit 100 attempts for this problem, so do not waste them.

To solve the problem **7** you need to repeat the black box logic. The black box you need to implement should accept *only one* single request and produce a single response. Your black box should take a request, *not a code* that produces that request.

Input

There is no input for the problem.

Output

On the first line print a single integer n ($1 \le n \le 10$) — number of requests. Next n lines should contain single request per line. Note that request should have length no more than 1000.

Examples

standard input

```
No input for the problem
```

standard output

```
7
bool Alex_Artem_Slava(int, long, short, float, double, void, char);
bool wish_you_good_luck(long long, long double, long int);
bool and_have_fun(unsigned int, unsigned long, unsigned short);
bool and_dont_solve_this_problem(unsigned float, unsigned double);
unsigned float read_other_problems_first(double, long, int, short);
bool if_you_solve_it_we_will_give_you_some_prize(int);
int glhf(unsigned void, unsigned char);
```