Problem A. Barabashka

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

The tests for this problem are open. In this problem, you have to choose one of the five items according to a textual description of an image.

"Barabashka" (originally named "Geistesblitz") is a board game that favors fast reaction. Below are the rules of the game for this problem. Please read them carefully, as they are slightly different from the original game rules.

The game set includes five items of different colors: a white ghost named Barabashka, a green bottle, a gray mouse, a blue book and a red chair, and also special cards. Each card contains an image with exactly two of these five items. Each of the items also has one of the aforementioned five colors, but the color of each item may be wrong. The colors of these two items are different. The image can contain other objects, but their colors are guaranteed to be different from the five colors above.

A single move in the game proceeds as follows. All items are put on the table, and after that, a card is drawn from the deck and shown to the players. If the card contains an item of the right color (the color on the picture matches the color of the real item on the table), the players have to grab that item. Otherwise, the players have to grab the item which is not present on the picture, and the true color of which also does not appear on the picture. The winner is the player who grabs the right item earlier than the others. The winner gets the card as a reward. It is guaranteed that all cards contain images for which there exists a unique right answer.

In this problem, each test consists of exactly five sentences. A sentence is a textual description of one card in English. The right answer for a sentence is the description of the item to grab when the players see such card.

Each sentence is written in English and can contain English letters, spaces and characters "`'`", "`,`", "`-`" and "`.`" (ASCII codes 39, 44, 45 and 46). The names and the right colors of the items are the following: "`white Barabashka`", "`blue book`", "`red chair`", "`gray mouse`" and "`green bottle`".

Let a word be a sequence of English letters surrounded from both ends by line ends or non-letter characters. Then we can formulate the following restrictions on the sentences:

- A sentence contains exactly two positions of the form "`color name`" where the word "`color`" is one of the five colors mentioned above, and the word "`name`" is one of the five item names.
- No other word in the sentence is equal to the name or the color of some of the items on the table.

The tests in this problem are open for contestants and can be downloaded at the following addresses:

- `http://acm.math.spbu.ru/141019_files/barabashka/tests.zip` (Windows line breaks),
- `http://acm.math.spbu.ru/141019_files/barabashka/tests.tar.gz` (Linux line breaks).

The problem contains 24 tests, that is, $24 \cdot 5 = 120$ card descriptions: one description for each possible combination of items and their colors on a card.

Input

The input consists of five lines. Each line contains one sentence of length from 1 to 80 characters. The exact format of each sentence is given in the problem description. Uppercase and lowercase letters are considered different.

Output

For each sentence, print a line containing two words separated by a space: the right color and name of the item to grab when the players see the card described by that sentence. Uppercase and lowercase letters are considered different.

Example

barabashka.in	barabashka.out

`A white Barabashka is staring at a gray bottle.`
`A red mouse is running from a green Barabashka and its shadows.`
`A gray book lies on a red chair's right arm.`
`Black smoke is rising from a blue bottle lying under a white chair.`
`A white mouse is trying to crawl out of a green bottle.`
	`white Barabashka`
	`blue book`
	`red chair`
	`gray mouse`
	`green bottle`

Explanation

The first, third and fifth sentences of the example describe items of the right colors. So, their colors and names must be printed. In the second and fourth sentences, this is not the case, so we must choose such an item that neither its name nor its color appear as words in the sentence.

Problem B. Compare Continued Fractions

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

In this problem, you have to compare two rational numbers represented by their continued fractions.

A *finite continued fraction* is a sequence $[a_0; a_1, a_2, \ldots, a_n]$. The following restrictions are applied:

- n is a non-negative finite integer,
- the elements a_0, a_1, a_2, …, a_n are integers,
- $a_i > 0$ for each $i > 0$,
- $a_n > 1$ if $n > 0$.

These restrictions allow to establish a one-to-one correspondence between rational numbers and finite continued fractions: every rational number x corresponds to the unique continued fraction $[a_0; a_1, a_2, \ldots, a_n]$ such that

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}.$$

Thus, the following notation is used: $x = [a_0; a_1, a_2, \ldots, a_n]$. For example,

$$\frac{17}{25} = 0 + \frac{1}{\frac{25}{17}} = 0 + \frac{1}{1 + \frac{8}{17}} = 0 + \frac{1}{1 + \frac{1}{\frac{17}{8}}} = \mathbf{0} + \frac{1}{\mathbf{1} + \frac{1}{\mathbf{2} + \frac{1}{\mathbf{8}}}},$$

so we write $\frac{17}{25} = [0; 1, 2, 8]$.

Given the continued fractions for two rational numbers x and y, find whether $x < y$, $x = y$, or $x > y$.

Input

The input consists of two lines. The first line contains the continued fraction for the rational number x. The second line contains the continued fraction for the rational number y.

Each continued fraction is given as a sequence of integers separated by single spaces. First goes an integer n, the length of the continued fraction ($0 \le n \le 100\,000$). It is followed by $(n + 1)$ integers which are the elements of the continued fraction: a_0, a_1, a_2, …, a_n ($|a_i| \le 10^9$). It is guaranteed that $a_i > 0$ for each $i > 0$ and $a_n > 1$ if $n > 0$.

Output

On the first line of output, print a single character: "<" if $x < y$, "=" if $x = y$, or ">" if $x > y$.

Examples

ccf.in	ccf.out	Notes

`1 0 3`
`2 0 1 2`
	`<`	$x = 0 + \frac{1}{3} = \frac{1}{3},\ y = 0 + \frac{1}{1+\frac{1}{2}} = \frac{2}{3}$

`0 1` `0 1`	`=`	$x = 1,\ y = 1$

`1 -1 2`
`0 -1`
	`>`	$x = -1 + \frac{1}{2} = -\frac{1}{2},\ y = -1$

Problem C. Graph Coloring

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

You are given a bidirectional graph where the degree of each vertex is at most 5. Paint its vertices in 3 colors in such a way that each vertex v has no more than one neighbor of the same color as v.

Input

On the first line, there are two integers n and m: the number of vertices and the number of edges ($1 \le n \le 100\,000$).

Each of next m lines contains two integers a and b: the numbers of vertices connected by an edge ($1 \le a, b \le n$).

It is guaranteed that there are no loops or multiple edges in the graph, and the degree of each vertex is at most 5.

Output

It there is no valid coloring, print one number "`-1`" (without quotes). Otherwise, print n integers c_1, c_2, …, c_n: the colors of all vertices ($1 \le c_i \le 3$). If there is more than one solution, print any one of them.

Examples

coloring.in	coloring.out

3 3
1 2
2 3
1 3
	2 1 1

6 15
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
	2 2 3 3 1 1

3 1 1 2	1 1 1

Problem D. Convolution

Input file:	`convolution.in`
Output file:	`convolution.out`
Time limit:	2 second
Memory limit:	256 mebibytes

Consider all subsets of set $U = \{0, 1, 2, \ldots, n-1\}$. Every subset $A = \{a_1, a_2, \ldots, a_k\}$ corresponds to a unique integer $p(A) = \sum_{i=1}^{k} 2^{a_i}$. Let function F of an n-element set be defined by an array of integers f of length 2^n: the value $F(A)$ is equal to $f[p(A)]$.

You are given two functions F and G. Your task is to find such function H that

$$H(A) = \sum_{B \cup C = A} F(B)G(C).$$

Input

The first line contains two integers n and t ($1 \le n \le 16$, $1 \le t \le 100$). Here, n is the size of the set U, and t is number of test cases. The second line contains two integers a and b, each from 1 to 10^9. These numbers are used in the following pseudo-random generator:

```
1. unsigned int cur = 0; // unsigned 32-bit integer
2. unsigned int nextRand16() {
3.     cur = cur * a + b; // calculated modulo $2^{32}$
4.     return cur / $2^{16}$; // integer from 0 to $2^{16} - 1$
5. }
```

The test cases are generated successively. In each of them, first, you must generate the elements of array f (values of F) in the order of increasing array index, and after that, you must generate the elements of g (values of G) in the same order. Each element is generated by calling the function `nextRand16()`.

Output

For each test case, print one integer on a separate line: $\left(\sum_A H(A) \cdot (p(A)+1)\right) \bmod 2^{32}$.

Examples

convolution.in	convolution.out
3 2 30 239017	2723387430 3167905008
16 2 239 17	551267264 1632349120

Explanations

The arrays in the first example are the following:

f_1: 3, 113, 3395, 36331, 41370, 61471, 9130, 11774

g_1: 25547, 45526, 55066, 13590, 14501, 41817, 9356, 18543

h_1: 76641, 8167827, 273846333, 5284992017, 1656829263, 11450721456, 3699971823, 14260048942

f_2: 32024, 43238, 51978, 52034, 53714, 38578, 43250, 52338

g_2: 62834, 50034, 59250, 8050, 44914, 36722, 53106, 20338

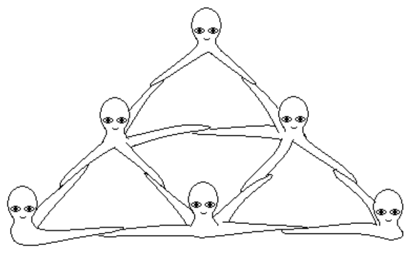
h_2: 2012196016, 6482475400, 8243104152, 15561662464, 7225902008, 16869349792, 22350138288, 44342816072

Problem E. Octopus's Garden

Input file:	`garden.in`
Output file:	`garden.out`
Time limit:	**5 seconds**
Memory limit:	256 mebibytes

Octopuses are located on the surface of the planar seabed in the most beatiful manner. Octopuses like their location if it specifies a partition of the seabed plane into triangles such that the octopuses' heads are in triangle vertices, and triangle edges are formed by their outstretched tentacles. They call such partition a triangulation. Let us formalize this notion with the help of graph theory.

A *planar graph* is a graph that can be drawn on a plane in such a way that no edges cross each other.



A *face* of a graph drawn without crossings is a simple cycle which surrounds such a region that there are no edges or vertices inside the region.

An *outer face* of a graph drawn without crossings is the region which has infinite area.

A *triangulation* of a set of points on the plane is a connected planar graph drawn in such a way that its vertices are in these points, and every face except the outer face is a non-degenerate triangle formed by exactly three edges of the graph.

Two triangles are *connected by side* if they have a common side.

A triangle and the outer face are *connected by side* if one of the triangle's sides touches the outer face.

Let us denote the set of faces of the triangulation by S.

A sequence of elements of S is called a *path* if every two consecutive members of it are connected by side.

A subset of S is said to be *connected by side* if for any two elements of this subset, there exists a path such that all its elements belong to this subset.

We are given a triangulation S.

Consider a subset of S that does not contain the outer face. We will say such a subset is *simply connected* if it satisfies the following conditions:

- the subset is connected by side;
- the complement of the subset in S also is connected by side.

Octopuses like simply connected sets because they plan to take contour integrals around their border.

One of the triangles in the triangulation is fixed as the starting triangle.

Help the octopuses to find a sequence of all triangles in the triangulation which starts with the starting triangle, and for any $1 \le i \le K$ where K is the number of triangles, the set of triangles T_1, T_2, …, T_i is simply connected.

Input

The first line of input contains two integers N ($3 \le N \le 100\,000$) and M: the number of octopuses and the number of edges formed by their outstretched tentacles. The second line contains three integers A, B and C: the numbers of octopuses that occupy the vertices of the starting triangle.

Each of the following N lines contains two integers X and Y: coordinates of points on the seabed plane where octopuses are located. ($-10\,000 \le X, Y \le 10\,000$). All these points are distinct.

Each of the following M lines contains two integers U and V: the numbers of two octopuses forming an edge by their outstretched tentacles.

Octopuses are numbered from one in the order they are described. An octopus may have any number of tentacles, not only eight as in nature.

It is known that the subset of S consisting of all its elements except the outer face is connected by side. The union of all elements of this subset forms a convex polygon.

Output

On the first line of output, print an integer K: the number of triangles formed by octopuses. On each of following $(K-1)$ lines, print tree integers $P_i^{(1)}$, $P_i^{(2)}$, $P_i^{(3)}$: the numbers of three octopuses which form triangle T_i $(2 \le i \le K)$. For every i such that $1 \le i \le K$, the set of triangles T_1, T_2, …, T_i must be simply connected. If there is more than one possible solution, output any one of them.

Examples

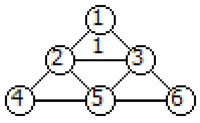
garden.in	garden.out

6 9
1 2 3
2 0
1 1
3 1
0 2
2 2
4 2
1 2
1 3
2 3
2 4
2 5
3 5
3 6
4 5
5 6
	4
	2 3 5
	2 4 5
	3 5 6

3 3
1 2 3
1 1
1 2
2 1
1 2
2 3
3 1
	1

Explanations

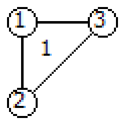
In the first example:



the triangle sequence such that the first triangle is formed by octopuses 1, 2, 3 and the second one by 2, 4, 5 is incorrect because one of conditions is not satisfied: this triangles are not connected by side;

the triangle sequence such that the first triangle is formed by octopuses 1, 2, 3 and the second one by 2, 3, 5 is correct;

the triangle sequence such that the first triangle is formed by octopuses 1, 2, 3, the second one by 3, 5, 6 and the third one by 2, 4, 5 is incorrect because both conditions are not satisfied: the set formed by this sequence is not connected by side, and the complement set is also not connected by side.



The second example contains only one triangle, so output only one integer K.

Problem F. Pentagon

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

You are given a pentagon. Your task is to find the number of its triangulations.

A *triangulation of polygon P* is a collection of triangles such that the internal areas of these triangles have no common points, and their union is equal to P. Each vertex of each triangle must coincide with some of the polygon's vertices. Additionally, each side of each traingle must either fully coincide with some of the polygon's sides or have exactly two common points (endpoints) with the border of the polygon.

Input

The first line of input contains one integer n, the number of test cases ($1 \le n \le 10\,000$). Then, the test cases follow. There is a single empty line before each test case.

Each test case describes one pentagon and consists of five lines. Each of these lines contains two integers x and y separated by a single space: coordinates of the next vertex of the pentagon in either clockwise or counter-clockwise direction ($-100 \le x, y \le 100$). It is guaranteed that every pentagon has no self-intersections and no self-touchings. Additionally, no two adjacent sides of the polygon lie on the same line.

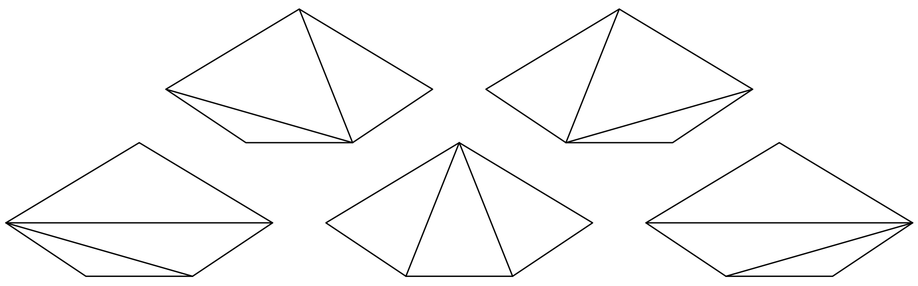
Output

Print n lines, one line for each test case. Each of these lines must contain one integer: the number of different triangulations of the corresponding pentagon.

Example

pentagon.in	pentagon.out
`2`	`5`
	`1`
`-5 0`	
`0 3`	
`5 0`	
`2 -2`	
`-2 -2`	
`-5 -5`	
`-5 5`	
`5 5`	
`5 -5`	
`0 0`	

Explanation



This image shows all possible triangulations of the first pentagon in the example.

Problem G. Squares Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem. In this problem, you have to win against an opponent who makes random moves.

Felix and Sophia play a game on a rectangular board consisting of $m \times n$ square cells. They move in turns. Each cell of the board is either free or painted. Initially all cells are free. A move consists in selecting any four free cells which form a 2×2 square and painting these cells. The player who can not make a move loses the game.

On each of his turns, Felix chooses one of the possible 2×2 squares uniformly at random and paints it. Your task is to play as Sophia. Your solution will be considered correct if it loses no more than 10 games out of 300.

In this problem, each test specifies the size of the board. In each test, you have to play exactly 300 games. In each game, Felix moves first and Sophia moves second.

Input

The first line contains two integers m and n separated by a space: the size of the board ($16 \le m, n \le 25$). Note the lower limit: the board can not be too small. The rows of the board are numbered from 1 to m, and the columns from 1 to n.

The following input is divided into blocks, and each block corresponds to one game. Each block starts with a line "`start` k" where k is the number of the game starting from 1. Each of the following lines has the form "`move` r c" ($1 \le r < m$, $1 \le c < n$). Such a line means that on his turn, Felix painted four cells with coordinates (r, c), $(r, c+1)$, $(r+1, c)$ and $(r+1, c+1)$. If a game finishes because Sophia wins, the line given instead of the next Felix's turn is "`won`", and after that, the block terminates. If a game finishes because Felix wins, the next line after the Felix's turn is "`lost`", and after that, the block also terminates. The blocks follow one after another without any additional separators. After the last block, there is a single line "`end`".

It is guaranteed that Felix makes all moves according to the rules of the game, and each move is chosen pseudo-randomly and uniformly. It is worth noting that, before the first game, the pseudo-random number generator is always initialized by the same value. This means that, if your solution is playing as Sophia deterministically, on each test, all moves and the results of all games will remain the same if the solution is run repetitively.

Output

Each time when it is Sophia's turn in a game and there is at least one possible move, print a line of the form "`move` r c" ($1 \le r < m$, $1 \le c < n$). Such a line will mean that the move you selected for Sophia is to paint four cells with coordinates (r, c), $(r, c+1)$, $(r+1, c)$ and $(r+1, c+1)$.

To prevent output buffering, after printing each line, consider issuing the command which flushes the buffer. For example, this command may be `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

If your solution makes an invalid move or loses more than 10 games, it gets the «`Wrong Answer`» outcome on the respective test.

Example

standard input	standard output
2 5	*<reading input>*
start 1	*<reading input>*
move 1 4	*<reading input>*
<waiting for output>	move 1 1
won	*<reading input>*
start 2	*<reading input>*
move 1 2	*<reading input>*
<waiting for output>	move 1 4
won	*<reading input>*
end	*<terminating>*

Explanation

When your solution is checked, the first test is the example from the statement. Note that the constraint $m, n \geq 16$ is not satisfied for this test, but still, it is satisfied for all tests starting from the second one. Furthermore, in the example test, there are only two games instead of 300 games as in all other tests. The choice of moves by Felix does not depend on Sophia's moves and is determined in advance. Finally, on a 2×5 board, the second player wins regardless of how the game goes.

Your program passes the example test if it wins both games and successfully terminates after that. Recall that your program passes a regular test if it loses no more than 10 games and successfully terminates after that.

Problem H. Removing Vertices

Input file:	`removing.in`
Output file:	`removing.out`
Time limit:	2 seconds
Memory limit:	256 mebybytes

You are given an bidirectional graph with $(n+1)$ vertices numbered from 0 to n. There are no loops and no multiple edges in this graph. Additionally, all cycles pass through vertex 0. Your task is to remove the minimum possible number of vertices so that there are no cycles in the graph. You can not remove the vertex number 0.

Input

On the first line, there are two integers n and m: the number of vertices excluding vertex 0 and the number of edges ($1 \le n \le 10^5$, $1 \le m \le 10^6$).

Each of next m lines contains two integers a and b: the numbers of vertices connected by an edge ($0 \le a, b \le n$).

It is guaranteered that all cycles pass through vertex 0.

Output

On the first line, print one integer k: the minimum possible number of vertices to remove. On the second line, print k integers: the numbers of the vertices to remove in any order. If there are several optimal answers, print any one of them.

Examples

removing.in	removing.out

4 6
0 1
1 2
2 0
0 3
3 4
4 0
	2
	1 3

3 5
0 1
1 2
2 0
2 3
3 0
	1
	2

Problem I. Two Paths

Input file:	`two-paths.in`
Output file:	`two-paths.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are given a directed acyclic graph with two fixed pairs of vertices: (A, B) and (C, D). Consider all pairs of edge-disjoint simple paths — one from A to B, another from C to D. Your task is to find pair with minimal total length.

Input

The first line of input contains two integers n and m: the number of vertices and edges in graph, respectively ($1 \le n \le 5000$, $0 \le m \le 20\,000$).

Each of the next m lines contains two integers u and v which mean that there is an edge from u to v. ($1 \le u, v \le n$).

It is guaranteed that the given graph contains no loops, no multiple edges and no cycles.

The last line of the input contains the numbers of the fixed vertices in the order A, B, C, D ($1 \le A, B, C, D \le n$).

Output

If no pair of paths exists, print a single line with a single word "`NO`" on it. Otherwise, print "`YES`" on the first line. After that, print the descriptions of the paths: first the path from A to B, then the path from C to D.

The description of each path must consist of two lines. On the first of these lines, print the number of vertices in the path. The following line must contain the numbers of vertices in the path in the order in which they are visited. The paths must be simple (each vertex of the graph must be visited no more than once by each path). If there are several possible solutions, print any one of them.

Examples

two-paths.in	two-paths.out

4 3
1 2
2 3
3 4
1 2 3 4
	YES
	2
	1 2
	2
	3 4

4 3
1 2
2 3
3 4
1 3 2 4
	NO

Problem J. Two Rectangles

Input file:	`two-rectangles.in`
Output file:	`two-rectangles.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

In this problem, you have to find two rectangles with the given total area which have the minimum possible total perimeter.

Recall that the area of a rectangle having sides of length m and n is $m \cdot n$, and its perimeter is $2 \cdot (m + n)$.

Given an integer $s \geq 2$, consider two rectangles with positive integer lengths of sides such that the sum of their areas is s. What is the minimum possible sum of their perimeters?

Formally, choose four positive side lengths a, b, c and d so that the total area $a \cdot b + c \cdot d$ equals s and the total perimeter $2 \cdot (a + b) + 2 \cdot (c + d)$ is minimum possible.

Input

The first line of input contains one integer s ($2 \leq s \leq 10^{18}$).

Output

On the first line, print one number: the minimum possible total perimeter. On the second line, print a and b, the side lengths of the first rectangle, separated by a space. On the third line, print c and d, the side lengths of the second rectangle, also separated by a space. If there is more than one possible answer, print any one of them.

Examples

two-rectangles.in	two-rectangles.out

5	12
	1 1
	2 2

8	16
	3 2
	1 2

Explanations

In the first example, the only optimal answer is to choose squares of sizes 1×1 and 2×2. They can be printed in any order.

In the second example, there is another optimal answer: instead of rectangles 1×2 and 2×3, we can choose two squares of size 2×2 each.