

Problem A. Abbreviation

Input file: abbreviation.in
Output file: abbreviation.out

An abbreviation (from Latin *brevis*, meaning short) is a shortened form of a word or phrase. In this problem you must write an automated tool that replaces a sequence of capitalized words with the corresponding abbreviation that consists of the the first capital letters only, followed by a full definition in parenthesis. See sample input and output.

Let us make some formal definitions. A *word* in a text is a maximally long sequence of lower and upper case English letters. A *capitalized word* is a word that consists of an upper case letter followed by one or more lower case letters. For example, “Ab”, “Abc”, “Abcd”, and “Abcde” are all capitalized words, while “ab”, “A”, “AB”, “ABc” and “AbC” are not.

An *abbreviatable sequence of words* is a sequence of two or more capitalized words that are separated by **exactly** one space, no line breaks are allowed inside it.

An *abbreviation* of a an abbreviatable sequence of words is a sequence of the first (upper case) letters of each word, followed by a single space, an opening parenthesis, the original abbreviatable sequence, and a closing parenthesis.

Input

The input file consists of at least one line and up to 1 000 lines of text with up to 120 characters on each line. Each line consists of spaces, upper and lower case letters, commas or dots. There are no leading or trailing spaces on lines and there are no empty lines.

Output

Write to the output file the the original text with every abbreviatable sequence of words replaced with with the corresponding abbreviation.

Example

abbreviation.in
This is ACM North Eastern European Regional Contest, sponsored by International Business Machines. The. Best. Contest. Ever. A Great Opportunity for all contestants.
abbreviation.out
This is ACM NEERC (North Eastern European Regional Contest), sponsored by IBM (International Business Machines). The. Best. Contest. Ever. A GO (Great Opportunity) for all contestants.
abbreviation.in
ab Ab A Abc AB Abcd ABc Abcde AbC
abbreviation.out
ab Ab A Abc AB Abcd ABc Abcde AbC
abbreviation.in
Oh No Extra Spaces.And,Punctuation Ruin Everything
abbreviation.out
Oh No ES (Extra Spaces).And,PRE (Punctuation Ruin Everything)

Problem B. Binary Code

Input file: `binary.in`
Output file: `binary.out`

Ben has recently learned about binary prefix codes. A binary code is a set of n distinct nonempty code words s_i , each consisting of 0 and 1. A code is called *prefix code* if for every $i \neq j$ neither s_i is a prefix of s_j nor s_j is a prefix of s_i . A word x is called a *prefix* of a word w if there exists a word y , such that $xy = w$.

Ben found a paper with n lines of binary code in it. But this paper is pretty old and there are some unreadable characters. Fortunately, each word contains no more than one unreadable character.

Ben wants to know, whether these n lines could represent a binary prefix code. In other words, can he replace every unreadable character with 0 or 1, so that the code becomes a prefix code.

Input

The first line contains integer n — the number of code words ($1 \leq n \leq 5 \cdot 10^5$)

Next n lines contain nonempty code word records, one per line. Each code word record consists of “0”, “1” and “?” characters. Every code word contains at most one “?” character that represents unreadable character.

The total length of words does not exceed $5 \cdot 10^5$.

Output

Output “NO” on the first line if the code is not a prefix code.

Otherwise, output “YES” on the first line. Next n lines shall contain code words in the same order as the corresponding word records in the input.

If there are several prefix codes, that could be written on paper, output any one.

Example

binary.in	binary.out
4	YES
00?	000
0?00	0100
?1	11
1?0	100
3	NO
0100	
01?0	
01?0	

Problem C. Cactus Construction

Input file: `cactus.in`
Output file: `cactus.out`

Let us consider the following way of constructing graphs. Pick the number of colors \hat{c} . Let n be the number of a vertices in a graph. To build a graph, we start with a workspace with several graphs on it. Each vertex of each graph has a color. Colors are denoted by integers from 1 to \hat{c} . Initially, we have n graphs with one vertex in each graph, all colored to color 1, and no edges. The only vertex of i -th graph has number i .

The following operations are permitted:

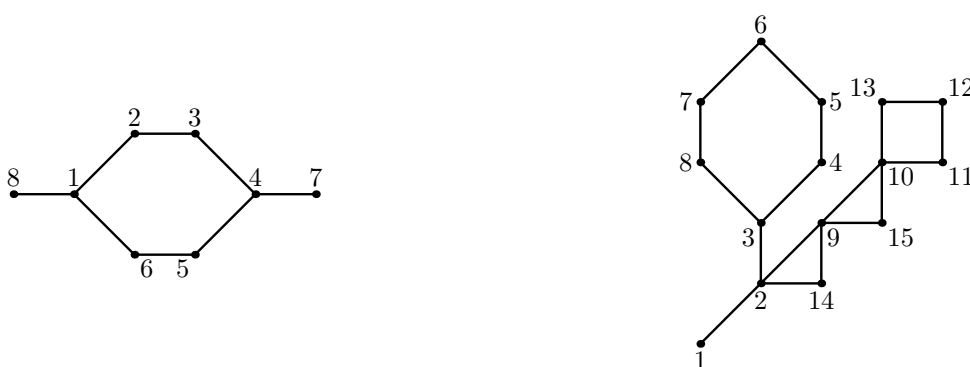
- *join a b*: join graphs containing vertices a and b into one graph. No edges are added. Vertices a and b must be in different graphs.
- *recolor a c_1 c_2* : in graph containing vertex a recolor all vertices of color c_1 to color c_2 .
- *connect a c_1 c_2* : in graph containing vertex a create edges between all pairs of vertices such that one vertex has color c_1 and the other has color c_2 . If $c_1 = c_2$ loops are not created. If such an edge already exists, then the second parallel edge *is* created. Multiedges are not allowed in this problem, so this case must not occur.

At the end we should have a single graph and colors of its vertices do not matter.

The minimal number of colors \hat{c} that can be used to construct a graph, is called a *clique width* of a graph. Clique width is one of the characteristics of graph complexity. Many NP-hard problems can be solved in polynomial time on graphs with bounded clique width, using dynamic programming on this process of building a graph. For general graph, calculating the exact value of a clique width is known to be NP-hard. However, for some graph classes good results are known.

Cactus is a connected undirected graph in which every edge lies on at most one simple cycle. Intuitively cactus is a generalization of a tree where some cycles are allowed. Multiedges (multiple edges between a pair of vertices) and loops (edges that connect a vertex to itself) are not allowed in a cactus. It is known that a clique width of a cactus does not exceed 4.

You are given a cactus. Find out how to build it in the described way using at most $\hat{c} = 4$ colors.



Input

The first line of the input file contains two integers n and m ($1 \leq n \leq 50\,000$, $0 \leq m \leq 50\,000$). Here n is the number of vertices in the graph. Vertices are numbered from 1 to n . Edges of the graph are represented by a set of edge-distinct paths, where m is the number of such paths.

Each of the following m lines contains a path in the graph. A path starts with an integer k_i ($2 \leq k_i \leq 1000$) followed by k_i integers from 1 to n . These k_i integers represent vertices of a path. Adjacent vertices in a path are distinct. Path can go to the same vertex multiple times, but every edge is traversed exactly once in the whole input file.

The graph in the input file is a cactus.

Output

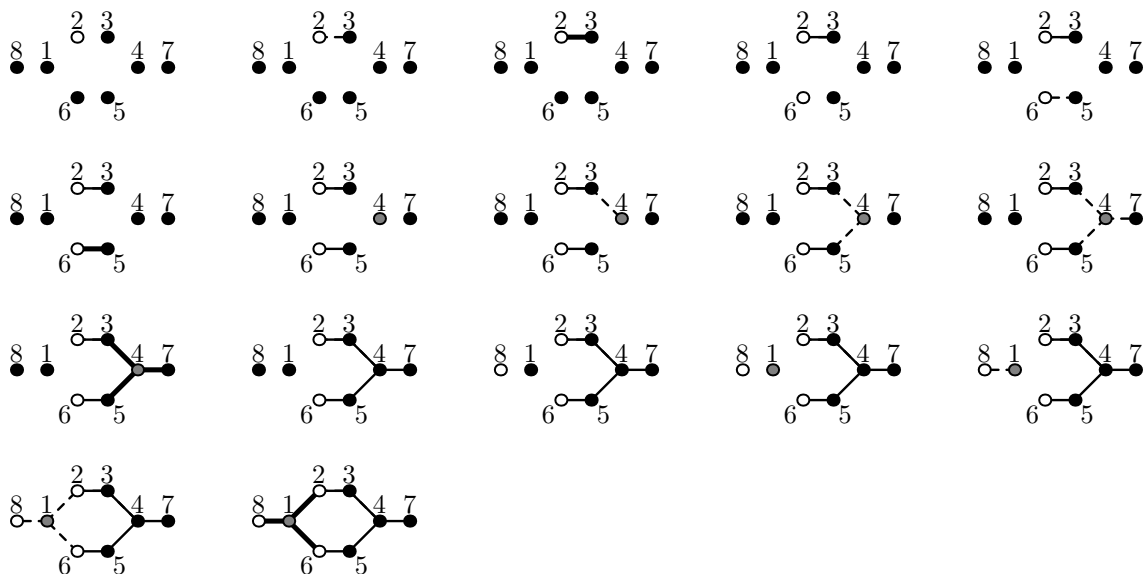
On the first line print one integer q — the number of operations you need. This number should not be greater than 10^6 . On the next q lines print operations. Each operation is denoted by its first letter (“j” for *join*, “r” for *recolor* and “c” for *connect*) and its arguments in order they are described in the problem statement.

At the end, after applying all these operations, one should have one graph, which is equal to the cactus in the input. This means that there should be exactly one edge between each pair of vertices connected in the input graph, and no edges between vertices not connected in the input graph.

Example

cactus.in	cactus.out
8 2	17
5 1 2 3 4 7	r 2 1 2
5 4 5 6 1 8	j 2 3
	c 2 1 2
	r 6 1 2
	j 5 6
	c 6 1 2
	r 4 1 3
	j 4 3
	j 4 6
	j 4 7
	c 4 3 1
	r 4 3 1
	r 8 1 2
	r 1 1 3
	j 1 8
	j 1 4
	c 1 3 2

The following picture visualizes the sequence 17 operations from the first sample output. If edge is not created yet, but it's vertices are already in one graph, this edge is drawn as dashed.



cactus.in	cactus.out
15 3	39
9 1 2 3 4 5 6 7 8 3	r 7 1 2
7 2 9 10 11 12 13 10	r 5 1 2
5 2 14 9 15 10	j 7 8
	c 7 1 2
	j 5 4
	c 5 1 2
	r 6 1 3
	j 6 7
	j 6 5
	c 6 3 2
	r 3 1 4
	j 6 3
	c 6 4 1
	r 11 1 2
	r 13 1 2
	j 12 11
	j 12 13
	c 11 1 2
	r 10 1 3
	j 12 10
	c 10 2 3
	r 10 1 2
	r 10 4 2
	r 15 1 3
	j 15 10
	c 15 3 3
	j 9 10
	c 9 1 3
	r 9 3 2
	r 9 1 4
	r 14 1 4
	j 9 14
	c 9 4 4
	r 1 1 4
	r 3 1 2
	j 2 1
	j 2 14
	j 2 3
	c 2 1 4

Problem D. Delight for a Cat

Input file: `delight.in`
Output file: `delight.out`

A cat is going on an adventure.

Each hour, the cat can be either *sleeping* or *eating*. The cat cannot be doing both actions at the same hour, and the cat is doing exactly one of these actions for the whole hour.

For each of the next n hours, the amount of *delight* the cat is getting if it will be sleeping or eating during that hour is known. These amounts can be different for each hour.

An integer time period k is also known. Among every k consecutive hours, there should be at least $minS$ hours when the cat is sleeping, and at least $minE$ hours when the cat is eating. So, there are exactly $n - k + 1$ segments of k hours for which this condition must be satisfied.

Find the maximum amount of delight the cat can get during the next n hours.

Input

The first line of the input contains four integers n , k , $minS$ and $minE$ ($1 \leq k \leq n \leq 1000$; $0 \leq minS, minE \leq k$; $minS + minE \leq k$) — the number of upcoming hours, the length of the period (in hours), and the minimum number of hours the cat should be sleeping and eating out of every k consecutive hours, respectively.

The second line contains n integers s_1, s_2, \dots, s_n ($0 \leq s_i \leq 10^9$) — the amount of delight the cat gets when it is sleeping during the first, second, ..., n -th hour.

The third line contains n integers e_1, e_2, \dots, e_n ($0 \leq e_i \leq 10^9$) — the amount of delight the cat gets when it is eating during the first, second, ..., n -th hour.

Output

In the first line, output a single integer — the maximum amount of delight the cat can get during the next n hours.

In the second line, output a string of length n consisting of characters “S” and “E”. The i -th character of this string should correspond to whether the cat should sleep (“S”) or eat (“E”) in the i -th hour to get the maximum amount of delight out of these n hours.

Example

<code>delight.in</code>	<code>delight.out</code>
10 4 1 2 1 2 3 4 5 6 7 8 9 10 10 9 8 7 6 5 4 3 2 1	69 EEESESEESS

Problem E. Expect to Wait

Input file: `expect.in`
Output file: `expect.out`

Mayor Adam East wants to improve the public transport network of Harshel city by introducing the network of stations with unicycles. Any person who owns a special card can come to a station and request a unicycle to ride or drop one.

The procedure of requesting a unicycle is simple. The person enters a *queue*. If there is a unicycle available, then the first person from the queue takes it immediately. Otherwise, people in the queue wait until someone drops a unicycle at the station.

Let the *wait time* be the time that person spends between the request (entrance to the queue) and obtaining a unicycle. If the person does not receive a unicycle at all, then the wait time is equal to infinity. The total wait time is the sum of wait times for each person.

Adam already knows the schedule of all the people for every day. He knows at what times people request and drop unicycles at the Central Station that can hold any number of unicycles at the same time. The only thing he does not know is how many unicycles should be placed there at the start of each day. He asks you several questions to calculate the total wait time given the starting number of unicycles.

Input

The first line contains n and q ($1 \leq n, q \leq 10^5$), where n is the total number of unicycle requests and unicycle drops at the Central Station, and q the number of questions Adam asks you. Next n lines describe one operation at the Central Station. Each line contains one description of operation:

- “+ t k ” when k unicycles are dropped at the time t ;
- “- t k ” when k persons request unicycles at the time t .

For each of the described operations $1 \leq t \leq 10^9$ and $1 \leq k \leq 10^4$. The last line of the input contains q different integers b_i ($0 \leq b_i \leq 10^9$) — the number of unicycles at the start of the day.

The operations are given in the strongly increasing order of time.

Output

The output shall contain q lines. i -th line shall contain the total wait time for the case of b_i unicycles at the start of the day. If the total wait time is infinite, then the corresponding line shall contain the word “INFINITY”.

Example

<code>expect.in</code>	<code>expect.out</code>
5 4	INFINITY
- 1 1	0
- 2 2	8
+ 4 1	3
- 6 1	
+ 7 2	
0 3 1 2	

Problem F. Foreign Postcards

Input file: `foreign.in`
Output file: `foreign.out`

Fedor is an avid traveler. As the result of his hobby, he has gathered a big collection of postcards from all over the world. Each postcard has a unique picture on the front side and some fields for address information and text on the back side.

During one of the parties at Fedor's house, he decided to show all his postcards to the guests. To achieve that, he wants to lay them all out on the table. Initially, all his postcards are arranged in a single stack that Fedor is holding in his hands. Unfortunately, some of the postcards in that stack can be turned incorrectly — upside down. Ideally, Fedor would like all postcards on the table to lie with the picture on top, but looking at every postcard and turning it over individually can be very time-consuming. Instead, he came up with the following process:

1. Let n be the number of postcards remaining in the stack in his hands. Fedor chooses a random number k uniformly between 1 and n and takes top k postcards from the stack.
2. He looks at the topmost postcard among those k postcards. If it's oriented the wrong way, he turns **the whole stack of k postcards** upside down together.
3. He then puts these k postcards on the table without any further rotations.
4. If there are still some postcards remaining in the stack in his hands, Fedor goes back to step 1.

Of course, after all the postcards are on the table, there might still be some that lie back side up. What is the expected number of such postcards?

Input

Input contains a single line consisting of "C" and "W" characters — i -th character corresponds to i -th postcard in the stack, counting from the top of the stack. C means that i -th postcard is oriented correctly in the initial stack, and W means that i -th postcard is not oriented correctly. The number of characters is between 1 and 1 000 000 inclusive.

Output

Output one real number — the expected number of incorrectly placed postcards on the table. The absolute or relative error should not exceed 10^{-9} .

Example

<code>foreign.in</code>	<code>foreign.out</code>
CWCC	1.0
WWCWCCW	2.333333333333

Problem G. Game on graph

Input file: `game.in`
Output file: `game.out`

Gennady and Georgiy are playing interesting game on a directed graph. The graph has n vertices and m arcs. Gennady and Georgiy have a token placed in one of the graph vertices. They make moves in turn. On his move a player must move the token along one of the arcs that starts in the vertex the token is currently in. When there is no such arc, then this player loses the game.

For each initial position of the token and the player who is moving first, your task is to determine what kind of result the game is going to have. Seems to be easy? Not so much.

On one side, Gennady is having a lot of fun playing this game, so he wants to play as long as possible. He even prefers a strategy that leads to infinite game to a strategy that makes him a winner. But if he cannot make the game infinite, then he obviously prefers winning to losing.

On the other side, Georgiy has a lot of other work, so he does not want to play the game infinitely. Georgiy wants to win the game, but if he cannot win, then he prefers losing game to making it infinite.

Both players are playing optimally. Both players know preferences of the other player.

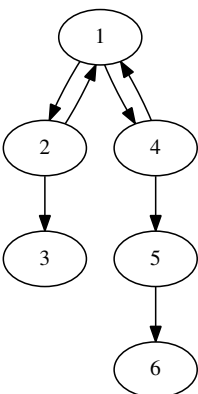
Input

In the first line there are two integers — the number of vertices n ($1 \leq n \leq 100\,000$) and the number of arcs m ($1 \leq m \leq 200\,000$). In the next m lines there are two integers a and b on each line, denoting an arc from vertex a to vertex b . Vertices are numbered from 1 to n . Each arc is listed at most once.

Output

In the first line print n characters. i -th character should denote the result of the game if Gennady starts in vertex i . In the second line print n characters. i -th character should denote the result of the game if Georgiy starts in vertex i . The result of the game is denoted by “W” if the starting player wins the game, “L” if the starting player loses the game, and “D” if the game runs infinitely.

Example

game.in	game.out	Illustration
6 7 1 2 2 1 2 3 1 4 4 1 4 5 5 6	WDLDWL DWLLWL	

Note

In vertices 3 and 6 the game is already lost. In vertex 5, the only move is to vertex 6, and this player wins. If Georgiy starts in vertex 1, or Gennady in vertices 2 or 4, Gennady can always go to vertex 1, and make the game infinite. If Georgiy starts in vertex 4, he can either go to vertex 1 (which leads to a draw) or to vertex 5, which leads to losing. Georgiy prefers the latter. Similarly, from vertex 2, he prefers to go to 3 and win. From vertex 1, Gennady can go to vertex 2 and lose, or go to vertex 4 and win. He prefers the latter.

Problem H. Hard Refactoring

Input file: `hard.in`
Output file: `hard.out`

Helen had come upon a piece of code that uses a lot of “magical constants”. She found a logical expression that checks if an integer number x belongs to a certain set of ranges, like the one shown below:

```
x >= 5 && x <= 10 ||  
x >= 7 && x <= 20 ||  
x <= 2 ||  
x >= 21 && x <= 25 ||  
x >= 8 && x <= 10 ||  
x >= 100
```

Helen does not like “magical constants”, so she decided to *refactor* this expression and all similar ones in such a way, that the refactored expression still has the same boolean result for all integer numbers x , but it uses as few integer literals as possible.

Integers in this problem, including integer x , come from the range of all signed 16 bit integer numbers starting from -2^{15} ($-32\,768$) to $2^{15} - 1$ ($32\,767$) inclusive.

Input

The input file contains at most 1000 lines. Each line contains either one comparison or two comparisons separated by *boolean and* operator “&&”. Each comparison starts with “ x ”, followed by *greater-or-equals* operator “ $>=$ ” or *less-or-equals* operator “ $<=$ ”, followed by an integer. When two comparisons are on the same line, the first one is always *greater-or-equals*, followed by *less-or-equals*.

All lines, but the last one, are terminated by *boolean or* operator “ $||$ ”. All tokens on a line are separated by a single space and there are no trailing or leading spaces.

Output

Write the refactored expression to the output file in the same format as in the input. You can arrange lines in any order, as long as the resulting expression has the right format, produces the same boolean result on all integer numbers x , and contains the minimal possible number of integer numbers in its representation. Numbers must be formatted without leading zeros and there must be precisely one space between tokens on a line.

Write a single line with the word “**true**” if the expression is true on all integers. Write a single line with the word “**false**” if the expression is false on all integers.

Example

hard.in	hard.out
x >= 5 && x <= 10 x >= 7 && x <= 20 x <= 2 x >= 21 && x <= 25 x >= 8 && x <= 10 x >= 100	x <= 2 x >= 5 && x <= 25 x >= 100
x >= 10 && x <= 0	false
x <= 10 x >= 0	true
x >= -32768	true

Problem I. Indiana Jones and the Uniform Cave

Input file: standard input
Output file: standard output

Indiana Jones had stuck in the Uniform Cave. There are many round chambers in the cave, and all of them are indistinguishable from each other. Each chamber has the same number of one-way passages evenly distributed along the chamber's wall. Passages are indistinguishable from each other, too. The Cave is magical. All passages lead to other chambers or to the same one. However, the last passage, after all passages are visited, leads to the treasure. Even the exact number of chambers is a mystery. It is known that each chamber is reachable from each other chamber using the passages.

Dr. Jones noticed that each chamber has a stone in the center. He decided to use these stones to mark chambers and passages. A stone can be placed to the left or to the right of one of the passages. When Indiana enters the chamber all that he can observe is the location of the stone in the chamber. He can move the stone to the desired location and take any passage leading out of the chamber.

Help Indiana Jones to pass every passage in the Uniform Cave and find the treasure room.

Interaction Protocol

First, the testing system writes the number m — the number of passages in each chamber ($1 \leq m \leq 20$).

Dr. Jones enters the chamber and sees, on the next line, where the stone is placed: either in the **center** of the chamber or to the **left**, or to the **right** of some passage. On the first visit to the chamber, the stone is in the center.

Your solution shall output Indiana's actions: the number and the side of the passage to place the stone to, and the number of the passage to take. Both numbers are relative to the passage marked by the stone, counting clockwise from 0 to $m - 1$. If the stone is in the center of the chamber, the origin is random.

For example, "3 left 1" tells that Dr. Jones moves the stone three passages clockwise and places it to the left of the passage, then Dr. Jones takes the passage to the right of the initial stone position.

After each move testing system tells either the location of the stone in the next chamber or "**treasure**", if Dr. Jones had found it. The testing system writes "**treasure**" when all the passages are visited.

If Dr. Jones does not find the treasure room after 10 000 passages taken, he starves to death, and your solution receives the "Wrong Answer" outcome. You also receive this outcome if your solution terminates before all passages were taken.

The total number of chambers in the cave n is unknown, but you may assume that $1 \leq n \leq 20$ and each chamber is reachable from every other chamber.

Example

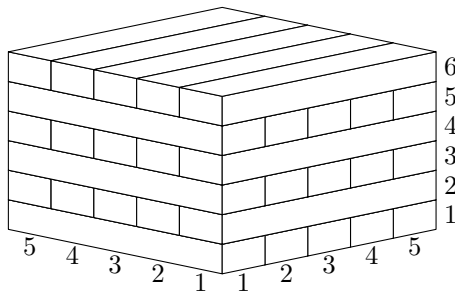
standard input	standard output
2	1 left 0
center	0 left 0
left	1 right 0
center	0 left 0
left	1 right 0
right	
treasure	

Dr. Jones enters the example cave and sees that the stone in the first chamber is in the center. He marks the chamber by placing the stone on the left of some passage and takes another one. He sees the chamber where the stone is on the left of the passage, so he is in the first chamber again. He leaves the stone in place and takes the passage marked by it. This passage leads to the second chamber. He marks it by placing the stone on the right of some passage and takes another one. He is in the first chamber again, so he returns to the second chamber and takes the remaining passage. This passage leads to the treasure.

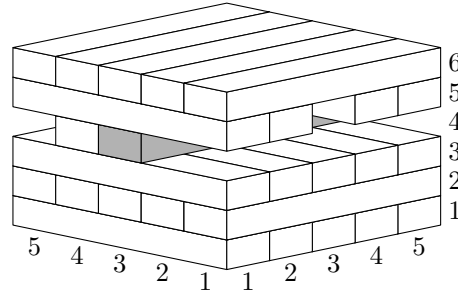
Problem J. Jenga Boom

Input file: `jenga.in`
Output file: `jenga.out`

Jane is a game designer and she designs the next version of Jenga Boom, where the blocks have dimensions of $1 \times w \times wn$ instead of the ordinary $1 \times 2 \times 6$. As usual, the initial tower is created at the game start. It consists of the blocks in levels of n blocks placed next to each other along their long sides and at a right angle to the previous level. Players remove blocks from the tower in turns, until the tower collapses.



Initial tower



Tower before collapse

Jane wants to build a game simulator that helps her to decide the best n and w . The simulator shall compute the moment when the tower collapses if blocks are removed in the specified order. Tower collapses if there is a cross-section between levels such that the center of the mass of the levels above it does not belong to or is at the edge of the convex hull of the previous level projection.

Input

The first line contains two integers n and w that define the dimensions of the block as described in the problem statement ($1 \leq n, w \leq 10\,000$). The second line also contains two integers: h — the number of levels in the tower and m — the number of removed blocks ($1 \leq h, m \leq 5\,000$).

The following m lines contain coordinates of the removed blocks with two integers each: l_i — the level of the block, counting from the bottom and k_i — the position of the block, counting from the edge nearest to the players. Blocks are removed one by one and no block is removed twice ($1 \leq l_i \leq h, 1 \leq k_i \leq n$).

Output

In the first line output “yes” if the tower collapses, and “no” otherwise. In the former case, output the number of the block (starting from 1) that was removed just before the collapse.

Example

<code>jenga.in</code>	<code>jenga.out</code>
5 2 6 6 4 1 4 2 4 5 5 3 4 3 1 1	yes 5
3 3 10 1 10 3	no
2 2 2 1 1 1	yes 1

Problem K. Kids Designing Kids

Input file: kids.in
Output file: kids.out

Kevin and Kimberly have freckles on their foreheads.

They both drew their freckle pictures on sheets of paper. Each picture is a rectangle of “pixels”: every cell either has a freckle or does not.

They are jokingly proposing that when they grow up, marry and have a child, her freckle picture will be a result of the following procedure:

Kevin’s and Kimberly’s pictures will be moved by a parallel translation, and then in each cell, a child will have a freckle if and only if exactly one of the parents has a freckle in this position.

Now they wonder, whether there is a parallel translation that will give their child a specific freckle picture (for example, a lightning), and what is this parallel translation.

Input

The first line contains two integers, h_1 and w_1 ($1 \leq h_1, w_1 \leq 1000$) — the height and the width of Kevin’s freckle picture. Each of the next h_1 lines contain w_1 characters ‘*’ and ‘.’. Character ‘*’ means that there is a freckle, and ‘.’ that there is not.

The following lines contain Kimberly’s picture in the same format. Its height and width h_2 and w_2 follow the same constraints.

It is guaranteed that Kevin and Kimberly has at least one freckle each.

The following lines contain the picture they want their child to have. Its dimensions h_3 and w_3 also have the same constraints.

Output

On the first line output “YES” if the desired picture can be received, and “NO” otherwise.

If the answer is positive, on the second line output two integers, x and y , with the following meaning. If you overlay the pictures so that their upper left corners coincide, then move Kimberly’s picture x cells right (negative number means moving picture left) and y cells down (negative number means moving picture up), and then apply the procedure described above, the resulting picture will can moved by a parallel translation to coincide with the third picture from the input file.

Example

kids.in	kids.out
3 3 ..* *. *.* 3 3 **. ..* *. 5 2 .* *. ** .* *.	YES 0 2

Problem L. List of Primes

Input file: `list.in`
Output file: `list.out`

Lidia likes sets of prime numbers. When she is bored, she starts writing them down into the Extremely Long Notebook for Prime Sets.

Elements of each set are written down in ascending order. Each set of prime numbers will appear in her notebook eventually. A set with a smaller sum always appears before a set with a larger sum. Sets with the same sum are sorted in ascending lexicographical order: they are compared by the first element, if they are equal then by second element, and so on.

Just in case someone decides to parse her notebook, she writes down her sets in machine-readable JSON format. Of course, she puts a space after each comma. Here's the beginning of her notebook:

[2], [3], [2, 3], [5], [2, 5], [7], [3, 5], [2, 7], [2, 3, 5], [3, 7], [11], [2, 3, 7], [5, 7], [2, 11], [13], [2, 5, 7],

Lidia wants to double-check her work, so here is her request to you: given two integers, a and b , output a substring of her notebook from position a to position b (inclusive, counting from 1).

Input

The first line contains two integers, a and b ($1 \leq a \leq b \leq 10^{18}$; $b - a \leq 10^5$).

Output

Output the substring of the notebook described in the problem statement from position a to position b . You must write a line with exactly $b - a + 1$ characters, including any leading and/or trailing spaces.

Example

<code>list.in</code>	<code>list.out</code>
1 35	[2], [3], [2, 3], [5], [2, 5], [7],
36 41	[3, 5

Problem M. Mole Tunnels

Input file: mole.in
 Output file: mole.out

Moles create tunnels for two purposes — for traveling and for feeding. In this problem we investigate one tunnel system which was built by moles. It consists of n holes and $n - 1$ tunnels connecting them. Let us number all holes from 1 to n . Then for all $i > 1$, a hole number i is connected by a tunnel to a hole number $\lfloor \frac{i}{2} \rfloor$. Tunnels are bidirectional. For each hole i we know the *amount of food* c_i in that hole. It means that there is enough food for exactly c_i moles in that hole.

There are m moles living in the tunnel system. For mole i you are given a number p_i — a hole number where mole i is currently sleeping. In the morning the first k moles wake up and want to eat while $m - k$ others are sleeping. Each of them chooses some hole and crawls to it. They are quite smart so they want to minimize the total distance travelled. The distance travelled by one mole is the total number of tunnels which it uses to get from one hole to another. The first k moles who woke up early want to move in such a way, so that there is enough food at the holes they choose to crawl to for them. It means that after all movements are done in hole i there are no more than c_i not sleeping moles.

You must to find the minimum total distance for all k from 1 to m . It is guaranteed that there always exists a way for all moles to eat.

Input

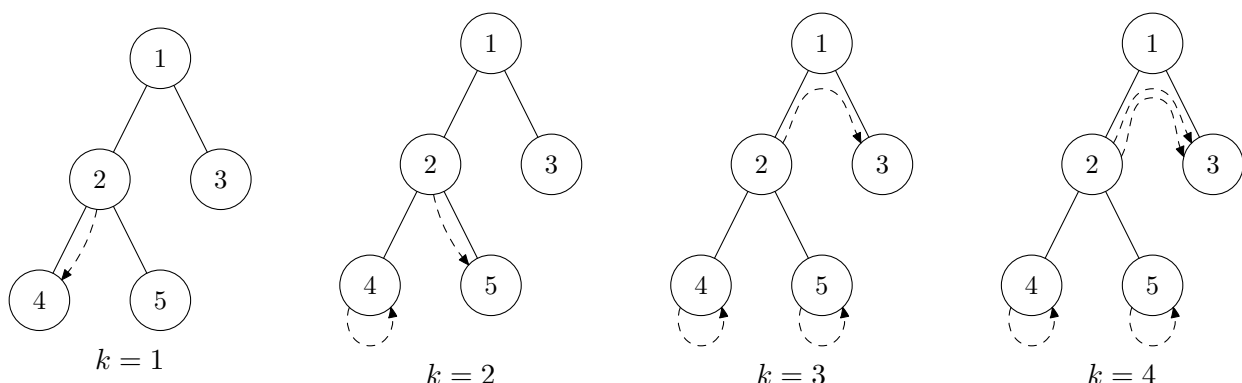
The first line contains two integers n and m ($1 \leq n, m \leq 10^5$) — the number of holes and moles. The second line contains n integers c_i ($0 \leq c_i \leq m$) — the amount of food in hole i . The third line contains m integers p_i ($1 \leq p_i \leq n$) — the starting positions of the moles.

Output

You must print m numbers. The k -th number is the minimum total distance the first k moles need to travel if they woke up first.

Example

mole.in	mole.out
5 4 0 0 4 1 1 2 4 5 2	1 1 2 4



Dashed arrows show possible moles movement which minimizes total distance. For example for $k = 2$ first mole goes from hole 2 to hole 5 and second mole stays in hole 4.