Codeforces LATOKEN Round 1 (Div. 1 + Div. 2)

A. Colour the Flag

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Today we will be playing a red and white colouring game (no, this is not the Russian Civil War; these are just the colours of the Canadian flag).

You are given an $n \times m$ grid of "R", "W", and "." characters. "R" is red, "W" is white and "." is blank. The neighbours of a cell are those that share an edge with it (those that only share a corner do not count).

Your job is to colour the blank cells red or white so that every red cell only has white neighbours (and no red ones) and every white cell only has red neighbours (and no white ones). You are not allowed to recolour already coloured cells.

Input

The first line contains t ($1 \leq t \leq 100$), the number of test cases.

In each test case, the first line will contain n ($1 \leq n \leq 50$) and m ($1 \leq m \leq 50$), the height and width of the grid respectively.

The next n lines will contain the grid. Each character of the grid is either 'R', 'W', or '.'.

Output

For each test case, output "YES" if there is a valid grid or "NO" if there is not.

If there is, output the grid on the next n lines. If there are multiple answers, print any.

In the output, the "YES"s and "NO"s are case-insensitive, meaning that outputs such as "yEs" and "nO" are valid. However, **the grid is case-sensitive**.

Example

input
3 4 6 .R.... .... .W.... 4 4 .R.W .... 5 1 R W R W R
output
YES WRWRWR RWRWRW WRWRWR RWRWRW NO YES R W R W R

Note

The answer for the first example case is given in the example output, and it can be proven that no grid exists that satisfies the requirements of the second example case. In the third example all cells are initially coloured, and the colouring is valid.

B. Histogram Ugliness

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input

output: standard output

Little Dormi received a histogram with n bars of height $a_1, a_2, \ldots, a_n$ for Christmas. However, the more he played with his new histogram, the more he realized its imperfections, so today he wanted to modify it to his liking.

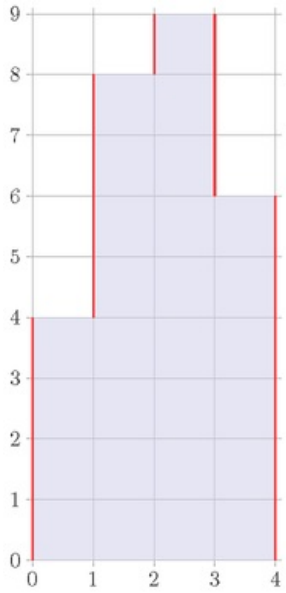
To modify the histogram, Little Dormi is able to perform the following operation an arbitrary number of times:

- Select an index i ($1 \leq i \leq n$) where $a_i > 0$, and assign $a_i := a_i - 1$.

Little Dormi defines the ugliness score of his histogram (after performing some number of operations) as the sum of the vertical length of its outline and the number of operations he performed on it. And to make the histogram as perfect as possible, he would like to minimize the ugliness score after modifying it with some number of operations.

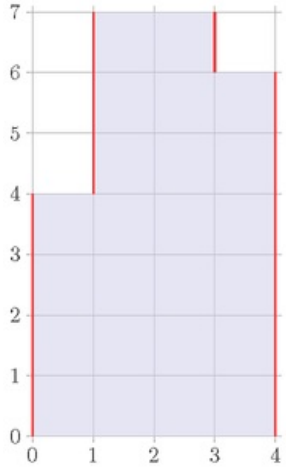
However, as his histogram is very large, Little Dormi is having trouble minimizing the ugliness score, so as Little Dormi's older brother, help him find the minimal ugliness.

Consider the following example where the histogram has 4 columns of heights $4, 8, 9, 6$:



The blue region represents the histogram, and the red lines represent the vertical portion of the outline. Currently, the vertical length of the outline is $4 + 4 + 1 + 3 + 6 = 18$, so if Little Dormi does not modify the histogram at all, the ugliness would be 18.

However, Little Dormi can apply the operation once on column 2 and twice on column 3, resulting in a histogram with heights $4, 7, 7, 6$:



Now, as the total vertical length of the outline (red lines) is $4 + 3 + 1 + 6 = 14$, the ugliness is $14 + 3 = 17$ dollars. It can be proven that this is optimal.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). Description of the test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 4 \cdot 10^5$).

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i \leq 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output

For each test case output one integer, the minimal ugliness Little Dormi can achieve with the histogram in that test case.

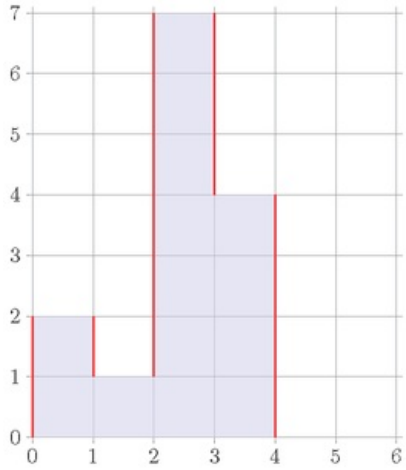
Example

input
2 4 4 8 9 6 6 2 1 7 4 0 0
output
17 12

Note

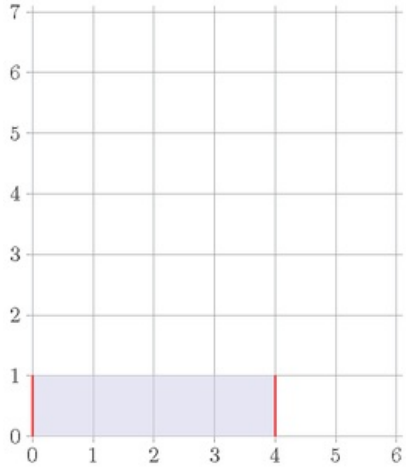
Example 1 is the example described in the statement.

The initial histogram for example 2 is given below:



The ugliness is currently $2 + 1 + 6 + 3 + 4 = 16$.

By applying the operation once on column 1, six times on column 3, and three times on column 4, we can end up with a histogram with heights $1, 1, 1, 1, 0, 0$:



The vertical length of the outline is now $1 + 1 = 2$ and Little Dormi made $1 + 6 + 3 = 10$ operations, so the final ugliness is $2 + 10 = 12$, which can be proven to be optimal.

C. Little Alawn's Puzzle

time limit per test: 2.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

When he's not training for IOI, Little Alawn enjoys playing with puzzles of various types to stimulate his brain. Today, he's playing with a puzzle that consists of a $2 \times n$ grid where each row is a permutation of the numbers $1, 2, 3, \ldots, n$.

The goal of Little Alawn's puzzle is to make sure no numbers on the same column or row are the same (we'll call this state of the puzzle as solved), and to achieve this he is able to swap the numbers in any column. However, after solving the puzzle many times, Little Alawn got bored and began wondering about the number of possible solved configurations of the puzzle he could achieve from an initial **solved** configuration only by swapping numbers in a column.

Unfortunately, Little Alawn got stuck while trying to solve this harder problem, so he was wondering if you could help him with it. Find the answer modulo $10^9 + 7$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). Description of the test cases

follows.

The first line of each test case contains a single integer n ($2 \le n \le 4 \cdot 10^5$).

The next two lines of each test case describe the initial state of the puzzle grid. Each line will be a permutation of the numbers $1, 2, 3, \ldots, n$ and the numbers in each column and row will be pairwise distinct.

It is guaranteed that the sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output
For each test case output a single integer, the number of possible solved configurations of the puzzle Little Alawn can achieve from an initial solved configuration only by swapping numbers in a column. As the answer can be very large, please output it modulo $10^9 + 7$.

The answer for each test case should be on a separate line.

Example

input
2 4 1 4 2 3 3 2 1 4 8 2 6 5 1 4 3 7 8 3 8 7 5 1 2 4 6
output
2 8

Note
The two possible puzzle configurations for example 1 are:

- $[1, 4, 2, 3]$ in the first row and $[3, 2, 1, 4]$ in the second;
- $[3, 2, 1, 4]$ in the first row and $[1, 4, 2, 3]$ in the second.

D. Lost Tree

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

Little Dormi was faced with an awkward problem at the carnival: he has to guess the edges of an unweighted tree of n nodes! The nodes of the tree are numbered from 1 to n.

The game master only allows him to ask one type of question:

- Little Dormi picks a node r ($1 \le r \le n$), and the game master will reply with an array $d_1, d_2, \ldots, d_n$, where d_i is the length of the shortest path from node r to i, for all $1 \le i \le n$.

Additionally, to ~~make the game unfair~~ challenge Little Dormi the game master will allow at most $\lceil \frac{n}{2} \rceil$ questions, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

Faced with the stomach-churning possibility of not being able to guess the tree, Little Dormi needs your help to devise a winning strategy!

Note that the game master creates the tree before the game starts, and does not change it during the game.

Input
The first line of input contains the integer n ($2 \le n \le 2\,000$), the number of nodes in the tree.

You will then begin interaction.

Output
When your program has found the tree, first output a line consisting of a single "!" followed by $n - 1$ lines each with two space separated integers a and b, denoting an edge connecting nodes a and b ($1 \le a, b \le n$). Once you are done, terminate your program normally immediately after flushing the output stream.

You may output the edges in any order and an edge (a, b) is considered the same as an edge (b, a). Answering is not considered as a query.

Interaction
After taking input, you may make at most $\lceil \frac{n}{2} \rceil$ queries. Each query is made in the format "? r", where r is an integer $1 \le r \le n$ that denotes the node you want to pick for that query.

You will then receive n space separated integers $d_1, d_2, \ldots, d_n$, where d_i is the length of the shortest path from node r to i, followed by a newline.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

If at any point you make an invalid query or try to make more than $\lceil \frac{n}{2} \rceil$ queries, the interaction will terminate immediately and you will receive a Wrong Answer verdict.

Hacks

To hack a solution, use the following format.

The first line contains the integer n ($2 \leq n \leq 2\,000$).

The next $n - 1$ lines contain two integers u and v ($1 \leq u, v \leq n$) denoting an edge between u and v ($u \neq v$). These $n - 1$ edges must form a tree.

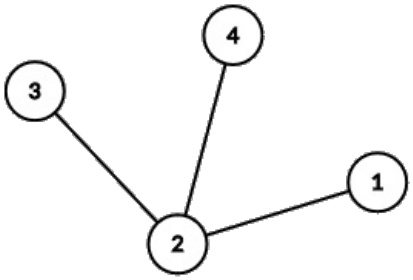
Examples

input
4 0 1 2 2 1 0 1 1
output
? 1 ? 2 ! 4 2 1 2 2 3

input
5 2 2 1 1 0
output
? 5 ! 4 5 3 5 2 4 1 3

Note

Here is the tree from the first example.

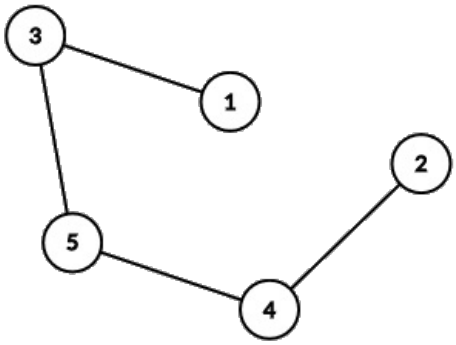


Notice that the edges can be output in any order.

Additionally, here are the answers for querying every single node in example 1:

- 1: $[0, 1, 2, 2]$
- 2: $[1, 0, 1, 1]$
- 3: $[2, 1, 0, 2]$
- 4: $[2, 1, 2, 0]$

Below is the tree from the second example interaction.



Lastly, here are the answers for querying every single node in example 2:

- 1: $[0, 4, 1, 3, 2]$
- 2: $[4, 0, 3, 1, 2]$
- 3: $[1, 3, 0, 2, 1]$
- 4: $[3, 1, 2, 0, 1]$
- 5: $[2, 2, 1, 1, 0]$

E. Lost Array

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

Note: the XOR-sum of an array $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) is defined as $a_1 \oplus a_2 \oplus \ldots \oplus a_n$, where $\oplus$ denotes the bitwise XOR operation.

Little Dormi received an array of n integers $a_1, a_2, \ldots, a_n$ for Christmas. However, while playing with it over the winter break, he accidentally dropped it into his XOR machine, and the array got lost.

The XOR machine is currently configured with a query size of k (which you cannot change), and allows you to perform the following type of query: by giving the machine k **distinct** indices $x_1, x_2, \ldots, x_k$, it will output $a_{x_1} \oplus a_{x_2} \oplus \ldots \oplus a_{x_k}$.

As Little Dormi's older brother, you would like to help him recover the **XOR-sum** of his array $a_1, a_2, \ldots, a_n$ by querying the XOR machine.

Little Dormi isn't very patient, so to be as fast as possible, you must query the XOR machine the **minimum** number of times to find the XOR-sum of his array. Formally, let d be the minimum number of queries needed to find the XOR-sum of any array of length n with a query size of k. Your program will be accepted if you find the correct XOR-sum in at most d queries.

Lastly, you also noticed that with certain configurations of the machine k and values of n, it may not be possible to recover the XOR-sum of Little Dormi's lost array. If that is the case, you should report it as well.

The array $a_1, a_2, \ldots, a_n$ is fixed before you start querying the XOR machine and does not change with the queries.

Input

The only line of input contains the integers n and k ($1 \leq n \leq 500$, $1 \leq k \leq n$), the length of the lost array and the configured query size of the XOR machine.

Elements of the original array satisfy $1 \leq a_i \leq 10^9$.

It can be proven that that if it is possible to recover the XOR sum under the given constraints, it can be done in at most 500 queries. That is, $d \leq 500$.

After taking n and k, begin interaction.

Output

If it is impossible to recover the XOR-sum of the array, output -1 **immediately** after taking n and k. Do not begin interaction.

Otherwise, when your program finds the XOR-sum of the lost array $a_1, a_2, \ldots, a_n$, report the answer in the following format: "! x", where x is the XOR sum of the array $a_1, a_2, \ldots, a_n$, and terminate your program normally immediately after flushing the output stream.

Note that answering does not count as a query.

Interaction

Each query is made in the format "? b", where b is an array of exactly k **distinct** integers from 1 to n denoting the indices of the elements in the lost array that you want to query the XOR sum of.

You will then receive an integer x, the XOR sum of the queried elements. It can be proven that $0 \leq x \leq 2 \cdot 10^9$ will always be true.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

If at any point you make an invalid query or try to make more than 500 queries (which is the hard limit), the interaction will terminate immediately and give you a Wrong Answer verdict. Note that if you exceed d queries, the interaction will continue normally unless you also exceed the 500 query hard limit, though you will still receive a Wrong Answer verdict either way.

Hacks

To hack a solution, use the following format.

The first line contains the integers n and k ($1 \le n \le 500$, $1 \le k \le n$).

The second line contains the the array $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

Examples

input
5 3 4 0 1
output
? 1 2 3 ? 2 3 5 ? 4 1 5 ! 7

input
3 2
output
-1

Note

In the first example interaction, the array $a_1, a_2, \ldots, a_n$ is $2, 1, 7, 5, 6$ and its XOR-sum is 7.

The first query made asks for indices $1, 2, 3$, so the response is $a_1 \oplus a_2 \oplus a_3 = 2 \oplus 1 \oplus 7 = 4$.

The second query made asks for indices $2, 3, 5$, so the response is $a_2 \oplus a_3 \oplus a_5 = 1 \oplus 7 \oplus 6 = 0$.

The third query made asks for indices $4, 1, 5$, so the response is $a_4 \oplus a_1 \oplus a_5 = 5 \oplus 2 \oplus 6 = 1$. Note that the indices may be output in any order.

Additionally, even though three queries were made in the example interaction, it is just meant to demonstrate the interaction format and **does not** necessarily represent an optimal strategy.

In the second example interaction, there is no way to recover the XOR-sum of Little Dormi's array no matter what is queried, so the program immediately outputs -1 and exits.

F1. Falling Sand (Easy Version)

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

This is the easy version of the problem. The difference between the versions is the constraints on a_i. You can make hacks only if all versions of the problem are solved.

Little Dormi has recently received a puzzle from his friend and needs your help to solve it.

The puzzle consists of an upright board with n rows and m columns of cells, some empty and some filled with blocks of sand, and m non-negative integers $a_1, a_2, \ldots, a_m$ ($0 \le a_i \le n$). In this version of the problem, a_i will be **equal to** the number of blocks of sand in column i.

When a cell filled with a block of sand is disturbed, the block of sand will fall from its cell to the sand counter at the bottom of the column (each column has a sand counter). While a block of sand is falling, other blocks of sand that are adjacent at any point to the falling block of sand will also be disturbed and start to fall. Specifically, a block of sand disturbed at a cell (i, j) will pass through all cells below and including the cell (i, j) within the column, disturbing all adjacent cells along the way. Here, the cells adjacent to a cell (i, j) are defined as $(i - 1, j)$, $(i, j - 1)$, $(i + 1, j)$, and $(i, j + 1)$ (if they are within the grid). Note that the newly falling blocks can disturb other blocks.

In one operation you are able to disturb any piece of sand. The puzzle is solved when there are **at least** a_i blocks of sand counted in the i-th sand counter for each column from 1 to m.

You are now tasked with finding the minimum amount of operations in order to solve the puzzle. Note that Little Dormi will never give you a puzzle that is impossible to solve.

Input

The first line consists of two space-separated positive integers n and m ($1 \leq n \cdot m \leq 400\,000$).

Each of the next n lines contains m characters, describing each row of the board. If a character on a line is '.', the corresponding cell is empty. If it is '#', the cell contains a block of sand.

The final line contains m non-negative integers $a_1, a_2, \ldots, a_m$ ($0 \leq a_i \leq n$) — the minimum amount of blocks of sand that needs to fall below the board in each column. In this version of the problem, a_i will be **equal to** the number of blocks of sand in column i.

Output

Print one non-negative integer, the minimum amount of operations needed to solve the puzzle.

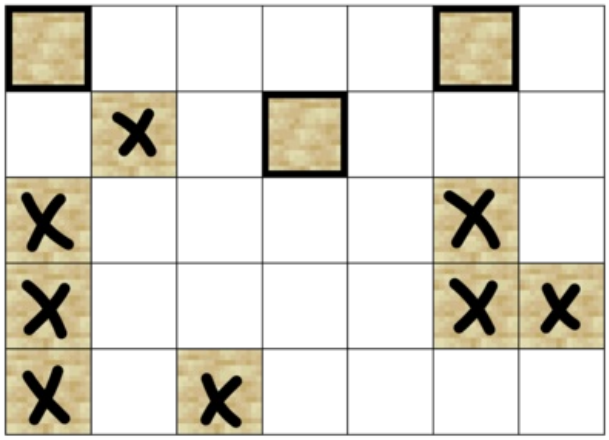
Examples

input
5 7 #....#. .#.#... #....#. #....## #.#.... 4 1 1 1 0 3 1
output
3

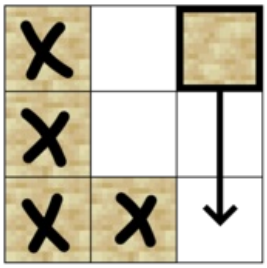
input
3 3 #.# #.. ##. 3 1 1
output
1

Note

For example 1, by disturbing both blocks of sand on the first row from the top at the first and sixth columns from the left, and the block of sand on the second row from the top and the fourth column from the left, it is possible to have all the required amounts of sand fall in each column. It can be proved that this is not possible with fewer than 3 operations, and as such the answer is 3. Here is the puzzle from the first example.



For example 2, by disturbing the cell on the top row and rightmost column, one can cause all of the blocks of sand in the board to fall into the counters at the bottom. Thus, the answer is 1. Here is the puzzle from the second example.



F2. Falling Sand (Hard Version)

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

This is the hard version of the problem. The difference between the versions is the constraints on a_i. You can make hacks only if all versions of the problem are solved.

Little Dormi has recently received a puzzle from his friend and needs your help to solve it.

The puzzle consists of an upright board with n rows and m columns of cells, some empty and some filled with blocks of sand, and m non-negative integers $a_1, a_2, \ldots, a_m$ ($0 \le a_i \le n$). In this version of the problem, a_i will always be **not greater than** the number of blocks of sand in column i.

When a cell filled with a block of sand is disturbed, the block of sand will fall from its cell to the sand counter at the bottom of the column (each column has a sand counter). While a block of sand is falling, other blocks of sand that are adjacent at any point to the falling block of sand will also be disturbed and start to fall. Specifically, a block of sand disturbed at a cell (i, j) will pass through all cells below and including the cell (i, j) within the column, disturbing all adjacent cells along the way. Here, the cells adjacent to a cell (i, j) are defined as $(i-1, j)$, $(i, j-1)$, $(i+1, j)$, and $(i, j+1)$ (if they are within the grid). Note that the newly falling blocks can disturb other blocks.

In one operation you are able to disturb any piece of sand. The puzzle is solved when there are **at least** a_i blocks of sand counted in the i-th sand counter for each column from 1 to m.

You are now tasked with finding the minimum amount of operations in order to solve the puzzle. Note that Little Dormi will never give you a puzzle that is impossible to solve.

Input

The first line consists of two space-separated positive integers n and m ($1 \le n \cdot m \le 400\,000$).

Each of the next n lines contains m characters, describing each row of the board. If a character on a line is '.', the corresponding cell is empty. If it is '#', the cell contains a block of sand.

The final line contains m non-negative integers $a_1, a_2, \ldots, a_m$ ($0 \le a_i \le n$) — the minimum amount of blocks of sand that needs to fall below the board in each column. In this version of the problem, a_i will always be **not greater than** the number of blocks of sand in column i.

Output

Print one non-negative integer, the minimum amount of operations needed to solve the puzzle.

Examples

input
5 7 #....#. .#.#... #....#. #....## #.#.... 4 1 1 1 0 3 1
output
3

input
3 3 #.# #.. ##. 3 1 1
output
1

input

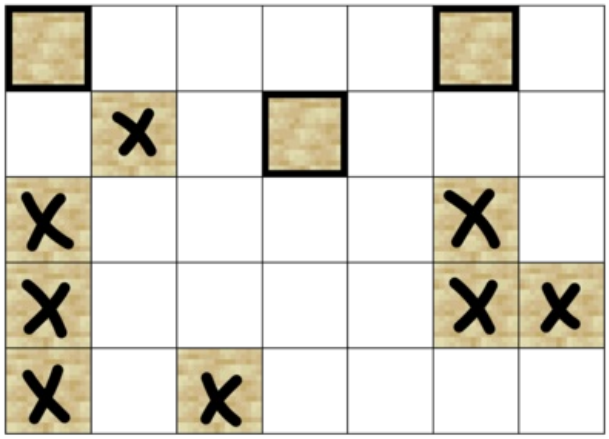
```
7 5
.#..#
#....
..##.
..##.
..###
..#..
#.##.
0 0 2 4 2
```

output

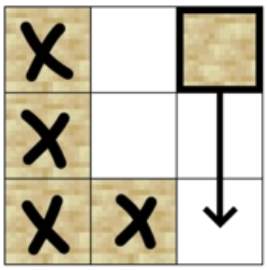
```
1
```

Note

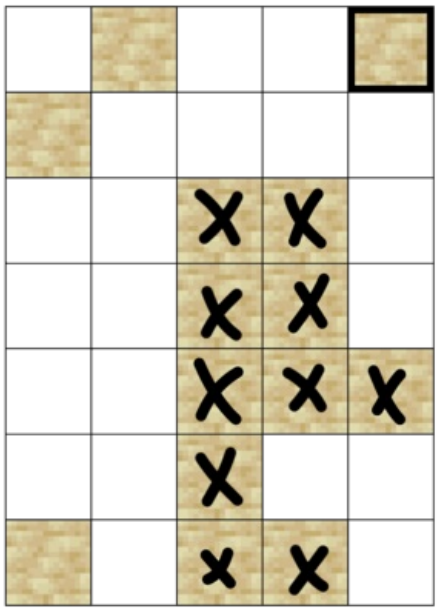
For example 1, by disturbing both blocks of sand on the first row from the top at the first and sixth columns from the left, and the block of sand on the second row from the top and the fourth column from the left, it is possible to have all the required amounts of sand fall in each column. It can be proved that this is not possible with fewer than 3 operations, and as such the answer is 3. Here is the puzzle from the first example.



For example 2, by disturbing the cell on the top row and rightmost column, one can cause all of the blocks of sand in the board to fall into the counters at the bottom. Thus, the answer is 1. Here is the puzzle from the second example.



For example 3, by disturbing the cell on the top row and rightmost column, it is possible to have all the required amounts of sand fall in each column. It can be proved that this is not possible with fewer than 1 operation, and as such the answer is 1. Here is the puzzle from the third example.



G. A New Beginning

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Annie has gotten bored of winning every coding contest and farming unlimited rating. Today, she is going to farm potatoes instead.

Annie's garden is an infinite 2D plane. She has n potatoes to plant, and the i-th potato must be planted at (x_i, y_i). Starting at the point $(0, 0)$, Annie begins walking, in one step she can travel one unit **right** or **up** (increasing her x or y coordinate by 1 respectively). At any point (X, Y) during her walk she can plant some potatoes at arbitrary points using her potato gun, consuming $\max(|X - x|, |Y - y|)$ units of energy in order to plant a potato at (x, y). Find the minimum total energy required to plant every potato.

Note that Annie may plant any number of potatoes from any point.

Input

The first line contains the integer n ($1 \le n \le 800\,000$).

The next n lines contain two integers x_i and y_i ($0 \le x_i, y_i \le 10^9$), representing the location of the i-th potato. It is possible that some potatoes should be planted in the same location.

Output

Print the minimum total energy to plant all potatoes.

Examples

input
2 1 1 2 2
output
0

input
2 1 1 2 0
output
1

input
3 5 5 7 7 4 9
output
2

input
10 5 1 4 0 9 6 0 2 10 1 9 10 3 10 0 10 8 9 1 5
output
19

input
10 1 1 2 2 2 0 4 2 4 0 2 0 0 2 4 0 4 2

5 1
output
6

Note

In example 1, Annie can travel to each spot directly and plant a potato with no energy required.

In example 2, moving to $(1, 0)$, Annie plants the second potato using 1 energy. Next, she travels to $(1, 1)$ and plants the first potato with 0 energy.

H. Lost Nodes

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

As he qualified for IOI this year, Little Ericyi was given a gift from all his friends: a tree of n nodes!

On the flight to IOI Little Ericyi was very bored, so he decided to play a game with Little Yvonne with his new tree. First, Little Yvonne selects two (not necessarily different) nodes a and b on the tree (without telling Ericyi), and then gives him a hint f (which is some node on the path from a to b).

Then, Little Ericyi is able to ask the following question repeatedly:

- If I rooted the tree at node r (Ericyi gets to choose r), what would be the Lowest Common Ancestor of a and b?

Little Ericyi's goal is to find the nodes a and b, and report them to Little Yvonne.

However, Little Yvonne thought this game was too easy, so before he gives the hint f to Little Ericyi, he also wants him to first find the maximum number of queries required to determine a and b over all possibilities of a, b, and f assuming Little Ericyi plays optimally. Little Ericyi defines an optimal strategy as one that makes the minimum number of queries. Of course, once Little Ericyi replies with the maximum number of queries, Little Yvonne will only let him use that many queries in the game.

The tree, a, b, and f are all fixed before the start of the game and do not change as queries are made.

Interaction

First read a line containing the integer n ($1 \leq n \leq 10^5$), the number of nodes in the tree.

The next $n - 1$ lines describe Little Ericyi's tree. These lines contain two integers u and v ($1 \leq u, v \leq n$) denoting an edge between u and v ($u \neq v$). It is guaranteed that these edges form a tree.

After that you should output k, the maximum number of queries needed to determine a and b over all possibilities of a, b, and f assuming Little Ericyi plays optimally. You should output end of line and flush the output after printing k.

After that read a line containing the integer f ($1 \leq f \leq n$) — the hint: a node on the path from a to b, inclusive.

After that, you can start making queries. You will be limited to making at most k queries, where k is the number you printed.

Each query is made in the format "`? r`", where r is an integer $1 \leq r \leq n$ denoting the root node you want for the query.

You will then receive an integer x ($1 \leq x \leq n$), the Lowest Common Ancestor of a and b if the tree was rooted at r.

When your program has found the nodes a, b, report the answer in the following format: "`! a b`", where a and b are the two hidden nodes and terminate your program normally immediately after flushing the output stream. You may output a and b in any order.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

If at any point you make an invalid output or make more than k queries, the interaction will terminate and you will receive a Wrong Answer verdict. An invalid output is defined as either an invalid query or a value of k less than 0 or greater than n.

Hacks

To hack a solution, use the following format:

The first line contains the integer n ($1 \leq n \leq 10^5$).

The next $n - 1$ lines contain two integers u and v ($1 \leq u, v \leq n$) denoting an edge between u and v ($u \neq v$). These $n - 1$ edges

must form a tree.

The next line of input contains the nodes a and b ($1 \leq a, b \leq n$), separated by a space.

The final line of input contains the integer f ($1 \leq f \leq n$). Node f should be on the simple path from a to b (inclusive).

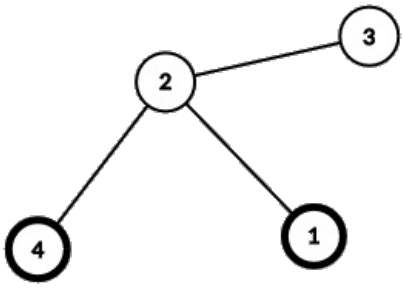
Examples

input
4 3 2 2 1 2 4 1 1 2 2
output
3 ? 1 ? 2 ? 3 ! 4 1

input
5 3 1 1 4 4 5 4 2 1 4 1 4
output
3 ? 4 ? 1 ? 5 ! 1 4

Note

Here is the the tree from the first sample interaction. Nodes a and b are highlighted.

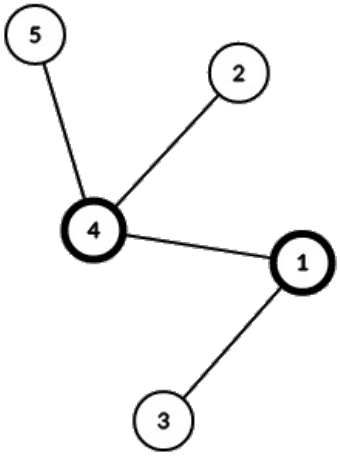


Notice that a and b can be output in any order.

Additionally, here are the answers to querying every single node $1, 2, \ldots, n$ for your convenience:

- 1: 1
- 2: 2
- 3: 2
- 4: 4

Here is the the tree from the second sample interaction. Again, nodes a and b are highlighted.



Lastly, here are the answers to querying every single node $1, 2, \ldots, n$ (in example 2) for your convenience:

- 1: 1
- 2: 4
- 3: 1
- 4: 4
- 5: 4
