A. Windblume Ode,

1 second, 256 megabytes, standard input, standard output

A bow adorned with nameless flowers that bears the earnest hopes of an equally nameless person.

You have obtained the elegant bow known as the Windblume Ode. Inscribed in the weapon is an array of n ($n \geq 3$) positive **distinct** integers (i.e. different, no duplicates are allowed).

Find the largest subset (i.e. having the maximum number of elements) of this array such that its sum is a composite number. A positive integer x is called composite if there exists a positive integer y such that $1 < y < x$ and x is divisible by y.

If there are multiple subsets with this largest size with the composite sum, you can output any of them. It can be proven that under the constraints of the problem such a non-empty subset always exists.

Input

Each test consists of multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 100$). Description of the test cases follows.

The first line of each test case contains an integer n ($3 \leq n \leq 100$) — the length of the array.

The second line of each test case contains n **distinct** integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 200$) — the elements of the array.

Output

Each test case should have two lines of output.

The first line should contain a single integer x: the size of the largest subset with composite sum. The next line should contain x space separated integers representing the indices of the subset of the initial array.

input

```
4
3
8 1 2
4
6 9 4 2
9
1 2 3 4 5 6 7 8 9
3
200 199 198
```

output

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

In the first test case, the subset $\{a_2, a_1\}$ has a sum of 9, which is a composite number. The only subset of size 3 has a prime sum equal to 11. Note that you could also have selected the subset $\{a_1, a_3\}$ with sum $8 + 2 = 10$, which is composite as it's divisible by 2.

In the second test case, the sum of all elements equals to 21, which is a composite number. Here we simply take the whole array as our subset.

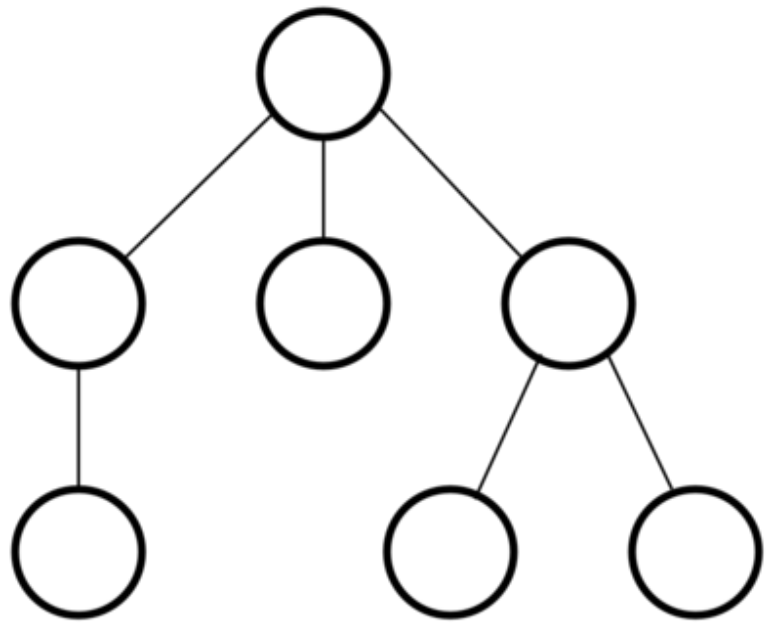
B. Omkar and Heavenly Tree,

2 seconds, 256 megabytes, standard input, standard output

Lord Omkar would like to have a tree with n nodes ($3 \leq n \leq 10^5$) and has asked his disciples to construct the tree. However, Lord Omkar has created m ($\mathbf{1 \leq m < n}$) restrictions to ensure that the tree will be as heavenly as possible.

A tree with n nodes is an connected undirected graph with n nodes and $n - 1$ edges. Note that for any two nodes, there is exactly one simple path between them, where a simple path is a path between two nodes that does not contain any node more than once.

Here is an example of a tree:



A restriction consists of 3 pairwise distinct integers, a, b, and c ($1 \leq a, b, c \leq n$). It signifies that node b cannot lie on the simple path between node a and node c.

Can you help Lord Omkar and become his most trusted disciple? You will need to find heavenly trees for multiple sets of restrictions. It can be shown that a heavenly tree will always exist for any set of restrictions under the given constraints.

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 two integers, n and m ($3 \leq n \leq 10^5$, $\mathbf{1 \leq m < n}$), representing the size of the tree and the number of restrictions.

The i-th of the next m lines contains three integers a_i, b_i, c_i ($1 \leq a_i, b_i, c_i \leq n$, **$a, b, c$ are distinct**), signifying that node b_i cannot lie on the simple path between nodes a_i and c_i.

It is guaranteed that the sum of n across all test cases will not exceed 10^5.

Output

For each test case, output $n - 1$ lines representing the $n - 1$ edges in the tree. On each line, output two integers u and v ($1 \leq u, v \leq n$, $u \neq v$) signifying that there is an edge between nodes u and v. Given edges have to form a tree that satisfies Omkar's restrictions.

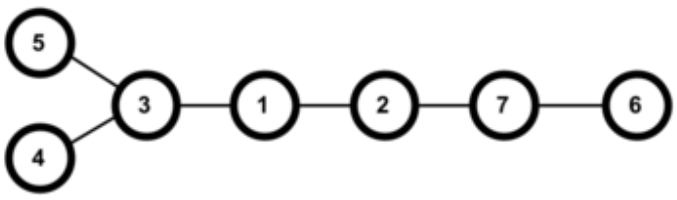
input

```
2
7 4
1 2 3
3 4 5
5 6 7
6 5 4
5 3
1 2 3
2 3 4
3 4 5
```

output

```
1 2
1 3
3 5
3 4
2 7
7 6
5 1
1 3
3 2
2 4
```

The output of the first sample case corresponds to the following tree:



For the first restriction, the simple path between 1 and 3 is $1, 3$, which doesn't contain 2. The simple path between 3 and 5 is $3, 5$, which doesn't contain 4. The simple path between 5 and 7 is $5, 3, 1, 2, 7$, which doesn't contain 6. The simple path between 6 and 4 is $6, 7, 2, 1, 3, 4$, which doesn't contain 5. Thus, this tree meets all of the restrictions.

The output of the second sample case corresponds to the following tree:

C. Omkar and Determination, 2 seconds, 256 megabytes, standard input, standard output

The problem statement looms below, filling you with determination.

Consider a grid in which some cells are empty and some cells are filled. Call a cell in this grid **exitable** if, starting at that cell, you can exit the grid by moving up and left through only empty cells. This includes the cell itself, so all filled in cells are not exitable. Note that you can exit the grid from any leftmost empty cell (cell in the first column) by going left, and from any topmost empty cell (cell in the first row) by going up.

Let's call a grid **determinable** if, given only which cells are exitable, we can exactly determine which cells are filled in and which aren't.

You are given a grid a of dimensions $n \times m$, i. e. a grid with n rows and m columns. You need to answer q queries ($1 \leq q \leq 2 \cdot 10^5$). Each query gives two integers x_1, x_2 ($1 \leq x_1 \leq x_2 \leq m$) and asks whether the subgrid of a consisting of the columns $x_1, x_1 + 1, \ldots, x_2 - 1, x_2$ is determinable.

Input

The first line contains two integers n, m ($1 \leq n, m \leq 10^6, nm \leq 10^6$) — the dimensions of the grid a.

n lines follow. The y-th line contains m characters, the x-th of which is 'X' if the cell on the intersection of the the y-th row and x-th column is filled and "." if it is empty.

The next line contains a single integer q ($1 \leq q \leq 2 \cdot 10^5$) — the number of queries.

q lines follow. Each line contains two integers x_1 and x_2 ($1 \leq x_1 \leq x_2 \leq m$), representing a query asking whether the subgrid of a containing the columns $x_1, x_1 + 1, \ldots, x_2 - 1, x_2$ is determinable.

Output

For each query, output one line containing "YES" if the subgrid specified by the query is determinable and "NO" otherwise. The output is case insensitive (so "yEs" and "No" will also be accepted).

input

```
4 5
..XXX
...X.
...X.
...X.
5
1 3
3 3
4 5
5 5
1 5
```

output

```
YES
YES
NO
YES
NO
```

For each query of the example, the corresponding subgrid is displayed twice below: first in its input format, then with each cell marked as "E" if it is exitable and "N" otherwise.

For the first query:

```
..X EEN
... EEE
... EEE
... EEE
```

For the second query:

```
X N
. E
. E
. E
```

Note that you can exit the grid by going left from any leftmost cell (or up from any topmost cell); you do not need to reach the top left corner cell to exit the grid.

For the third query:

```
XX NN
X. NN
X. NN
X. NN
```

This subgrid cannot be determined only from whether each cell is exitable, because the below grid produces the above "exitability grid" as well:

```
XX
XX
XX
XX
```

For the fourth query:

```
X N
. E
. E
. E
```

For the fifth query:

```
..XXX EENNN
...X. EEENN
...X. EEENN
...X. EEENN
```

This query is simply the entire grid. It cannot be determined only from whether each cell is exitable because the below grid produces the above "exitability grid" as well:

```
..XXX
...XX
...XX
...XX
```

D. Omkar and the Meaning of Life,

2 seconds, 256 megabytes, standard input, standard output

It turns out that the meaning of life is a permutation $p_1, p_2, \ldots, p_n$ of the integers $1, 2, \ldots, n$ ($2 \le n \le 100$). Omkar, having created all life, knows this permutation, and will allow you to figure it out using some queries.

A query consists of an array $a_1, a_2, \ldots, a_n$ of integers between 1 and n. a is **not** required to be a permutation. Omkar will first compute the pairwise sum of a and p, meaning that he will compute an array s where $s_j = p_j + a_j$ for all $j = 1, 2, \ldots, n$. Then, he will find the smallest index k such that s_k occurs more than once in s, and answer with k. If there is no such index k, then he will answer with 0.

You can perform at most $2n$ queries. Figure out the meaning of life p.

Interaction

Start the interaction by reading single integer n ($2 \le n \le 100$) — the length of the permutation p.

You can then make queries. A query consists of a single line "$?\ \ a_1\ \ a_2\ \ \ldots\ \ a_n$" ($1 \le a_j \le n$).

The answer to each query will be a single integer k as described above ($0 \le k \le n$).

After making 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.

To output your answer, print a single line "$!\ \ p_1\ \ p_2\ \ \ldots\ \ p_n$" then terminate.

You can make at most $2n$ queries. Outputting the answer does not count as a query.

Hack Format

To hack, first output a line containing n ($2 \le n \le 100$), then output another line containing the hidden permutation $p_1, p_2, \ldots, p_n$ of numbers from 1 to n.

input

```
5

2

0

1
```

output

```
? 4 4 2 3 2

? 3 5 1 5 5

? 5 2 4 3 1

! 3 2 1 5 4
```

In the sample, the hidden permutation p is $[3, 2, 1, 5, 4]$. Three queries were made.

The first query is $a = [4, 4, 2, 3, 2]$. This yields $s = [3+4, 2+4, 1+2, 5+3, 4+2] = [7, 6, 3, 8, 6]$. 6 is the only number that appears more than once, and it appears first at index 2, making the answer to the query 2.

The second query is $a = [3, 5, 1, 5, 5]$. This yields $s = [3+3, 2+5, 1+1, 5+5, 4+5] = [6, 7, 2, 10, 9]$. There are no numbers that appear more than once here, so the answer to the query is 0.

The third query is $a = [5, 2, 4, 3, 1]$. This yields $s = [3+5, 2+2, 1+4, 5+3, 4+1] = [8, 4, 5, 8, 5]$. 5 and 8 both occur more than once here. 5 first appears at index 3, while 8 first appears at index 1, and $1 < 3$, making the answer to the query 1.

Note that the sample is only meant to provide an example of how the interaction works; it is not guaranteed that the above queries represent a correct strategy with which to determine the answer.

E. Moment of Bloom,

3 seconds, 512 megabytes, standard input, standard output

She does her utmost to flawlessly carry out a person's last rites and preserve the world's balance of yin and yang.

Hu Tao, being the little prankster she is, has tried to scare you with this graph problem! You are given a connected undirected graph of n nodes with m edges. You also have q queries. Each query consists of two nodes a and b.

Initially, all edges in the graph have a weight of 0. For each query, you must choose a simple path starting from a and ending at b. Then you add 1 to every edge along this path. Determine if it's possible, after processing all q queries, for all edges in this graph to have an even weight. If so, output the choice of paths for each query.

If it is not possible, determine the smallest number of extra queries you could add to make it possible. It can be shown that this number will not exceed 10^{18} under the given constraints.

A simple path is defined as any path that does not visit a node more than once.

An edge is said to have an even weight if its value is divisible by 2.

Input

The first line contains two integers n and m ($2 \le n \le 3 \cdot 10^5$, $n - 1 \le m \le \min\left(\frac{n(n-1)}{2}, 3 \cdot 10^5\right)$).

Each of the next m lines contains two integers x and y ($1 \le x, y \le n$, $x \ne y$) indicating an undirected edge between node x and y. The input will not contain self-loops or duplicate edges, and the provided graph will be connected.

The next line contains a single integer q ($1 \le q \le 3 \cdot 10^5$).

Each of the next q lines contains two integers a and b ($1 \le a, b \le n, a \ne b$), the description of each query.

It is guaranteed that $nq \le 3 \cdot 10^5$.

Output

If it is possible to force all edge weights to be even, print "YES" on the first line, followed by $2q$ lines indicating the choice of path for each query in the same order the queries are given. For each query, the first line should contain a single integer x: the number of nodes in the chosen path. The next line should then contain x spaced separated integers p_i indicating the path you take ($p_1 = a, p_x = b$ and all numbers should fall between 1 and n). This path cannot contain duplicate nodes and must be a valid simple path in the graph.

If it is impossible to force all edge weights to be even, print "NO" on the first line and the minimum number of added queries on the second line.

input

```
6 7
2 1
2 3
3 5
1 4
6 1
5 6
4 5
3
1 4
5 1
4 5
```

output

```
YES
2
1 4
4
5 3 2 1
5
4 1 2 3 5
```

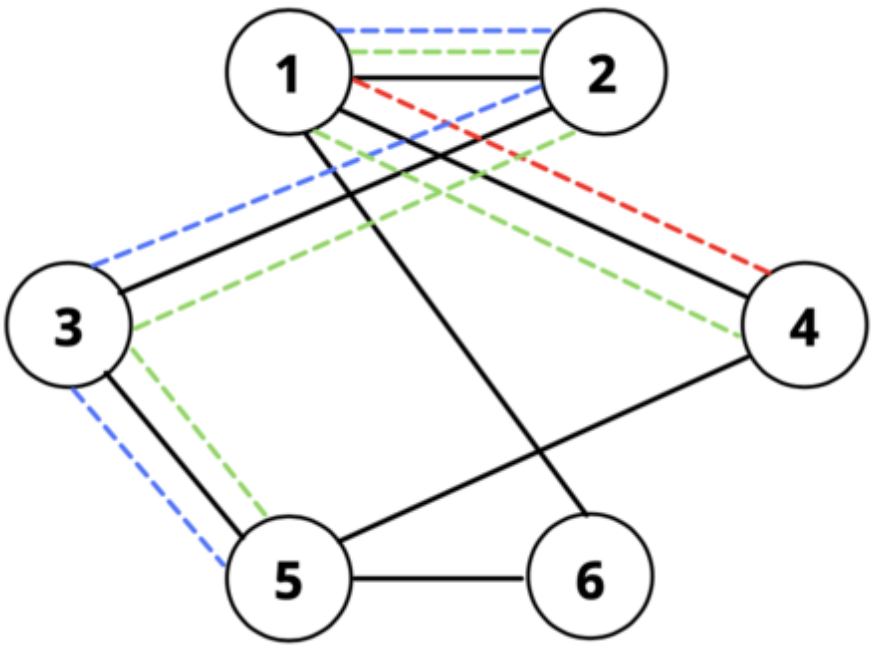
input

```
5 7
4 3
4 5
2 1
1 4
1 3
3 5
3 2
4
4 2
3 5
5 1
4 5
```

output

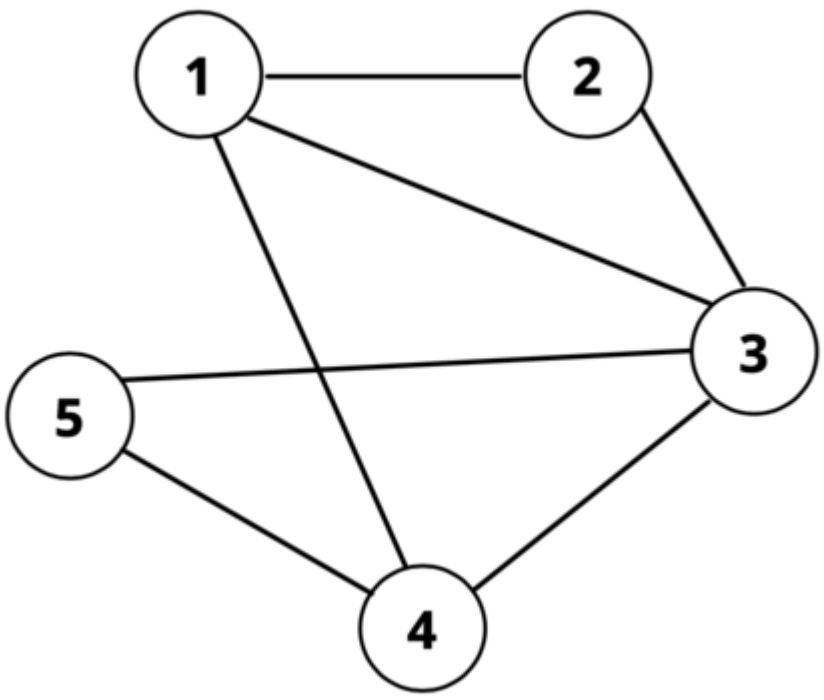
```
NO
2
```

Here is what the queries look like for the first test case (red corresponds to the 1st query, blue 2nd query, and green 3rd query):



Notice that every edge in the graph is part of either 0 or 2 colored query edges.

The graph in the second test case looks like this:



There does not exist an assignment of paths that will force all edges to have even weights with the given queries. One must add at least 2 new queries to obtain a set of queries that can satisfy the condition.

F. Defender of Childhood Dreams,

3 seconds, 512 megabytes, standard input, standard output

Even if you just leave them be, they will fall to pieces all by themselves. So, someone has to protect them, right?

You find yourself playing with Teucer again in the city of Liyue. As you take the eccentric little kid around, you notice something interesting about the structure of the city.

Liyue can be represented as a directed graph containing n nodes. Nodes are labeled from 1 to n. There is a directed edge from node a to node b if and only if $a < b$.

A path between nodes a and b is defined as a sequence of edges such that you can start at a, travel along all of these edges in the corresponding direction, and end at b. The length of a path is defined by the number of edges. A rainbow path of length x is defined as a path in the graph such that there exists at least 2 distinct colors among the set of x edges.

Teucer's favorite number is k. You are curious about the following scenario: If you were to label each edge with a color, what is the minimum number of colors needed to ensure that all paths of length k or longer are rainbow paths?

Teucer wants to surprise his older brother with a map of Liyue. He also wants to know a valid coloring of edges that uses the minimum number of colors. Please help him with this task!

Input

The only line of input contains two integers n and k ($2 \leq k < n \leq 1000$).

Output

On the first line, output c, the minimum colors you need to satisfy the above requirements.

On the second line, print a valid edge coloring as an array of $\frac{n(n-1)}{2}$ integers ranging from 1 to c. Exactly c distinct colors should exist in the construction. Print the edges in increasing order by the start node first, then by the second node.

For example, if $n = 4$, the edge colors will correspond to this order of edges: $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 3)$, $(2, 4)$, $(3, 4)$

input

```
5 3
```

output

```
2
1 2 2 2 2 2 2 1 1 1
```

input

```
5 2
```

output

```
3
3 2 2 1 2 2 1 3 1 1
```

input

```
8 7
```

output

```
2
2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

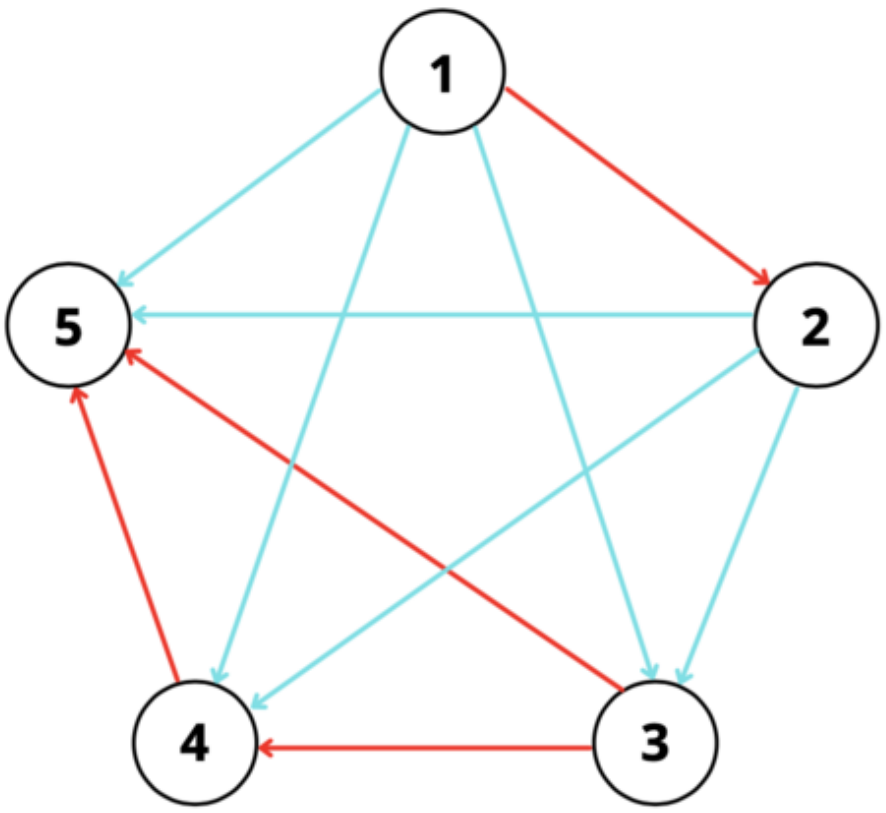
input

```
3 2
```

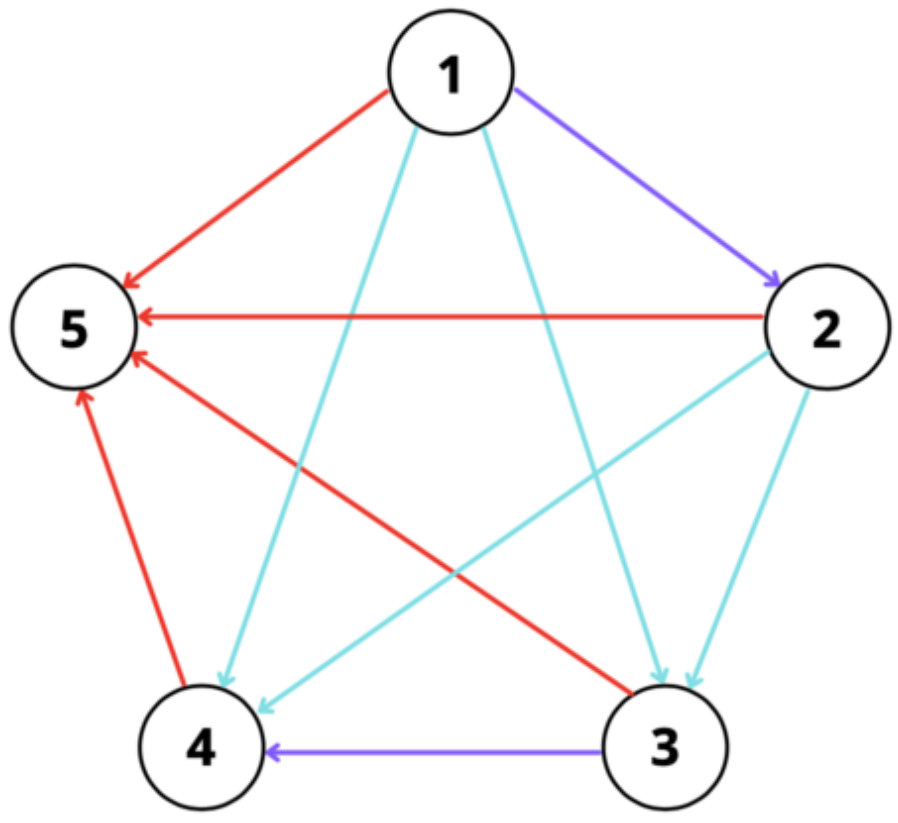
output

```
2
1 2 2
```

The corresponding construction for the first test case looks like this:



It is impossible to satisfy the constraints with less than 2 colors.
The corresponding construction for the second test case looks like this:



One can show there exists no construction using less than 3 colors.

G. Omkar and Time Travel, 2 seconds, 256 megabytes, standard input, standard output

El Psy Kongroo.

Omkar is watching *Steins;Gate*.

In *Steins;Gate*, Okabe Rintarou needs to complete n tasks ($1 \leq n \leq 2 \cdot 10^5$). Unfortunately, he doesn't know when he needs to complete the tasks.

Initially, the time is 0. Time travel will now happen according to the following rules:

- For each $k = 1, 2, \ldots, n$, Okabe will realize at time b_k that he was supposed to complete the k-th task at time a_k ($a_k < b_k$).
- When he realizes this, if k-th task was already completed at time a_k, Okabe keeps the usual flow of time. Otherwise, he time travels to time a_k then immediately completes the task.
- If Okabe time travels to time a_k, all tasks completed after this time will become incomplete again. That is, for every j, if $a_j > a_k$, the j-th task will become incomplete, if it was complete (if it was incomplete, nothing will change).
- Okabe has bad memory, so he can time travel to time a_k **only immediately after** getting to time b_k and learning that he was supposed to complete the k-th task at time a_k. That is, even if Okabe already had to perform k-th task before, he wouldn't remember it before stumbling on the info about this task at time b_k again.

Please refer to the notes for an example of time travelling.

There is a certain set s of tasks such that the first moment that all of the tasks in s are simultaneously completed (regardless of whether any other tasks are currently completed), a funny scene will take place. Omkar loves this scene and wants to know how many times Okabe will time travel before this scene takes place. Find this number modulo $10^9 + 7$. It can be proven that eventually all n tasks will be completed and so the answer always exists.

Input

The first line contains an integer n ($1 \leq n \leq 2 \cdot 10^5$) — the number of tasks that Okabe needs to complete.

n lines follow. The k-th of these lines contain two integers a_k and b_k ($1 \leq a_k < b_k \leq 2n$) — the time at which Okabe needs to complete the k-th task and the time that he realizes this respectively. All $2n$ of these times are distinct (so every time from 1 to $2n$ inclusive appears exactly once in the input).

The next line contains a single integer t ($1 \leq t \leq n$) — the size of the set s of tasks that lead to the funny scene.

The last line contains t integers $s_1, s_2, \ldots, s_t$ — ($1 \leq s_k \leq n$, the numbers $s_1, s_2, \ldots, s_t$ are distinct) — the set s of tasks.

Output

Output a single integer — the number of times that Okabe time travels until all tasks in the set s are simultaneously completed, modulo $10^9 + 7$.

input

```
2
1 4
2 3
2
1 2
```

output

```
3
```

input

```
2
1 4
2 3
1
1
```

output

```
2
```

input

```
1
1 2
1
1
```

output

```
1
```

input

```
6
10 12
3 7
4 6
2 9
5 8
1 11
3
2 4 6
```

output

```
17
```

input

```
16
31 32
3 26
17 19
4 24
1 28
15 21
12 16
18 29
20 23
7 8
11 14
9 22
6 30
5 10
25 27
2 13
6
3 8 2 5 12 11
```

output

```
138
```

For the first sample, all tasks need to be completed in order for the funny scene to occur.

Initially, the time is 0. Nothing happens until time 3, when Okabe realizes that he should have done the 2-nd task at time 2. He then time travels to time 2 and completes the task.

As the task is done now, he does not time travel again when the time is again 3. However, at time 4, he travels to time 1 to complete the 1-st task.

This undoes the 2-nd task. This means that the 2-nd task is not currently completed, meaning that the funny scene will **not** occur at this point even though the 1-st task is currently completed and Okabe had previously completed the 2-nd task.

Once it is again time 3 he travels back to time 2 once more and does the 2-nd task again.

Now all tasks are complete, with Okabe having time travelled 3 times.

The second sample has the same tasks for Okabe to complete. However, this time the funny scene only needs the first task to be completed in order to occur. From reading the above sample you can see that this occurs once Okabe has time travelled 2 times.

H. Omkar and Tours, 3 seconds, 512 megabytes, standard input, standard output

Omkar is hosting tours of his country, Omkarland! There are n cities in Omkarland, and, rather curiously, there are exactly $n-1$ bidirectional roads connecting the cities to each other. It is guaranteed that you can reach any city from any other city through the road network.

Every city has an enjoyment value e. Each road has a capacity c, denoting the maximum number of vehicles that can be on it, and an associated toll t. However, the toll system in Omkarland has an interesting quirk: if a vehicle travels on multiple roads on a single journey, they pay only the highest toll of any single road on which they traveled. (In other words, they pay $\max t$ over all the roads on which they traveled.) If a vehicle traverses no roads, they pay 0 toll.

Omkar has decided to host q tour groups. Each tour group consists of v vehicles starting at city x. (Keep in mind that a tour group with v vehicles can travel only on roads with capacity $\geq v$.) Being the tour organizer, Omkar wants his groups to have as much fun as they possibly can, but also must reimburse his groups for the tolls that they have to pay. Thus, for each tour group, Omkar wants to know two things: first, what is the enjoyment value of the city y with maximum enjoyment value that the tour group can reach from their starting city, and second, how much per vehicle will Omkar have to pay to reimburse the entire group for their trip from x to y? (This trip from x to y will always be on the shortest path from x to y.)

In the case that there are multiple reachable cities with the maximum enjoyment value, Omkar will let his tour group choose which one they want to go to. Therefore, to prepare for all possible scenarios, he wants to know the amount of money per vehicle that he needs to guarantee that he can reimburse the group regardless of which city they choose.

Input

The first line contains two integers n and q ($2 \leq n \leq 2 \cdot 10^5$, $1 \leq q \leq 2 \cdot 10^5$), representing the number of cities and the number of groups, respectively.

The next line contains n integers $e_1, e_2, \ldots, e_n$ ($1 \leq e_i \leq 10^9$), where e_i represents the enjoyment value for city i.

The next $n-1$ lines each contain four integers a, b, c, and t ($1 \leq a, b \leq n, 1 \leq c \leq 10^9, 1 \leq t \leq 10^9$), representing an road between city a and city b with capacity c and toll t.

The next q lines each contain two integers v and x ($1 \leq v \leq 10^9$, $1 \leq x \leq n$), representing the number of vehicles in the tour group and the starting city, respectively.

Output

Output q lines. The i-th line should contain two integers: the highest possible enjoyment value of a city reachable by the i-th tour group, and the amount of money per vehicle Omkar needs to guarantee that he can reimburse the i-th tour group.

input

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

output

```
3 8
3 0
3 2
```

input

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

output

```
1 0
2 1
3 1
4 1
5 1
```

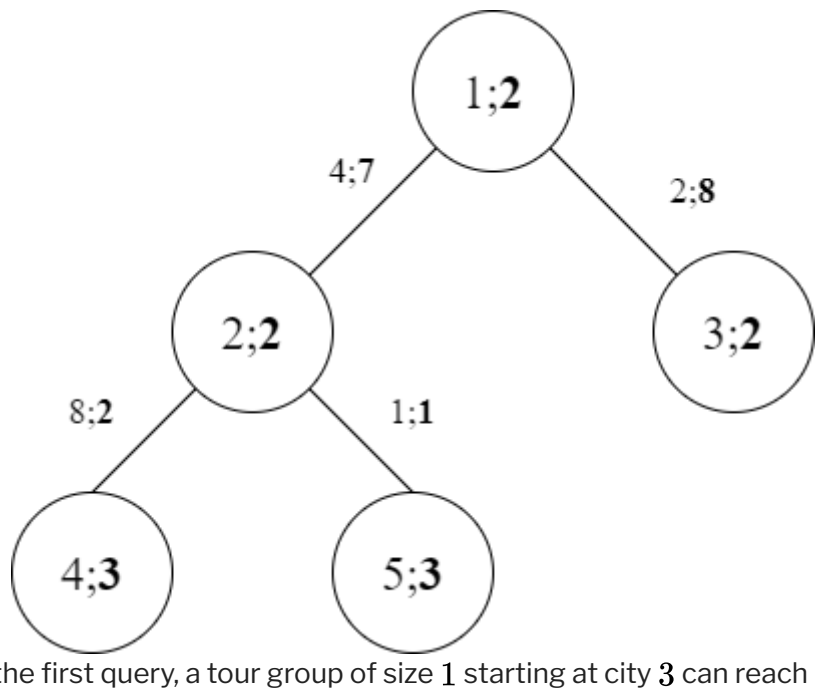
input

```
5 5
1 2 2 2 2
1 2 5 8
1 3 6 3
1 4 4 5
1 5 7 1
4 1
5 1
6 1
7 1
8 1
```

output

```
2 8
2 8
2 3
2 1
1 0
```

A map of the first sample is shown below. For the nodes, unbolded numbers represent indices and bolded numbers represent enjoyment values. For the edges, unbolded numbers represent capacities and bolded numbers represent tolls.

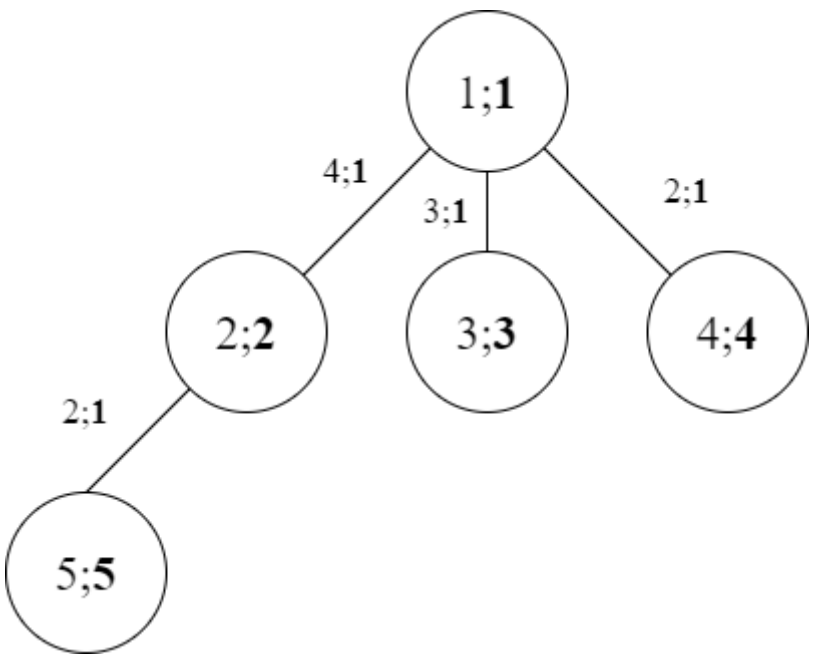


For the first query, a tour group of size 1 starting at city 3 can reach cities 1, 2, 3, 4, and 5. Thus, the largest enjoyment value that they can reach is 3. If the tour group chooses to go to city 4, Omkar will have to pay 8 per vehicle, which is the maximum.

For the second query, a tour group of size 9 starting at city 5 can reach only city 5. Thus, the largest reachable enjoyment value is still 3, and Omkar will pay 0 per vehicle.

For the third query, a tour group of size 6 starting at city 2 can reach cities 2 and 4. The largest reachable enjoyment value is again 3. If the tour group chooses to go to city 4, Omkar will have to pay 2 per vehicle, which is the maximum.

A map of the second sample is shown below:



For the first query, a tour group of size 5 starting at city 1 can only reach city 1. Thus, their maximum enjoyment value is 1 and the cost Omkar will have to pay is 0 per vehicle.

For the second query, a tour group of size 4 starting at city 1 can reach cities 1 and 2. Thus, their maximum enjoyment value is 2 and Omkar will pay 1 per vehicle.

For the third query, a tour group of size 3 starting at city 1 can reach cities 1, 2, and 3. Thus, their maximum enjoyment value is 3 and Omkar will pay 1 per vehicle.

For the fourth query, a tour group of size 2 starting at city 1 can reach cities 1, 2, 3 and 4. Thus, their maximum enjoyment value is 4 and Omkar will pay 1 per vehicle.

For the fifth query, a tour group of size 1 starting at city 1 can reach cities 1, 2, 3, 4, and 5. Thus, their maximum enjoyment value is 5 and Omkar will pay 1 per vehicle.

I. Omkar and Mosaic,

2 seconds, 256 megabytes, standard input, standard output

Omkar is creating a mosaic using colored square tiles, which he places in an $n \times n$ grid. When the mosaic is complete, each cell in the grid will have either a glaucous or sinoper tile. However, currently he has only placed tiles in some cells.

A completed mosaic will be a **mastapeece** if and only if each tile is adjacent to exactly 2 tiles of the same color (2 tiles are adjacent if they share a side.) Omkar wants to fill the rest of the tiles so that the mosaic becomes a **mastapeece**. Now he is wondering, is the way to do this unique, and if it is, what is it?

Input

The first line contains a single integer n ($1 \leq n \leq 2000$).

Then follow n lines with n characters in each line. The i-th character in the j-th line corresponds to the cell in row i and column j of the grid, and will be S if Omkar has placed a sinoper tile in this cell, G if Omkar has placed a glaucous tile, **.** if it's empty.

Output

On the first line, print UNIQUE if there is a unique way to get a mastapeece, NONE if Omkar cannot create any, and MULTIPLE if there is more than one way to do so. **All letters must be uppercase.**

If you print UNIQUE, then print n additional lines with n characters in each line, such that the i-th character in the j^{th} line is S if the tile in row i and column j of the mastapeece is sinoper, and G if it is glaucous.

input

```
4
S...
..G.
....
...S
```

output

```
MULTIPLE
```

input

```
6
S.....
....G.
..S...
.....S
....G.
G.....
```

output

```
NONE
```

input

```
10
.S....S...
..........
...SSS....
..........
..........
...GS.....
....G...G.
..........
......G...
..........
```

output
UNIQUE SSSSSSSSS SGGGGGGGS SGSSSSSGS SGSGGGGSGS SGSGSSGSGS SGSGSSGSGS SGSGGGGSGS SGSSSSSGS SGGGGGGGS SSSSSSSSS

input
1 .

output
NONE

For the first test case, Omkar can make the mastapeeces

SSSS

SGGS

SGGS

SSSS

and

SSGG

SSGG

GGSS

GGSS.

For the second test case, it can be proven that it is impossible for Omkar to add tiles to create a mastapeece.

For the third case, it can be proven that the given mastapeece is the only mastapeece Omkar can create by adding tiles.

For the fourth test case, it's clearly impossible for the only tile in any mosaic Omkar creates to be adjacent to two tiles of the same color, as it will be adjacent to 0 tiles total.