AIM Tech Round 4 (Div. 1)

A. Sorting by Subsequences

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a sequence $a_1, a_2, \ldots, a_n$ consisting of **different** integers. It is required to split this sequence into the **maximum** number of subsequences such that after sorting integers in each of them in increasing order, the total sequence also will be sorted in increasing order.

Sorting integers in a subsequence is a process such that the numbers included in a subsequence are ordered in increasing order, and the numbers which are not included in a subsequence don't change their places.

Every element of the sequence must appear in exactly one subsequence.

Input

The first line of input data contains integer n ($1 \le n \le 10^5$) — the length of the sequence.

The second line of input data contains n different integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — the elements of the sequence. It is guaranteed that all elements of the sequence are distinct.

Output

In the first line print the maximum number of subsequences k, which the original sequence can be split into while fulfilling the requirements.

In the next k lines print the description of subsequences in the following format: the number of elements in subsequence c_i ($0 < c_i \le n$), then c_i integers $l_1, l_2, \ldots, l_{c_i}$ ($1 \le l_j \le n$) — indices of these elements in the original sequence.

Indices could be printed in any order. Every index from 1 to n must appear in output **exactly once**.

If there are several possible answers, print any of them.

Examples

input
6 3 2 1 6 5 4

output
4 2 1 3 1 2 2 4 6 1 5

input
6 83 -75 -49 11 37 62

output
1 6 1 2 3 4 5 6

Note

In the first sample output:

After sorting the first subsequence we will get sequence $1\ 2\ 3\ 6\ 5\ 4$.

Sorting the second subsequence changes nothing.

After sorting the third subsequence we will get sequence $1\ 2\ 3\ 4\ 5\ 6$.

Sorting the last subsequence changes nothing.

B. Interactive LowerBound

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

You are given a **sorted** in increasing order singly linked list. You should find the minimum integer in the list which is greater than or equal to x.

More formally, there is a singly liked list built on an array of n elements. Element with index i contains two integers: $value_i$ is the integer value in this element, and $next_i$ that is the index of the next element of the singly linked list (or `-1`, if the current element is the last). The list is sorted, i.e. if $next_i \neq -1$, then $value_{next_i} > value_i$.

You are given the number of elements in the list n, the index of the first element $start$, and the integer x.

You can make up to 2000 queries of the following two types:

- `? i` ($1 \le i \le n$) — ask the values $value_i$ and $next_i$,
- `! ans` — give the answer for the problem: the minimum integer, greater than or equal to x, or `! -1`, if there are no such integers. Your program should terminate after this query.

Write a program that solves this problem.

Input

The first line contains three integers n, $start$, x ($1 \le n \le 50000$, $1 \le start \le n$, $0 \le x \le 10^9$) — the number of elements in the list, the index of the first element and the integer x.

Output

To print the answer for the problem, print `! ans`, where `ans` is the minimum integer in the list greater than or equal to x, or `-1`, if there is no such integer.

Interaction

To make a query of the first type, print `? i` ($1 \le i \le n$), where `i` is the index of element you want to know information about.

After each query of type `?` read two integers $value_i$ and $next_i$ ($0 \le value_i \le 10^9$, $-1 \le next_i \le n$, $next_i \neq 0$).

It is guaranteed that if $next_i \neq -1$, then $value_{next_i} > value_i$, and that the array values give a valid singly linked list with $start$ being the first element.

Note that you can't ask more than 1999 queries of the type `?`.

If $next_i = -1$ and $value_i = -1$, then it means that you asked more queries than allowed, or asked an invalid query. Your program should immediately terminate (for example, by calling `exit(0)`). You will receive "`Wrong Answer`", it means that you asked more queries than allowed, or asked an invalid query. If you ignore this, you can get other verdicts since your program will continue to read from a closed stream.

Your solution will get "Idleness Limit Exceeded", if you don't print anything or forget to `flush` the output, including the final answer.

To `flush` you can use (just after printing a query and line end):

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

Hacks format

For hacks, use the following format:

In the first line print three integers n, $start$, x ($1 \le n \le 50000$, $1 \le start \le n$, $0 \le x \le 10^9$).

In the next n lines print the description of the elements of the list: in the i-th line print two integers $value_i$ and $next_i$ ($0 \le value_i \le 10^9$, $-1 \le next_i \le n$, $next_i \neq 0$).

The printed structure should be a valid singly linked list. In particular, it should be possible to reach all elements from $start$ by following links $next_i$, and the last element end should have `-1` in the $next_{end}$.

Example

input
5 3 80 97 -1 58 5 16 2 81 1

```
79 4
```

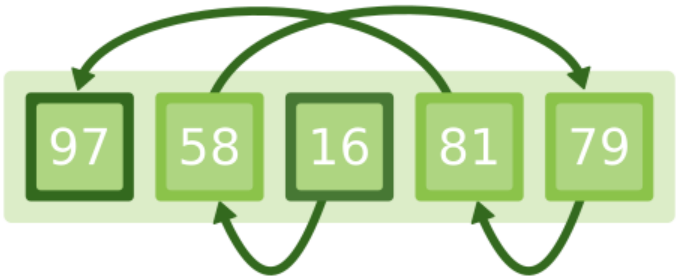
output

```
? 1
? 2
? 3
? 4
? 5
! 81
```

Note

You can read more about singly linked list by the following link: https://en.wikipedia.org/wiki/Linked_list#Singly_linked_list

The illustration for the first sample case. Start and finish elements are marked dark.



C. Upgrading Tree

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a tree with n vertices and you are allowed to perform **no more than** $2n$ transformations on it. Transformation is defined by three vertices x, y, y' and consists of deleting edge (x, y) and adding edge (x, y'). Transformation x, y, y' could be performed if all the following conditions are satisfied:

1. There is an edge (x, y) in the current tree.
2. After the transformation the graph remains a tree.
3. After the deletion of edge (x, y) the tree would consist of two connected components. Let's denote the set of nodes in the component containing vertex x by V_x, and the set of nodes in the component containing vertex y by V_y. Then condition $|V_x| > |V_y|$ should be satisfied, i.e. the size of the component with x should be strictly larger than the size of the component with y.

You should **minimize** the sum of squared distances between all pairs of vertices in a tree, which you could get after no more than $2n$ transformations and output any sequence of transformations leading initial tree to such state.

Note that you don't need to minimize the number of operations. It is necessary to minimize only the sum of the squared distances.

Input

The first line of input contains integer n ($1 \le n \le 2 \cdot 10^5$) — number of vertices in tree.

The next $n - 1$ lines of input contains integers a and b ($1 \le a, b \le n, a \ne b$) — the descriptions of edges. It is guaranteed that the given edges form a tree.

Output

In the first line output integer k ($0 \le k \le 2n$) — the number of transformations from your example, **minimizing** sum of squared distances between all pairs of vertices.

In each of the next k lines output three integers x, y, y' — indices of vertices from the corresponding transformation.

Transformations with $y = y'$ are allowed (even though they don't change tree) if transformation conditions are satisfied.

If there are several possible answers, print any of them.

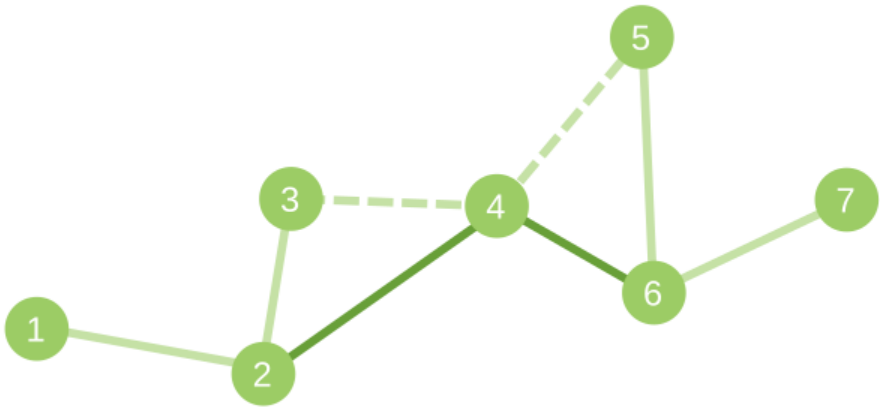
Examples

input
3 3 2 1 3
output
0

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

Note

This is a picture for the second sample. Added edges are dark, deleted edges are dotted.


5
3
4
7
6
1
2

D. Dynamic Shortest Path

time limit per test: 10 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a weighted directed graph, consisting of n vertices and m edges. You should answer q queries of two types:

- `1 v` — find the length of shortest path from vertex 1 to vertex v.
- `2 c` l_1 l_2 ... l_c — add 1 to weights of edges with indices l_1, l_2, ..., l_c.

Input

The first line of input data contains integers n, m, q ($1 \le n, m \le 10^5$, $1 \le q \le 2000$) — the number of vertices and edges in the graph, and the number of requests correspondingly.

Next m lines of input data contain the descriptions of edges: i-th of them contains description of edge with index i — three integers a_i, b_i, c_i ($1 \le a_i, b_i \le n$, $0 \le c_i \le 10^9$) — the beginning and the end of edge, and its initial weight correspondingly.

Next q lines of input data contain the description of edges in the format described above ($1 \le v \le n$, $1 \le l_j \le m$). It's guaranteed that inside single query all l_j are distinct. Also, it's guaranteed that a total number of edges in all requests of the second type does not exceed 10^6.

Output

For each query of first type print the length of the shortest path from 1 to v in a separate line. Print `-1`, if such path does not exists.

Examples

input

```
3 2 9
1 2 0
2 3 0
2 1 2
1 3
1 2
2 1 1
1 3
1 2
2 2 1 2
1 3
1 2
```

output

```
1
0
2
1
4
2
```

input

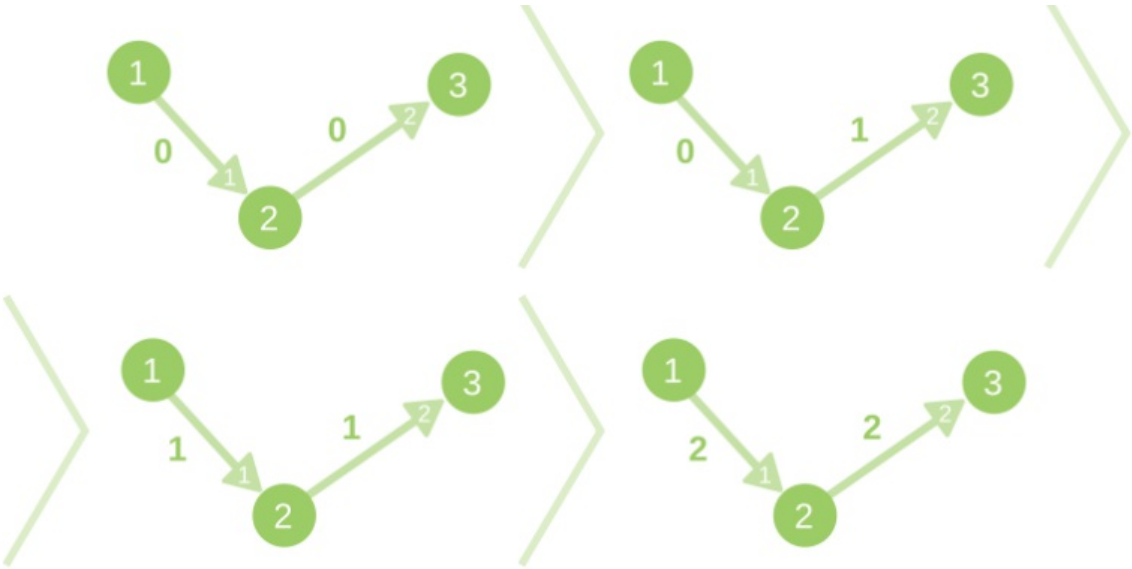
```
5 4 9
2 3 1
2 4 1
3 4 1
1 2 0
1 5
1 4
2 1 2
2 1 2
1 4
2 2 1 3
1 4
2 1 4
1 4
```

output

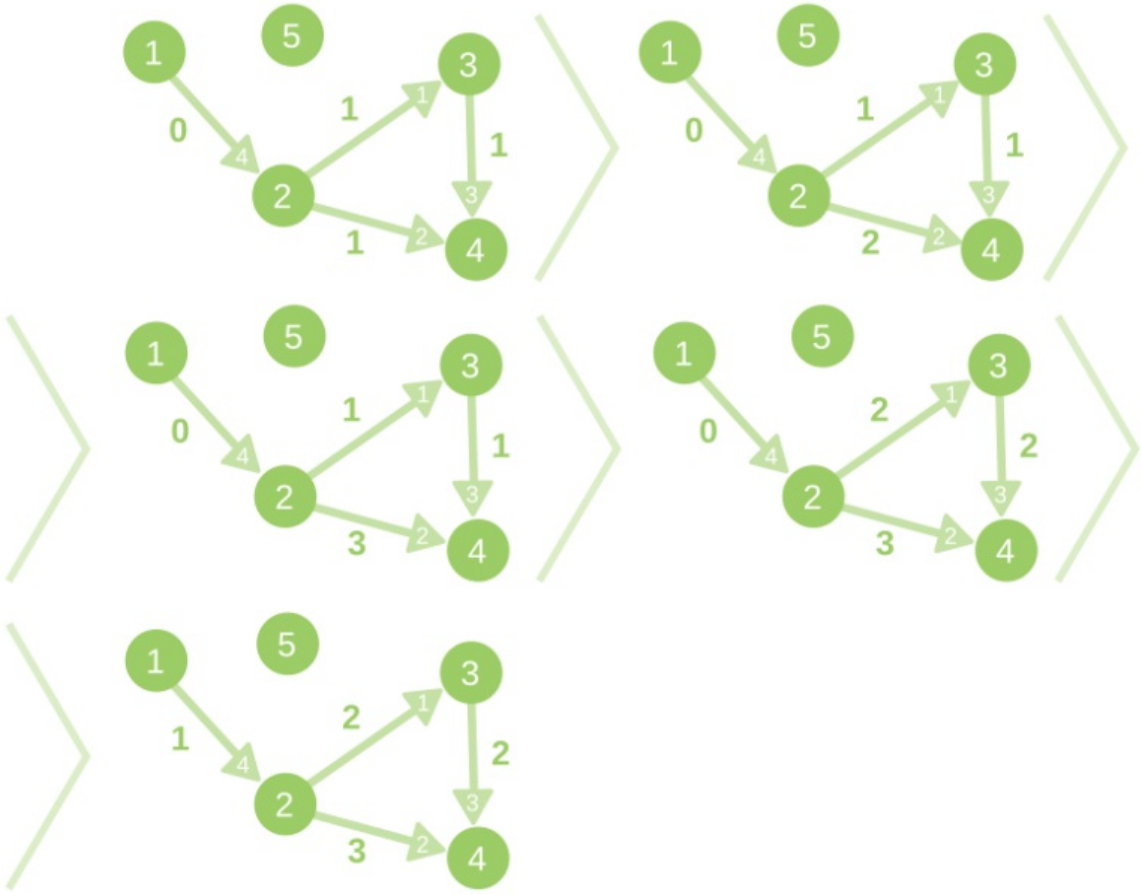
```
-1
1
2
3
4
```

Note

The description of changes of the graph in the first sample case:



The description of changes of the graph in the second sample case:



E. Maximum Flow

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a directed graph, consisting of n vertices and m edges. The vertices s and t are marked as source and sink correspondingly. Additionally, there are no edges ending at s and there are no edges beginning in t.

The graph was constructed in a following way: initially each edge had capacity $c_i > 0$. A maximum flow with source at s and sink at t was constructed in this flow network. Let's denote f_i as the value of flow passing through edge with index i. Next, all capacities c_i and flow value f_i were erased. Instead, indicators g_i were written on edges — if flow value passing through edge i was positive, i.e. 1 if $f_i > 0$ and 0 otherwise.

Using the graph and values g_i, find out what is the **minimum** possible number of edges in the initial flow network that could be saturated (the passing flow is equal to capacity, i.e. $f_i = c_i$). Also construct the corresponding flow network with maximum flow in it.

A flow in directed graph is described by flow values f_i on each of the edges so that the following conditions are satisfied:

- for each vertex, except source and sink, total incoming flow and total outcoming flow are equal,
- for each edge $0 \le f_i \le c_i$

A flow is maximum if the difference between the sum of flow values on edges from the source, and the sum of flow values on edges to the source (there are no such in this problem), is maximum possible.

Input

The first line of input data contains four positive integers n, m, s, t ($2 \le n \le 100$, $1 \le m \le 1000$, $1 \le s, t \le n$, $s \ne t$) — the number of vertices, the number of edges, index of source vertex and index of sink vertex correspondingly.

Each of next m lines of input data contain non-negative integers u_i, v_i, g_i ($1 \le u_i, v_i \le n$, $g_i \in \{0, 1\}$) — the beginning of edge i, the end of edge i and indicator, which equals to 1 if flow value passing through edge i was positive and 0 if not.

It's guaranteed that no edge connects vertex with itself. Also it's guaranteed that there are no more than one edge between each ordered pair of vertices and that there exists at least one network flow that satisfies all the constrains from input data.

Output

In the first line print single non-negative integer k — minimum number of edges, which should be saturated in maximum flow.

In each of next m lines print two integers f_i, c_i ($1 \le c_i \le 10^9$, $0 \le f_i \le c_i$) — the flow value passing through edge i and capacity of edge i.

This data should form a correct maximum flow in flow network. Also there must be exactly k edges with statement $f_i = c_i$ satisfied. Also statement $f_i > 0$ must be true if and only if $g_i = 1$.

If there are several possible answers, print any of them.

Example

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

Note

The illustration for second sample case. The saturated edges are marked dark, while edges with $g_i = 0$ are marked with dotted line. The integer on edge is the index of this edge in input list.