Day 4. DFS and Similar Problems

A. Depth-First Search

1 second, 256 megabytes

You are given an undirected graph consisting of n vertices.

You have to find the path from the vertex a to the vertex b using depth-first search. Use an implementation that iterates over adjacent vertices of current vertex in ascending order of indices. Only paths that are produced by this version of the algorithm will be accepted.

Input

The first line contains three integer numbers n, a, b ($1 \le n \le 500$, $1 \le a, b \le n$) — number of vertices and pair of vertices between which you have to find the path.

The following n lines contain square binary matrix g — adjacency matrix of the given graph. Each row consists of n space-separated integer numbers 0 or 1. Element $g_{i,j}$ equals to 1, if there is an edge between vertices i and j, or 0 otherwise.

It is guaranteed that there are no loops (i.e. $g_{i,i} = 0$ for each i) and that adjacency matrix is symmetric (i.e. $g_{i,j} = g_{j,i}$ for any pair i and j).

Output

If there is no path from a to b, print only one integer "-1" (without quotes).

Otherwise in the first line of the output print path length (number of edges in the path). In the second line print the path itself, beginning with the vertex a and ending with the vertex b.

input

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

output

```
3
1 2 3 4
```

input

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

output

```
-1
```

input

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

output

```
3
4 3 1 2
```

input

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

output

```
0
2
```

B. Connected Components

4 seconds, 256 megabytes

You are given an undirected graph consisting of n vertices and m edges. For each vertex u you have to calculate the size of the connected component containing the vertex u.

Input

The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n, u_i \ne v_i$). There is no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there no other pairs (v_i, u_i) and (u_i, v_i) in the list of edges.

Output

Print n integer numbers: number i must be equal to the size of the connected component containing the vertex i.

input

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

output

```
2 2 2 2 1
```

input

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

output

```
3 3 3
```

input

```
3 0
```

output

```
1 1 1
```

input

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

output

```
3 3 3 2 2
```

input
6 3 1 2 2 3 5 6
output
3 3 3 1 2 2

C. Tree?

1 second, 256 megabytes

Tree is a *connected* acyclic graph. You are given an undirected unweighted graph. You are to check if it is a tree or not.

Input
The first line contains a single integer N ($1 \le N \le 100$) — the number of vertices in the graph.

The next N lines contain N integers each — the adjacency matrix of the graph: the j-th integer of th i-th line is 1, if vertices i and j are connected with an edge, and 0, if there is no edge between them. The main diagonal contains zeros only. The matrix is symmetrical in respect to the main diagonal.

Output
Print "YES" if the graph is a tree, and "NO" otherwise.

input
6 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
output
NO

input
3 0 1 0 1 0 1 0 1 0
output
YES

D. Tree Diameter

1 second, 256 megabytes

You are given a tree. You need to find its diameter.

A diameter of a graph is the largest integer d such that there is a pair of vertices such that the distance between them is equal to d. Distance between a pair of vertices is the number of edges in the shortest path between them.

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

The second line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i < i$), where p_i means there is an edge between vertices i and p_i. It is guaranteed that this line describes a valid tree.

Output
Output a single integer — the diameter of the graph.

input
5 1 1 2 2
output
3

input
6 1 2 3 4 5
output
5

E. Cyclicity Checking

5 seconds, 256 megabytes

You are given a directed graph consisting of n vertices and m edges. You have to check that the given graph is acyclic, i.e. this graph contains no path from a vertex to itself with positive number of edges.

Input
The first line contains only one integer t ($1 \le t \le 20$) — the number of test cases.

The first line of the test case contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines of the test case contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n$, $u_i \ne v_i$). There are no multiple edges in same direction in the given graph, i.e. for each pair (v_i, u_i) there are no other pairs (v_i, u_i) in the list of edges.

Output
For each test case print the answer in the following format: if given graph is acyclic, print "YES" in only one line (without quotes), else print "NO" on the first line, then print length of the **any simple** cycle on the second line, then on the next line print this simple cycle itself.

input
2 4 4 1 2 2 3 2 4 3 4 4 5 1 2 1 3 3 4 2 4 4 1
output
YES NO 3 1 2 4 1

F. Bipartite Checking

3 seconds, 256 megabytes

You are given an undirected graph consisting of n vertices and m edges. You have to check that the given graph is bipartite, i.e. that it is possible to divide its vertices into two sets in such a way, that in each set there is no pair of adjacent vertices.

Input
The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains the edges: edge i is given as a pair of the vertices v_i, u_i ($1 \le v_i, u_i \le n$, $u_i \ne v_i$). There is no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there no other pairs (v_i, u_i) and (u_i, v_i) in the list of edges.

Output
If the given graph is bipartite, print "YES" (without quotes), otherwise print "NO".

input
4 4 1 2 2 3 3 4 4 1
output
YES

input
4 5 1 2 2 3 3 4 4 1 1 3
output
NO

G. Labyrinth Analysis

2 seconds, 256 megabytes

There is a labyrinth on a matrix $n \times m$. Each cell is either empty (denoted by a dot), or contains an obstacle (denoted by an asterisk).

A robot can only move through empty cells, and can move only to any of four cells that share an edge with current one.

Print the total number of connected components of empty cells and their sizes, in non-decreasing order.

Input
The first line contains two integers n and m ($1 \le n, m \le 500$) — the size of the labyrinth.

n lines follow, each containing m characters, describing the labyrinth.

Output
Print the number of components in the first line. In the second line print components' sizes, in non-decreasing order.

input
7 10 ..**...... ....*..... ******.... ......**** .....*..** .....***** ......**..
output
5 2 2 6 15 22

input
2 2 ** **
output
0

H. Topological Sort

1 second, 256 megabytes

You are given a directed graph of n nodes and m vertices. A topological sort of a directed graph is a permutation of vertex indices such that each edge of the graph goes from a vertex that appears strictly earlier on the list to a vertex that appears later.

Check if there is a topological sort of the given graph. If there is, output any valid topological sort.

Input
The first line contains two integers n and m ($1 \le n, m \le 10^5$) — the number of vertices and the number of edges.

The next m lines contain edges, one per line. Each line contains two integers a and b ($0 \le a, b < n$) — the startpoint and the endpoint of an edge. Loops and multiple edges are allowed.

Output
If there is no topological sort for the given graph, output "NO" (without quotes).

Otherwise, output "YES" in the first line. In the second line, output a permutation of indices of the graph (integers from 0 to $n - 1$), such that for each edge the startpoint appears in the permutation strictly earlier than the endpoint.

input
4 5 0 2 2 1 1 3 2 3 0 3
output
YES 0 2 1 3

input
3 3 0 1 1 2 2 0

output

```
NO
```

input

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

output

```
NO
```

I. Carrots Eating

5 seconds, 256 megabytes

You are given a forest which is represented as a rectangular grid of size $n \times m$. The forest containing the empty cells (which are represented as `'.'`), trees (`'#'`), rabbits (`'R'`) and carrots (`'C'`).

Your problem is to calculate for each rabbit the maximum number of carrots that he can eat if he can move to adjacent cells (adjacent cell is a cell sharing a side) and can't move to a cell containing a tree. **The answer for each rabbit must be calculated independently**.

Input

The first line of the input contains two integer numbers n and m ($1 \le n, m \le 500$) — the size of the forest.

The next n lines contain m characters each. The character $c_{i,j}$ is equal to `'.'` if the cell at position (i, j) is empty, `'#'` if the cell at the position (i, j) is a tree, `'R'` if the cell at the position (i, j) is a rabbit and `'C'` if the cell at the position (i, j) is a carrot.

Output

Print k numbers (k is equals to the number of rabbits). For the i-th rabbit print the maximum number of carrots that he can eat. The rabbits are numbered in the order of input (the rabbit with position (x_1, y_1) has lesser number than the rabbit with position (x_2, y_2) if $x_1 < x_2$ or $x_1 = x_2$ and $y_1 < y_2$).

input

```
4 4
.RCR
C###
#RRC
R##R
```

output

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

input

```
4 8
########
#.R#..R#
#C.#...#
########
```

output

```
1 0
```

input

```
3 3
RRR
##.
CCC
```

output

```
3 3 3
```

input

```
3 3
RRR
###
CCC
```

output

```
0 0 0
```

J. Edges Replacing

5 seconds, 256 megabytes

You are given an undirected graph containing n vertices and m edges. It is guaranteed that graph doesn't contains loops and multiple edges.

You have to replace some minimal number of edges to transform this graph to the tree. Replacing the edge means adding it to the graph if there is no such edge and erasing it from the graph otherwise.

Input

The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n, u_i \ne v_i$). There is no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there no other pairs (v_i, u_i) and (u_i, v_i) in the list of edges.

Output

In the first line of the output print k — the minimal number of edges you need to replace to transform the given graph to the tree. In the next k lines print the edges itself.

input

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

output

```
3
1 3
4 6
1 4
```

input

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

output

```
2
2 3
1 2
```

input

```
2 1
2 1
```

output
0

K. Bridges

5 seconds, 256 megabytes

You are given an undirected graph consisting of n vertices and m edges. You have to find all edges, which are bridges. I.e. the edges, removing which leads to increasing the number of connected components.

Input

The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n$, $u_i \ne v_i$). There is no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there no other pairs (v_i, u_i) and (u_i, v_i) in the list of edges.

Output

In the first line of output print the number of bridges. In the second line, print indices of edges which are bridges in **any** order.

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

input
4 4 1 2 2 3 3 4 4 1
output
0

input
5 4 1 2 1 3 3 4 4 5
output
4 1 4 3 2

L. Cut Vertices

5 seconds, 256 megabytes

You are given an undirected graph consisting of n vertices and m edges. You have to find cut vertices (articulation points) of the given graph. A cut vertex is any vertex whose removal increases the number of connected components.

Input

The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n$, $u_i \ne v_i$). There is no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there no other pairs (v_i, u_i) and (u_i, v_i) in the list of edges.

Output

In the first line of the output print the number of the cut-points. In the second line print indices of vertices which is a cut-points in **any** order.

input
5 4 1 2 1 3 3 4 3 5
output
2 1 3

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

M. Tree Queries

10 seconds, 512 megabytes

You are given an rooted tree with root vertex 1.

You have to answer m queries (a_i, b_i), and you have to find relative position of vertices a_i and b_i, i. e. state one of the following:

1. a_i is an ancestor of b_i;
2. b_i is an ancestor of a_i;
3. vertices are non-comparable.

Input

The first line of input contains one integer number n ($1 \le n \le 3 \cdot 10^5$) — the number of vertices.

The second line contains $n - 1$ integer numbers $p_2, p_3, \ldots, p_n$ ($1 \le p_i \le n$), where p_i is a parent of the vertex i.

The third line contains one integer number m ($0 \le m \le 3 \cdot 10^5$) — the number of queries.

Next m lines contains queries in the format a_i b_i ($1 \le a_i, b_i \le n$, $a_i \ne b_i$).

Output

Print m numbers, one in one line — the answers for the queries. The i-th number must be equal to 1, if a_i is an ancestor of b_i, must be equal to 2, if b_i is an ancestor of a_i, and 3 otherwise.

input

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

output

```
2
1
3
1
```

input

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

output

```
2
1
3
1
```

N. Biconnected Components

8 seconds, 256 megabytes

A connected undirected graph that does not contain cut-points is called biconnected. A maximal subgraph that does not contain cut-points (here we mean cut points of this subgraph if it is considered as a separate graph) is called a biconnected component. Obviously, a vertex, in general, could belong to several biconnected components, also, each edge belongs to exactly one biconnected component. Moreover, each biconnected component is strictly defined by the set of edges that belongs to it.

You are given an undirected graph consisting of n vertices and m edges. Print all biconnected components as lists of edges.

Input

The first line contains two integers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — the number of vertices and the number of edges.

The following m lines contains edges: edge i is given as a pair of vertices v_i, u_i ($1 \le v_i, u_i \le n, u_i \ne v_i$). There are no multiple edges in the given graph, i.e. for each pair (v_i, u_i) there is no other pair (v_i, u_i) nor (u_i, v_i) in the list of edges.

Output

In the first line print w — the number of biconnected components.

In the following lines print the components. For each component, print the number of edges first, then print the edges. Both the components and the edges inside a component may be printed in any order. Ends of the edge also may be printed in any order.

input

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

output

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

input

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

output

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

input

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

output

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

O. Restoring of Directed Graph

4 seconds, 256 megabytes

Initially there was a directed graph with n vertices and m edges. The graph did not contain self-loops or multiple edges. Someone started a depth-first search from the first vertex that always went into adjacent vertices in the increasing order of their indices.

You are given a tree of this depth-first search, you are to restore any such directed graph with n vertices and m edges, that a depth-first search that starts from the first vertex and that always goes into adjacent vertices in the increasing order of their indices builds exactly the same tree of depth-first search. If there is no such graph, print "-1" (without quotes).

Input
The first line contains two integers n and m ($1 \le n, m \le 2 \cdot 10^5$) — the number of vertices and the number of edges in the initial directed graph.

The second line contains the depth-first search tree as an array $p_2, p_3, ..., p_n$ ($1 \le p_i \le n$), where p_i is the parent of the i-th vertex in the tree of depth-first search. The root is the vertex with index 1.

Output
If there is no directed graph with n vertices and m edges and the given tree of depth-first search, print "-1" (without quotes).

Otherwise, print m pairs of integers — the edges of such a directed graph, that a depth-first search that starts from the first vertex and that always goes into adjacent vertices in the increasing order of their indices builds exactly the given tree of depth-first search. It is guaranteed, that, if such a graph exists, it does not contain loops nor multiple edges.

input
```
4 4
4 1 1
```
output
```
4 2
1 3
1 4
3 1
```

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

P. Graph Condensation

4 seconds, 256 megabytes

You are given a directed graph consisting of n vertices and m edges. Loops and multiple edges are allowed.

A pair of vertices (u, v) belongs to the same strongly connected component, if v can be reached from u traversing edges and vice versa.

We can construct a new graph where vertices are strongly connected components of the given graph, and an edge exists if and only if there is an edge between some pair of vertices from corresponding strongly connected components. Obviously, the constructed graph is acyclic. It is called a `condensation`. Since it is acyclic, it is possible to sort this graph topologically. Do it.

Input
The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5$, $0 \le m \le 2 \cdot 10^5$) — number of vertices and edges.

The following m lines contains **directed** edges: edge i is given as a pair of the vertices v_i, u_i ($1 \le v_i, u_i \le n, u_i \ne v_i$). Loops and multiple edges are allowed for this graph.

Output
In the first line print the number k — the number of strongly connected components of the given graph.

In the second line print n numbers — for each vertex, print its strongly connected component id. Strongly connected components must be numbered in such a way, that for each edge (u, v) the id of strongly connected component of u must be not greater than the id of strongly connected component of v.

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

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