CSE221 Assignment 05 Summer 2025

A. Can you Traverse-1?

5 seconds, 256 megabytes

A useful tool for constructing graphs:
https://csacademy.com/app/graph_editor/

You are given an **undirected unweighted** graph with ***N*** cities and ***M*** roads. The cities are numbered from 1 to N. You may assume, the graph is connected, meaning there is a path between any pair of cities. There are no self-loops (no road connects a city to itself) and no multiple edges between the same pair of cities.

Your task is to perform a Breadth-First Search (BFS) starting from node $\mathbf{1}$ and print the order in which the nodes are visited.

Pseudocode of BFS

The breadth-first-search procedure below assumes that the input graph G = (V, E) is represented using adjacency lists.

```
BFS(G,s):

for each vertex u in G.V:
   u.colour = 0

Q = ∅ ;
s.colour = 1
ENQUEUE(Q,s)

while Q ≠ ∅;
   u = DEQUEUE(Q)
   for each v in G.Adj[u]:
      if v.colour == 0:
         v.colour = 1
         ENQUEUE(Q,v)
```

Input

The first line contains two integers N and M $(1 \leq N \leq 2 \times 10^5, 1 \leq M \leq 3 \times 10^5)$ — the number of cities and the total number of roads.

The next M lines will contain two integers $u_i, v_i (1 \leq u_i, v_i \leq N)$ — denoting there is an edge between city u_i and city v_i.

Output

Print the BFS traversal starting from node 1 as a space-separated list of visited nodes. If there are multiple BFS path traversal order, you may print any.

input

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

output

```
1 3 4 2
```

input

```
6 10
3 1
1 6
6 4
4 5
5 2
6 2
4 3
5 6
3 6
1 5
```

output

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

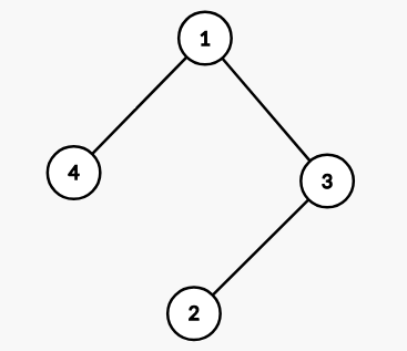
input

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

output

```
1 3 4 2
```

In Sample Test Case 1, the graph looks like this:



There are two valid paths that follow the BFS order: $1\ 4\ 3\ 2$ and $1\ 3\ 4\ 2$. You may print either one of these paths in the output.

B. Can you Traverse-2?

1 second, 1024 megabytes

You are given an **undirected unweighted** graph with ***N*** cities and ***M*** roads. The cities are numbered from 1 to N. The graph is connected, and contains no self-loops or multiple edges.

Your task is to perform a Depth-First Search (DFS) starting from node $\mathbf{1}$ and print the order in which the nodes are visited.

Pseudocode of DFS

The depth-first-search procedure below assumes that the input graph G = (V, E) is represented using adjacency lists.

```
colourInitializing(G):
   for each vertex u in G.V:
      u.colour = 0

DFS(G,u):
   u.colour = 1
   for each v in G.Adj[u]:
      if v.colour == 0:
         DFS(G,v)
```

Important Notes for Python Language

Python has a default recursion limit (usually around 1000) to prevent stack overflow from runaway recursion. As a result, if you are using recursion to solve a problem, you must increase the recursion limit manually.

- **How to Increase the Recursion Depth—**

```
import sys
sys.setrecursionlimit(#set_the_value)
```

- **What Value to Set—** Setting the recursion depth too high can cause a stack overflow or crash your program. This happens because each

recursive call uses stack memory, and the system allocates only a limited amount of memory for the stack. If the recursion goes deeper than the system can handle, it may lead to a segmentation fault. Check what the maximum recursion depth might be in your problem. In this problem, the maximum number of nodes the input graph can have is 2×10^5, so we set the limit accordingly. We add a small buffer (+5, or sometimes +10, +50, etc.) to avoid edge-case runtime errors.

```
import sys
sys.setrecursionlimit(2*100000+5)
```

- **Choose Python over PyPy to Avoid MLE/RTE—** While PyPy is generally faster due to Just-In-Time (JIT) compilation, it also uses more memory. In recursion-heavy problems, PyPy may consume significantly more stack memory per call compared to CPython. This can lead to Memory Limit Exceeded (MLE) errors on some platforms, especially when recursion depth is high. Therefore, for problems that involve deep recursion and strict memory constraints, it is safer to submit your solution using Python (CPython) rather than PyPy.

Input
The first line contains two integers N and M $(1 \leq N \leq 2 \times 10^5, 1 \leq M \leq 3 \times 10^5)$ — the number of cities and the total number of roads.

The second line contains M integers $u_1, u_2, u_3 \ldots u_m$ $(1 \leq u_i \leq N)$ — where the i-th integer represents the node that is one endpoint of the i-th edge.

The third line contains M integers $v_1, v_2, v_3 \ldots v_m$ $(1 \leq v_i \leq N)$ — where the i-th integer represents the node that is other endpoint of the i-th edge.

The i-th edge of this graph is between the i-th node in the second line and the i-th node in the third line.

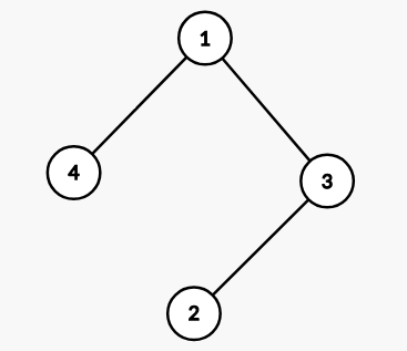
Output
Print the DFS traversal starting from node 1 as a space-separated list of visited nodes. If there are multiple DFS path traversal order, you may print any.

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

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

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

In the first Sample Input, the graph looks like this:



There are two valid paths that follow the DFS order: $1\ 4\ 3\ 2$ and $1\ 3\ 2\ 4$. You may print either one of these paths in the output.

C. Lightning McQueen

1 second, 256 megabytes

You are given an **undirected unweighted** graph with N nodes and M edges. The nodes are numbered from 1 to N. The graph contains no self-loops or multiple edges.

There is a source and a destination. Your task is to find the shortest distance from the source node to destination node and print the path taken. If multiple shortest paths exist, print the one that is lexicographically smallest.

A path $P1 = [a_1, a_2, \ldots a_n]$ is lexicographically smaller than a path $P2 = [b_1, b_2, \ldots b_m]$ if at the first position where they differ, $a_i < b_i$. For example, $[1, 4, 3]$ is smaller than $[1, 5, 7, 1]$.

If no path exists, print -1.

Input
The first line contains four integers N, M, S, D $(1 \leq N \leq 2 \times 10^5, 0 \leq M \leq 3 \times 10^5, 1 \leq S,D \leq N)$ — the number of vertices, total number of edges, source and destination.

The second line contains M integers $u_1, u_2, u_3 \ldots u_m$ $(1 \leq u_i \leq N)$ — where the i-th integer represents the node that is one endpoint of the i-th edge.

The third line contains M integers $v_1, v_2, v_3 \ldots v_m$ $(1 \leq v_i \leq N)$ — where the i-th integer represents the node that is other endpoint of the i-th edge.

The i-th edge of this graph is between the i-th node in the second line and the i-th node in the third line.

Output

- If a path exists, print the length of the shortest path (number of edges) on the first line.
- On the second line, print the lexicographically smallest shortest path from source to destination.
- If no path exists, print -1.

input
5 10 5 3 2 1 5 3 1 4 2 4 1 4 5 5 4 5 2 2 3 1 3 3
output
1 5 3

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

input

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

output

```
2
3 7 2
```

input

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

output

```
-1
```

input

```
1 0 1 1
```

output

```
0
1
```

D. Through the Jungle

1 second, 256 megabytes

You are given a **directed unweighted** graph with N nodes and M edges. The nodes are numbered from 1 to N. The graph contains no self-loops or multiple edges.

You have to find a shortest path from node S to node D that passes through node K. If multiple such paths exist, print any one of them. If no such path exists, print -1.

Input

The first line contains five integers N, M, S, D, K $(1 \leq N \leq 2 \times 10^5, 1 \leq M \leq 3 \times 10^5, 1 \leq S,D,K \leq N)$ — the number of vertices, total number of edges, source, destination and the mandatory node that must be included in the path.

The next M lines will contain two integers $u_i, v_i (1 \leq u_i, v_i \leq N)$ — denoting there is an edge from city u_i to city v_i.

Output

- If a valid path exists from S to D through K, print the length of the path (number of edges) on the first line.
- On the second line, print the nodes in the path in order from S to D.
- If no such path exists, print -1.

input

```
5 10 5 3 5
2 5
5 1
4 5
3 5
1 2
2 4
3 2
1 4
1 3
3 4
```

output

```
2
5 1 3
```

input

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

output

```
-1
```

input

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

output

```
-1
```

input

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

output

```
0
2
```

E. Easy Tree Queries

1 second, 1024 megabytes

There is a tree with **N** nodes. The tree is rooted at a given node **R**.

You will be given **Q** queries. In each query, you are asked to find the size of the subtree of a given node **X**.

Input

The first line contains two integers N, R $(1 \leq N \leq 2 \times 10^5, 1 \leq R \leq N)$ — the number of nodes and the root of the tree.

The next $N-1$ lines each contain two integers $u_i, v_i (1 \leq u_i, v_i \leq N)$ — representing an bidirectional edge between nodes u_i and v_i.

The next line contains an integer $Q (1 \leq Q \leq 2 \times 10^5)$ — the number of queries.

The next Q lines each contain a single integer $X (1 \leq X \leq N)$ — the node whose subtree size you need to compute.

Output

For each query, print a single integer — the size of the subtree of node X.

input

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

output

```
4
1
2
```

input

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

output

```
5
1
1
3
1
```

input

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

output

```
4
1
8
1
2
5
2
3
```

input

```
1 1
1
1
```

output

```
1
```

F. Cycle Detection

1 second, 256 megabytes

You are given a **directed unweighted** graph with N nodes and M edges. The nodes are numbered from 1 to N. The graph contains no self-loops or multiple edges.

Write a code to find if there is any cycle in the graph. In graph theory, a cycle in a graph is a non-empty trail in which only the first and last vertices are equal. [Wikipedia]

Input

The first line contains four integers N and M $(1 \leq N \leq 2 \times 10^5, 1 \leq M \leq 2 \times 10^5)$ — the number of vertices and total number of edges.

The next M lines will contain two integers $u_i, v_i (1 \leq u_i, v_i \leq N)$ — denoting there is an edge from city u_i to city v_i.

Output

Print YES if the graph contains any cycle, otherwise print NO.

input

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

output

```
YES
```

input

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

output

```
NO
```

G. Diamonds under W

1 second, 256 megabytes

You are given a 2D grid with R rows and H columns.

Each cell in the grid is one of the following:

1. . — Empty cell: You can move into this cell.
2. D — Cell with a diamond: You can move into this cell and collect the diamond.
3. $\#$ — Obstacle: You cannot move into this cell.

You may start from any non-obstacle cell and move in the four directions: up, down, left, or right. Your goal is to choose a starting cell such that you can collect the maximum number of diamonds

Input

The first line contains two integers R and H $(1 \leq R, H \leq 1000)$ — the number of rows and columns of the grid. The next R lines each contain H characters, describing the grid.

Output

Print a single integer — the maximum number of diamonds you can collect starting from a valid cell.

input

```
5 5
.#.DD
##.#.
####D
#DDD#
#..DD
```

output

```
5
```

input

```
5 5
D####
##.D#
#..D#
###D#
..##D
```

output

```
3
```