Programming Part

Problem 3 - Dynamic Spectrum Artistry (100 pts)

Problem Description

Story

One day, the employees at Dertian Space Agency grew tired of playing **Dungeons of Sacred Abyss** repeatedly, as they had already mastered its challenges. Recognizing their need for something new and exciting, Steve, the agency's president, reached out to Nextgen Technology Universal Company of Software (NTUCS) for help in creating a brand-new game.

Dynamic Spectrum Artistry is a fresh and innovative game from NTUCS. This game combines the enjoyable activity of coloring with an adventurous exploration of a colorful and vibrant universe. Its goal is to reignite the enthusiasm and creativity of the employees at Dertian Space Agency, providing them with an engaging and enjoyable gaming experience.

In this game, we have *blocks* which can be painted with different colors, and directed *arrows* that points from one block to another block. Let B denote the set of blocks and A denote the set of arrows in the game.

The colors of the blocks should satisfy the following rule. If $b_1, b_2 \in B, b_1 \neq b_2$, and there exist both a path from b_1 to b_2 and a path from b_2 to b_1, following the arrows in A, then $b_1.\mathit{color} = b_2.\mathit{color}$. On the other hand, if there exist either no path from b_1 to b_2 or no path from b_2 to b_1, following the arrows in A, then $b_1.\mathit{color} \neq b_2.\mathit{color}$.

Steve would like to determine *the number of colors* that needs to be used to color all the blocks in B. In addition, Steve would like to determine if there exist an acyclic path formed by arrows in A, such that the blocks included in this path have *all colors* used in B.

Reference Reading

The following provides information about *Strongly Connected Components* that may be helpful to solve this problem.

Strongly Connected Components Definition: [from Wikipedia] The strongly connected components (SCCs) of a directed graph form a partition into subgraphs that are themselves strongly connected. A directed graph is called *strongly connected* if there is a path in each direction between each pair of vertices of the graph. That is, a path exists from the first vertex in the pair to the second, and another path exists from the second vertex to the first. If each SCC is contracted to a single vertex, the resulting graph G^{SCC} is a directed acyclic graph (DAG), the condensation of G.

Finding SCCs. There are multiple methods for finding SCCs in a directed graph. In the following section, we will introduce one such approach: the Kosaraju's algorithm, though you are free to use other methods to solve this problem. The pseudo code provided below is from *Chapter 20.5* of the textbook, which we strongly encourage you to read and understand. Below we will introduce the pseudo code and provide a brief explanation.

```
STRONGLY-CONNECTED-COMPONENTS($G$)
1  Perform DFS($G$) to compute finishing times $u.f$ for each vertex $u$
2  Compute the transpose graph $G^T = (V, E^T)$ by reversing the direction of all edges
3  Perform DFS($G^T$), but in the main loop of DFS,
      consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4  Output the vertices of each tree in the depth-first forest
      formed in line 3 as a separate SCC
```

Assume we are trying to find all SCCs in a given directed graph $G = (V, E)$. Assume G has strongly connected components $C_1, C_2, \ldots, C_k$, each of which is a set of vertices in the component. We can also define the finish time for a set of vertices U to be the maximum of the finish times of all of its vertices, i.e., $f(U) = \max_{u \in U}\{u.f\}$.

Let C and C' be distinct SCCs in G. If we have an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$, then we always have $f(C) > f(C')$[a]. This says if we have an edge going from a vertex in C to a vertex in C', then $f(C)$ is larger. Now if we reverse the edge directions to obtain $G^T = (V, E^T)$ (line 3), and we have an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$, then because of the reversal of the edge directions we have the opposite result $f(C) < f(C')$[b]. Moreover, between two distinct SCCs C_i and C_j there can only exist edges going either from C_i to C_j or from C_j to C_i, but not both. Otherwise, C_i and C_j would be within the same SCC.

Now we are ready to take a look at STRONGLY-CONNECTED-COMPONENTS. Line 1 and line 2 pre-processes G by calculating the finish times of all vertices and obtain the transpose graph G^T.

Line 3 deserves additional explanation. We visit the SCC with the largest finish time (in the DFS in line 1) $C_{\max}$. This implies there can only exist edges going into $C_{\max}$ from other SCC, but not the opposite (otherwise, C_{max} would not have the largest finish time). If we start DFS traversal from any vertex in $C_{\max}$, it will visit all vertices in $C_{\max}$, but not moving to the other SCCs as there is no such edge. Then, DFS will move to the SCC with the second largest finish time $C_{\max-1}$, and in this case there can only exist edges going into $C_{\max-1}$ from any SCCs other than $C_{\max}$. DFS will now visit all vertices in $C_{\max-1}$ but not moving to other SCCs. The procedure continues until all SCCs are visited and the results are output in line 4.

Condensing the SCCs. We can transform the original graph G into a condensed graph G^{SCC} such that all vertices in the same SCC is merged into a single vertex. Any edge between vertices in the same SCC is now a self edge and removed. Multi-edges (duplicate edges) between SCCs are also removed.

Finding a path connecting all SCCs. We start by performing a topological sort on G^{SCC}. Let $v_1, v_2, \ldots, v_{|V^{SCC}|}$ denote the sorted sequence of vertices. If for any neighboring vertices in the sequence $v_i, v_{i+1}, 1 \leq i \leq |V^{SCC}| - 1$, there exist an edge (v_i, v_{i+1}), then we are able to establish a path connecting all vertices in G^{SCC}, i.e., connecting all SCCs in the original graph G.

[a]see the proof of Lemma 20.14 in the textbook

[b]see the proof of Corollary 20.15 in the textbook

Input

The first line contains 3 integers $|B|$, $|A|$, and `mode`. $|B|$ is the number of blocks, and $|A|$ is number of arrows in the game. `mode` has a value of either 1 or 2. If `mode` is 1, you only need to answer the first question. If `mode` is 2, you have to answer both questions. Each of the next $|A|$ lines contains two integers u_i and v_i, representing the starting block and ending block of the i-th arrow. The integers in the same line are separated by single white spaces.

Output

In the first line, please print the number of colors that needs to be used to color all the blocks in B. The second line is only required if `mode` is 2. If so, in the second line, indicate whether there exist an acyclic path formed by arrows in A, such that the blocks included in this path have all colors used in B. If the path does not exist, print 0; otherwise, print 1.

Constraints

- $0 \le |A| \le \min(|B| \cdot (|B| - 1), 10^6)$
- $1 \le u_i, v_i \le |B|$
- There are no duplicated arrows.
- `mode` $\in \{1, 2\}$
- Time Limit: 3 s
- Memory Limit: 262144 KB

Subtasks

Subtask 1 (15 pts)

- $1 \le |B| \le 10^3$
- `mode` $= 1$

Subtask 2 (25 pts)

- $1 \le |B| \le 10^5$
- `mode` $= 1$

Subtask 3 (15 pts)

- $1 \le |B| \le 10^3$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 1; the only difference is the `mode`.

Subtask 4 (45 pts)

- $1 \le |B| \le 10^5$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 2; the only difference is the `mode`.

Sample Testcases

Sample Input 1

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

Sample Output 1

```
2
1
```

Explanation for Sample 1

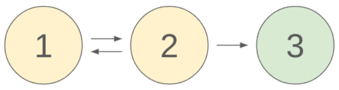

Figure 2.1

Block 1 can reach block 2, and vice versa; hence, blocks 1 and 2 should be painted the same color. Conversely, although block 2 can reach block 3, the reverse is not true; therefore, blocks 2 and 3 should NOT be painted the same color. You have used a total of 2 colors. As there exists a path that traverses all colors (2 $\rightarrow$ 3), the answer to the second problem is 1.

Sample Input 2

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

Sample Output 2

```
3
0
```

Explanation for Sample 2

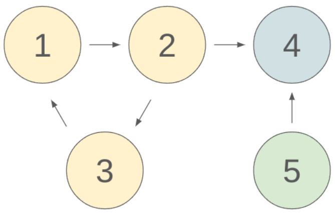

Figure 2.2

Block 1 can reach block 2 (1→2), and vice versa (2→3→1); thus, blocks 1 and 2 should be painted the same color. Similarly, blocks 2 and 3, as well as blocks 1 and 3, should also be painted the same color, resulting in colors being shared among blocks 1, 2, and 3. Additionally, blocks 4 and 5 must each be painted a different color from one another. You have used a total of 3 colors. Since there is no path that traverses all colors, the answer to the second problem is 0.