Graph Representation in Programming Assignments

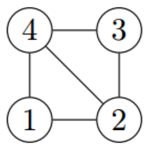
In programming assignments, graphs are given as follows. The first line contains non-negative integers n and m – the number of vertices and the number of edges respectively. The vertices are always numbered from 1 to n. Each of the following m lines defines an edge in the format u v where $1 \leq u, v \leq n$ are endpoints of the edge. If the problem deals with an undirected graph this defines an undirected edge between u and v, In case of a directed graph this defines a directed edge from u to v. If the problem deals with a weighted graph then each edge is given as u v w where u and v are vertices and w is a weight.

It is guaranteed that a given graph is simple. That is, it does not contain self-loops (edge going from a vertex to itself) and parallel edges.

Examples:

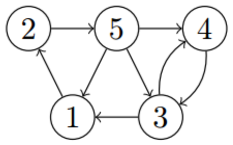
- An undirected graph with four vertices and five edges:

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



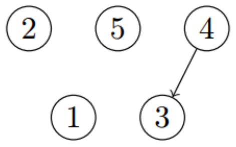
- A directed graph with five vertices and eight edges:

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



- A directed graph with five vertices and one edge:

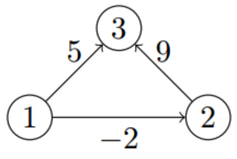
```
5 1
4 3
```



Note that the vertices 1, 2, and 5 are isolated (have no adjacent edges), but they are still present in the graph.

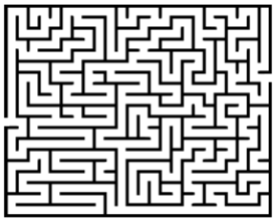
- A weighted directed graph with three vertices and three edges:

```
3 3
2 3 9
1 3 5
1 2 -2
```



1 Finding an Exit from a Maze

Problem Introduction



A maze is a rectangular grid of cells with walls between some of adjacent cells. You would like to check whether there is a path from a given cell to a given exit from a maze where an exit is also a cell that lies on the border of the maze (in the example shown to the right there are two exits: one on the left border and one on the right border). For this, you represent the maze as an undirected graph: vertices of the graph are cells of the maze, two vertices are connected by an undirected edge if they are adjacent and there is no wall between them. Then, to check whether there is a path between two given cells in the maze, it suffices to check that there is a path between the corresponding two vertices in the graph.

Problem Description

Task. Given an undirected graph and two distinct vertices u and v, check if there is a path between u and v.

Input Format. An undirected graph with n vertices and m edges. The next line contains two vertices u and v of the graph.

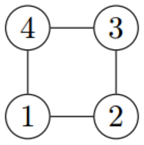
Constraints. $2 \leq n \leq 10^3$; $1 \leq m \leq 10^3$; $1 \leq u, v \leq n$; $u \neq v$.

Output Format. Output 1 if there is a path between u and v and 0 otherwise.

Sample 1.

Input:

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



Output:

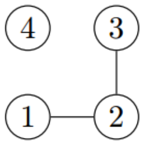
```
1
```

In this graph, there are two paths between vertices 1 and 4: $1 - 4$ and $1 - 2 - 3 - 4$.

Sample 2.

Input:

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



Output:

```
0
```

In this case, there is no path from 1 to 4.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `reachability`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

2 Adding Exits to a Maze

Problem Introduction

Now you decide to make sure that there are no dead zones in a maze, that is, that at least one exit is reachable form each cell. For this, you find connected components of the corresponding undirected graph and ensure that each component contains an exit cell.

Problem Description

Task. Given an undirected graph with n vertices and m edges, compute the number of connected components in it.

Input Format. A graph is given in the standard format.

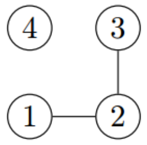
Constraints. $1 \leq n \leq 10^3$; $0 \leq m \leq 10^3$.

Output Format. Output the number of connected components.

Sample 1.

Input:

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



Output:

```
2
```

There are two connected components here: $\{1, 2, 3\}$ and $\{4\}$.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `connected_components`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

3 Checking Consistency of CS Curriculum

Problem Introduction

A computer Science curriculum specifies the prerequisites for each course as a list of courses that should be taken before taking this course. You would like to perform a consistency check of the curriculum, that is, to check that there are no cyclic dependencies. For this, you construct the following directed graph: vertices correspond to courses, there is a directed edge (u, v) if the course u should be taken before the course v. Then, it is enough to check whether the resulting graph contains a cycle.

Problem Description

Task. Check whether a given directed graph with n vertices and m edges contains a cycle.

Input Format. A graph is given in the standard format.

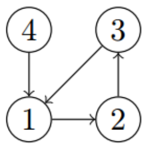
Constraints. $1 \leq n \leq 10^3$; $0 \leq m \leq 10^3$.

Output Format. Output 1 if the graph contains a cycle and 0 otherwise.

Sample 1.

Input:

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



Output:

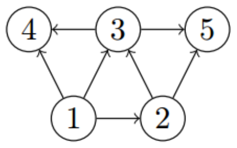
```
1
```

This graph contains a cycle: $3 \to 1 \to 2 \to 3$.

Sample 2.

Input:

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



Output:

```
0
```

There is no cycle in this graph. This can be seen, for example, by noting that all edges in this graph go from a vertex with a smaller number to a vertex with a larger number.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `acyclicity`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

4 Determine an Order of Courses

Problem Introduction

Now, when you are sure that there are no cyclic dependencies in the given CS curriculum, you would like to find an order of all courses that is consistent with all dependencies. For this, you find a topological ordering of the corresponding directed graph.

Problem Description

Task. Compute a topological ordering of a given directed acyclic graph (DAG) with n vertices and m edges.

Input Format. A graph is given in the standard format.

Constraints. $1 \leq n \leq 10^5$; $0 \leq m \leq 10^5$. The given graph is guaranteed to be acyclic.

Output Format. Output *any* topological ordering of its vertices. (Many DAGs have more than just one topological ordering. You may output any of them.)

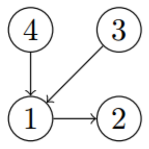
Sample 1.

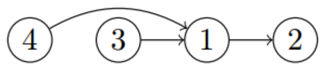
Input:

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

Output:

```
4 3 1 2
```





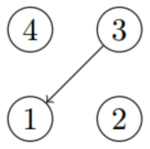
Sample 2.

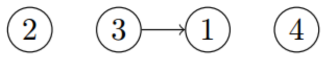
Input:

```
4 1
3 1
```

Output:

```
2 3 1 4
```





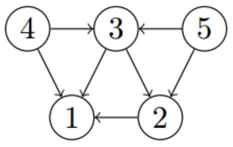
Sample 3.

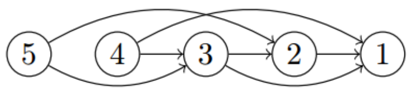
Input:

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

Output:

```
5 4 3 2 1
```





Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `toposort`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

5 Advanced Problem: Checking Whether Any Intersection in a City is Reachable from Any Other

It is strongly recommended that you start solving advanced problems only when you are done with the basic problems (for some advanced problems, algorithms are not covered in the video lectures and require additional ideas to be solved; for some other advanced problems, algorithms are covered in the lectures, but implementing them is a more challenging task than for other problems).

Problem Introduction

The police department of a city has made all streets one-way. You would like to check whether it is still possible to drive legally from any intersection to any other intersection. For this, you construct a directed graph: vertices are intersections, there is an edge (u, v) whenever there is a (one-way) street from u to v in the city. Then, it suffices to check whether all the vertices in the graph lie in the same strongly connected component.

Problem Description

Task. Compute the number of strongly connected components of a given directed graph with n vertices and m edges.

Input Format. A graph is given in the standard format.

Constraints. $1 \leq n \leq 10^4$; $0 \leq m \leq 10^4$.

Output Format. Output the number of strongly connected components.

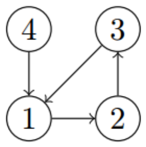
Sample 1.

Input:

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

Output:

```
2
```



This graph has two strongly connected components: $\{1, 3, 2\}$, $\{4\}$.

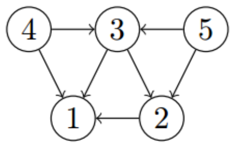
Sample 2.

Input:

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

Output:

```
5
```



This graph has five strongly connected components: {1}, {2}, {3}, {4}, {5}.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch.

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.