Graphs – Maze Revisited

**Purpose**

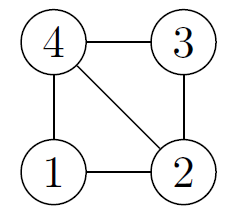
In this and the next programming assignments you will be practicing implementing the basic building blocks of graph algorithms: computing the number of connected components, checking whether there is a path between the given two vertices, checking whether there is a cycle, etc. Such building blocks are used practically in all applications working with graphs: for example, finding shortest paths on maps, analyzing social networks, analyzing biological data.

**Graph Representation in Programming Assignments**

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 ≤ 𝑢, 𝑣 ≤ 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 (edges going from a vertex to itself) and parallel edges.

Examples:

∙ An undirected graph with four vertices and five edges:

4 5

2 1

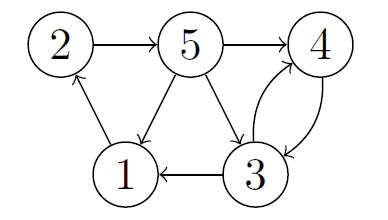
4 3

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

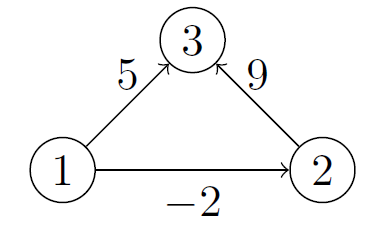
2 5

5 1

5 4

5 3

∙ A weighted directed graph with three vertices and three edges.

3 3

2 3 9

1 3 5

1 2 -2

**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

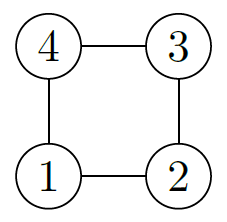
check that there is a path between the corresponding two vertices in the graph.

**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.

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

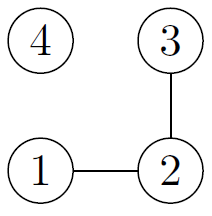
1 4

1 4

Output:

1

Explanation: 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

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

Filename: reachability, tinyG.txt, mediumG.txt, largeG.txt

**What To Do**

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

Run the Junit test cases provided. It took should take about 30 seconds.

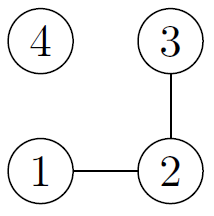
**Problem: Adding Exits to a Maze**

Now you decide to make sure that there are no dead zones in a maze, that is, that at least one exit is reachable from 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.

Output Format. Output the number of connected components.

Sample 1.

Input:

4 2

1 2

3 2

Output:

2

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

Filename: connected\_components

**What To Do**

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures. Your main goal in an algorithmic problem is to implement a program that solves a given computational problem in just few seconds even on massive datasets. Your program should read a dataset from the standard input and write an answer to the standard output.