**CSE 4304-Data Structures Lab. Winter 23-24**

**Batch:** CSE 22

**Date**: February 26, 2025

**Target Group:** All

**Topic**: Graph

**Instructions**:

* Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
* Task naming format: fullID\_T01L01\_2A.c/cpp
* If you find any issues in the problem description/test cases, comment in the Google Classroom.
* If you find any tricky test cases that I didn’t include but that others might forget to handle, please comment! I’ll be happy to add them.
* Use appropriate comments in your code. This will help you recall the solution easily in the future.
* Obtained marks will vary based on the efficiency of the solution.
* Do not use <bits/stdc++.h> library.
* Modified sections will be marked with BLUE color.
* You can use the STL stack unless it’s specifically mentioned that you should use manual functions.

| **Group** | **Tasks** |
| --- | --- |
| 2A | 1 2 3 4 |
| 1B | 1 2 3 4 |
| 1A |  |
| 2B |  |
| **Assignments** | 2A/1B:  1A/2B: |

**Task 1**: Represent a Graph using the Adjacency List and Adjacency Matrix

You are given a graph with V vertices and E edges. Your task is to represent the graph in two different ways:

* Adjacency List: This is a list of vertices where each vertex points to a list of vertices that are connected to it by an edge.
* Adjacency Matrix: This is a 2D matrix V x V, where matrix[i][j] is 1 if there is an edge from vertex i to vertex j and 0 otherwise.

Then, display the following information:

* The Adjacency List representation of the graph.
* The Adjacency Matrix representation of the graph.

**Input Format**:

* The first line contains two integers V and E, where V is the number of vertices and E is the number of edges in the graph.
* The next E lines each contain two integers u and v, indicating an edge from vertex u to vertex v.

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5 6  1 2  1 3  2 4  3 4  3 5  4 5 | Adjacency List:  1: 2 3  2: 1 4  3: 1 4 5  4: 2 3 5  5: 3 4  Adjacency Matrix:  0 1 1 0 0  1 0 0 1 0  1 0 0 1 1  0 1 1 0 1  0 0 1 1 0 |
| 6 3  1 2  2 3  4 5 | Adjacency List:  1: 2  2: 1 3  3: 2  4: 5  5: 4  6:  Adjacency Matrix:  0 1 0 0 0 0  1 0 1 0 0 0  0 1 0 0 0 0  0 0 0 0 1 0  0 0 0 1 0 0  0 0 0 0 0 0 |

**Task 2:** Check if a Graph is k-Regular

You are given an undirected graph with a set of edges. Your task is to determine whether the graph is regular. A graph is considered regular if all vertices have exactly same number of edges.

**Input**:

* The first line contains two integers V and E representing the total number of vertices and edges respectively.
* The following E lines contain edge information (u,v) where each pair represents an edge between two vertices (with no loops or multiple edges).

**Output**:

* Print "Yes" if the graph is regular.
* Print "No" if the graph is not regular.

| **Sample Input** | **Sample Output** | **Explanation** |
| --- | --- | --- |
| 4 4  1 2  2 3  3 4  4 1 | Yes | Each vertex has exactly 2 edges:  Vertex 1 has edges (1-2), (1-4) → degree = 2  Vertex 2 has edges (2-1), (2-3) → degree = 2  Vertex 3 has edges (3-2), (3-4) → degree = 2  Vertex 4 has edges (4-1), (4-3) → degree = 2  The graph is 2-regular. |
| 5 6  1 2  2 3  3 4  4 5  5 1  2 4 | No | Vertex 1 has edges (1-2), (1-5) → degree = 2  Vertex 2 has edges (2-1), (2-3), (2-4) → degree = 3  Vertex 3 has edges (3-2), (3-4) → degree = 2  Vertex 4 has edges (4-3), (4-5), (4-2) → degree = 3  Vertex 5 has edges (5-1), (5-4) → degree = 2  Not all vertices have the same degree, so the graph is not k-regular. |
| 5 10  1 2  2 3  1 3  3 5  3 4  4 5  1 4  1 5  2 4  2 5 | Yes |  |

**Task 3:** Topological Sort Using Queue Approach

You are given a Directed Acyclic Graph (DAG) with N nodes and M edges. Your task is to perform Topological Sort using the Queue approach.

The Queue approach is based on the Kahn's Algorithm for Topological Sorting. In this approach, we maintain a queue to store nodes with no incoming edges (i.e., their indegree is 0). We then remove these nodes from the graph and reduce the in-degree of their neighbors. This continues until there are no more nodes left in the queue.

Your task is to print the topologically sorted order of nodes.

If it is impossible to complete the topological sorting due to a cycle in the graph, return an empty array.

**Input Format**:

* The first line contains two integers N (the number of nodes) and M (the number of edges).
* The next M lines contain two integers a and b representing a directed edge from node a to node b.

**Output Format**:

* Print the topologically sorted order of nodes in a single line separated by spaces.
* If it is not possible to perform topological sorting due to a cycle, print an empty array [].

The graph is guaranteed to be valid (no multiple edges, no self-loops).

| **Sample Input** | **Sample Output** | **Explanation** |
| --- | --- | --- |
| 6 7  3 4  3 5  0 1  0 2  4 5  1 3  2 3 | 0 1 2 3 4 5 | The graph has multiple paths. Both [0, 1, 2, 3, 4, 5] and [0, 2, 1, 3, 4, 5 ] are valid topological sorts. |
| 5 5  4 2  4 3  3 1  2 0  3 0 | 4 2 0 3 1 |  |
| 4 4  0 1  0 2  1 3  2 3 | 0 1 2 3 | Both [0, 1, 2, 3] and [0, 2, 1, 3] are valid topological sorts. The graph allows for two different valid orderings. |
| 3 3  0 1  1 2  2 0 | [ ] | The graph contains a cycle (0 → 1 → 2 → 0), which makes topological sorting impossible. The output is an empty array []. |
| 6 3  0 1  2 3  4 5 | 0 1 2 3 4 5 | The graph is disconnected, but the topological sorting can still be done independently for each component. There can be multiple valid topological orderings like [0, 1, 2, 3, 4, 5] or [2, 3, 0, 1, 4, 5]. |
| 5 5  0 1  1 2  2 3  3 4  4 2 | [ ] | The graph contains a cycle (2 → 3 → 4 → 2), so topological sorting is not possible. |
| 6 6  0 1  0 2  1 3  2 3  3 4  3 5 | 0 1 2 3 4 5 | This is a larger graph with multiple valid topological sorts, and both orders are acceptable as valid outputs. |

**Task 4**: Check if there exists a path between two nodes

You are given a directed graph with N nodes and M edges. The graph is represented by pairs of integers, where each pair (a, b) denotes a directed edge from node a to node b. Your task is to determine whether there exists a path from a source node src to a destination node dest.

**Input**:

* The first line contains two integers N and M, where N is the number of nodes, and M is the number of edges.
* The next M lines each contain two integers a and b, denoting a directed edge from node a to node b.
* The last line contains two integers src and dest, representing the source and destination nodes.

**Output**:

* Print "YES" if there exists a path from src to dest.
* Print "NO" if there does not exist a path from src to dest.

| **Sample Input** | **Sample Output** | **Explanation** |
| --- | --- | --- |
| 4 3  1 2  2 3  3 4  1 4 | Yes | There is a path from node 1 to node 4: 1 → 2 → 3 → 4.  Therefore, the output is "YES". |
| 5 8  1 2  1 3  2 4  2 5  4 4  5 2  4 3  3 5  5 3 | Yes | 5 2 4 3 |
| 5 8  1 2  1 3  2 4  2 5  4 4  5 2  4 3  3 5  2 1 | No |  |
| 5 4  1 2  2 3  4 5  4 2  1 5 | No | Although node 1 can reach 2, 3, and 4, there is no path from node 1 to node 5. |
| 5 2  1 2  3 4  1 5 | No | Disconnected Graph |