CS204(2025)

Graphs

Instructions for Implementation

- Write the program in C or C++ using standard input/output.
- Follow the input/output format strictly.
- Ensure your code:
  - Handles **edge cases** correctly.
  - Meets the **time and space complexity constraints**.
- You are encouraged to write clean, modular, and well-documented code.

Q1. Graph Traversal using Depth-First Search (DFS)

Problem Statement:

You are given a **graph** represented as an `n × n` **adjacency matrix**, where each cell `graph[i][j]` is:

- `1` if there is an edge from vertex `i` to vertex `j`, and
- `0` otherwise.

Write a program to perform a **Depth-First Search (DFS)** traversal of the graph, starting from **vertex 0** (the first vertex).

Your task is to **visit each vertex exactly once**, and print the traversal order.

If multiple vertices are reachable from the same vertex, **visit the smallest-numbered vertex first** to ensure a **unique traversal order**.

Input Format:

1. The first line contains an integer `n` — the number of vertices in the graph.
2. The next `n` lines each contain `n` integers (0 or 1), representing the adjacency matrix.

Output Format:

- Print the **DFS traversal order** of the graph as space-separated vertex numbers on a single line.

Constraints:

- 1 ≤ n ≤ 20
- Vertices are numbered from 0 to n-1.

Example 1:

Input

5

0 1 1 0 0

```
1 0 0 1 0
1 0 0 0 1
0 1 0 0 1
0 0 1 1 0
```
Output:
```
0 1 3 4 2
```

Example 2:
Input:
```
4
0 1 0 0
1 0 1 0
0 1 0 0
0 0 0 0
```
Output:
```
0 1 2 3
```

Q2. Graph Traversal using Breadth-First Search (BFS)

Problem Statement:

You are given an **undirected graph** represented by an **edge list**.
Your task is to perform a **Breadth-First Search (BFS)** traversal starting from **vertex 0**, visiting each vertex **exactly once**.

If multiple vertices are reachable from a node, always visit the **smallest-numbered vertex first** to ensure a **unique traversal order**.

If the graph is **disconnected**, continue the BFS for all unvisited vertices in ascending order of vertex numbers.

Input Format:

1. The first line contains two integers n and e, where
    - n = number of vertices (0 to n-1)
    - e = number of edges.
2. The next e lines each contain two integers u v, representing an undirected edge between vertices u and v.

Output Format:

- Print the **BFS traversal order** as space-separated vertex numbers in a single line.

Constraints:

- $1 \le n \le 20$
- $0 \le e \le n \times (n-1)/2$
- Vertices are numbered from 0 to $n-1$
- No duplicate edges or self-loops

Example 1:

Input:

5 6
0 1
0 2
1 3
2 4
3 4
1 2
Output: 0 1 2 3 4
Example 2:

Input:
4 2
0 1
2 3
Output: 0 1 2 3

Q3. Given an m x n 2D binary grid which represents a map of '1's (land) and '0's (water), return the number of islands.
m = grid length and n = grid[i] length.
An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Input:
- First line contains m
- The second line contains n
- Next m lines contain n space-separated integers

Output:
- Print an integer representing no.of islands.

Test cases:

1. Input :
   4
   5
   1 1 1 1 0
   1 1 0 1 0
   1 1 0 0 0
   0 0 0 0 0
   Output: 1

2. Input:
   4
   5
   1 1 0 0 0
   1 1 0 0 0
   0 0 1 0 0
   0 0 0 1 1
   Output: 3

Q4. You are given an undirected graph with `n` vertices and `m` edges. Determine whether the graph is bipartite or not.

A graph is said to be bipartite if it is possible to divide its vertices into two disjoint sets such that no two vertices within the same set are connected by an edge.

Input Format:

- The first line contains two integers `n` and `m`, representing the number of vertices and edges, respectively.
- The next `m` lines each contain two integers `u` and `v`, indicating there is an undirected edge between vertex `u` and vertex `v`.

Output Format:

Print **"YES"** if the graph is bipartite, otherwise print **"NO"**.

Test Cases:

Input:
```
4 4
1 2
2 3
3 4
4 1
```
Output:
```
YES
```

Input:
```
3 3
1 2
2 3
3 1
```
Output:
```
NO
```

Q5. You are given a **directed graph** with `n` vertices and `m` edges. Your task is to determine whether the graph contains **a cycle** or not.

A **cycle** in a directed graph is a path that starts and ends at the same vertex, following the direction of edges.

Input Format:

- The first line contains two integers `n` and `m`, representing the number of vertices and edges, respectively
- The next `m` lines each contain two integers `u` and `v`, indicating that there is a **directed edge** from vertex `u` to vertex `v`

Output Format:
Print **"YES"** if the graph contains a cycle, otherwise print **"NO"**

Test Cases:

Input:
4 4
1 2
2 3
3 4
4 1
Output:
YES

Input:
3 2
1 2
2 3
Output:
NO

Q6. You are given `n` courses labeled from `1` to `n` and `m` prerequisite relations of the form **"course a must be completed before course b"**.
Your task is to determine a possible order in which all the courses can be completed.

If there is **no valid order** (i.e., the prerequisites form a cycle), print **"-1"**

This is a **topological sorting** problem on a **directed graph**, where each course represents a node and each prerequisite represents a directed edge.

Input Format:

- The first line contains two integers `n` and `m` — the number of courses and the number of prerequisite relations.
- The next `m` lines each contain two integers `a` and `b`, indicating that course `a` must be completed before course `b`.

Output Format:

- Print **"YES"** if there exists an ordering, otherwise print **"NO"**

Test Cases:

Input:
4 3
1 2
2 3
3 4

Output:
YES

Input:
4 4
1 2
2 3
3 1
3 4

Output:
NO

Q7. You are given an **undirected, connected graph** with `n` vertices and `m` edges.
Your task is to find the **diameter** of the graph.
The **diameter** of a graph is the **longest among all the shortest paths** between any two vertices.

Input Format:

The first line contains two integers `n` and `m`, representing the number of vertices and edges, respectively.
The next `m` lines each contain two integers `u` and `v`, representing an **undirected** edge between vertices `u` and `v`

Output Format:

Print a single integer — the **diameter of the graph**

Test Cases:

Input:
4 3
1 2
2 3
3 4

Output:
3

Input:
5 4
1 2
2 3
3 4
4 5

Output:
4

Q8. Given a directed graph with n vertices (numbered from 0 to n-1) and m directed edges, determine whether there exists a path from a given source vertex u to a destination vertex v.

Input:

First line: Two space-separated integers

n — number of vertices

m — number of directed edges

Next m lines: Each line contains two integers a b, meaning there is a directed edge from vertex a → vertex b.

Last line: Two integers u v — the source and destination vertices.

Output:

Output "YES" if there exists a path from u to v, otherwise output "NO".

Sample Testcases:

1. Input:
   ```
   4 4
   0 1
   1 2
   2 3
   0 3
   0 3
   ```
   Output: YES
2. Input:
   ```
   4 2
   0 1
   2 3
   0 3
   ```
   Output: NO