National University of Computer and Emerging Sciences



# Lab Manual

*for*

# Data Structure

Department of Computer Science FAST-NU, Lahore, Pakistan

**Objectives:**

After performing this lab, students shall be able to revise:

* Graphs
* BFS

**Question 1)**

Write a C++ program to:

1. Represent a graph using:
   * **Adjacency List**
2. Provide the following functionalities:
   * Add an edge between two vertices.
   * Display the adjacency list representations.
   * Determine if there is an edge between two given vertices

**Input:**

Enter the number of vertices: 4

Enter the number of edges: 5

Enter edges (u v format):

0 1

0 2

1 2

1 3

2 3

Check edge existence between two vertices:

Enter vertices (u v): 0 3

**Requirements**:

* Use appropriate data structures (vector for adjacency lists).
* Ensure the program works for both directed and undirected graphs (prompt the user for the graph type).
* Validate inputs (e.g., vertices and edges must be valid and within the graph's range).

**Question 2)**

Given an **undirected graph** with **N vertices** and **E edges**, perform the following tasks using the Breadth-First Search (BFS) algorithm:

1. **Implement BFS** to traverse the graph starting from a user-specified source vertex.
2. **Display the order of visited vertices** during the traversal.
3. **Check graph connectivity**, i.e., determine if every vertex can be reached from any other vertex.
4. **Count the number of connected components** in the graph if it is not connected.
5. Generate and display the BFS spanning tree.

**Tasks:**

1. **Graph Input**:  
   Write a function to input the graph as an adjacency list. The function should take the number of vertices N and edges E as input, followed by the edge list.
2. **BFS Traversal**:  
   Implement a function void BFS(int start, vector<bool> &visited) to perform Breadth-First Search starting from a given node. Use a queue for the traversal.
3. **Check Connectivity**:  
   Use BFS to determine whether the graph is connected. Extend the BFS function or write a separate function for connectivity analysis.
4. **Count Connected Components**:  
   Extend your solution to count the number of connected components in the graph if it is not connected.
5. **Print BFS Spanning Tree:** Implement functionality to generate and display the BFS spanning tree as a parent-child relationship.

**Requirements:**

* Use an **adjacency list** representation for the graph.
* Use appropriate data structures:
  + **Queue** for BFS.
  + **Vector** for visited status.
* Validate input to ensure:
  + Vertices are within range.
  + Edges are valid.
* Ensure the BFS spanning tree is printed in a readable format, indicating parent-child relationships during traversal.