# CC-213L

# Data Structures and Algorithms

## Laboratory 12

## Graphs and Sorting

**Version: 1.0.0**

**Release Date: 02-01-2025**

**Department of Information Technology**

**University of the Punjab**

**Lahore, Pakistan**

## Contents:



## Learning Objectives:

- Pointers and Dynamic Memory Allocation
- Non-Linear Data Structure
- Graph Data Structure

## Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

## General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, not even allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the following.

| **Teachers:** | | |
|---|---|---|
| Course Instructor | Prof. Dr. Syed Waqar ul Qounain | swjaffry@pucit.edu.pk |
| Teacher Assistants | Tahir Mustafvi | bitf20m018@pucit.edu.pk |
| | Maryam Rasool | bcsf21m055@pucit.edu.pk |

# Background and Overview

## Non-Linear Data structures

Non-linear data structures are data structures in which elements are not arranged in a sequential, linear manner. Unlike linear data structures (e.g., arrays, linked lists) where elements are stored in a linear order, non-linear data structures allow for more complex relationships among elements

## Graphs

A graph is a data structure that consists of a set of nodes (or vertices) and a set of edges connecting these nodes. Graphs are widely used to represent relationships and connections between different entities. The nodes in a graph can represent entities (such as people, cities, or web pages), and the edges represent relationships or connections between these entities.

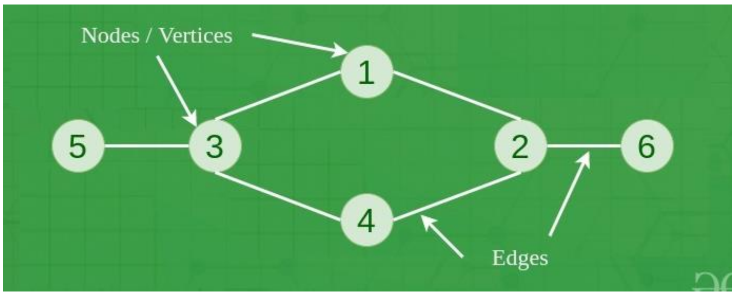



Figure 1(Graph)

## Representation of Graphs

### Adjacency Matrix:

In an adjacency matrix, a graph with n vertices is represented by an n x n matrix. The entry matrix[i][j] indicates whether there is an edge between vertices i and j. For an undirected graph, the matrix is symmetric.

### Example:

Consider the following undirected graph:

```
  A
 / \
B --- C
```

The adjacency matrix for this graph would be:

| | A | B | C |
|---|---|---|---|
| A | 0 | 1 | 1 |
| B | 1 | 0 | 1 |
| C | 1 | 1 | 0 |

**In this matrix:**

- The entry matrix [0][1] (or matrix [1][0]) is 1, indicating there is an edge between vertex A and vertex B.

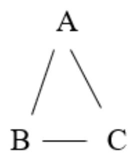
- The entry matrix [0][2] (or matrix [2][0]) is 1, indicating there is an edge between vertex A and vertex C.
- The entry matrix [1][2] (or matrix [2][1]) is 1, indicating there is an edge between vertex B and vertex C.

**Adjacency List:**

In an adjacency list representation, each vertex has a list of its neighboring vertices. This is an array of lists (or a HashMap), where each index represents a vertex, and the corresponding list contains the vertices adjacent to that vertex.

**Example:**

Using the same example graph:




The adjacency list for this graph would be:

A: [B, C]

B: [A, C]

C: [A, B]

**In this list:**

Vertex A is adjacent to vertices B and C.

Vertex B is adjacent to vertices A and C.

Vertex C is adjacent to vertices A and B.

## Comparison:

**1. Space Complexity:**

**Adjacency Matrix:** Takes O(V^2) space for V vertices. It's more space-efficient for dense graphs.

**Adjacency List**: Takes O (V + E) space for V vertices and E edges. It's more space-efficient for sparse graphs.

**2. Edge Existence Check:**

**Adjacency Matrix:** Checking if an edge exists takes constant time (O (1)).

**Adjacency List:** Checking if an edge exists may take time proportional to the degree of the vertex (O(degree)).

**3. Traversal:**

**Adjacency Matrix:** Traversing all neighbors of a vertex takes O(V) time.

**Adjacency List:** Traversing all neighbors of a vertex takes O(degree) time, which is generally faster for sparse graphs.

# Activities

## Pre-Lab Activities:

### Task 01: Dijkstra's Algo

**Objective:**
Design and implement a program to find the shortest path from a given source node to all other nodes in a weighted, directed graph using Dijkstra's algorithm.

**Problem Description:**
You are tasked with implementing a shortest path algorithm for a transportation network represented as a weighted, directed graph. The goal is to determine the minimum distance from a given source node S to every other node in the network.

The graph consists of 'n' nodes and 'm' edges, where each edge has a non-negative weight representing the cost to traverse it. Your program should output the shortest distances in ascending order of node numbers. If a node is unreachable from the source, indicate it by outputting "INF" for that node.

To achieve this, use **Dijkstra's algorithm**, which efficiently finds the shortest paths using a greedy approach. The algorithm relies on a priority queue (min-heap) to repeatedly select the node with the smallest tentative distance, updating the distances of its neighbours.

**Example:**

**Adjacency List Representation:**

1: [(2, 4), (3, 2)]

2: [(3, 5), (4, 10)]

3: [(5, 3), (6, 9)]

4: [(6, 11)]

5: [(4, 4), (6, 6)]

6: []

**Expected Output:**

**0 4 2 9 5 11**

**Explanation:**

- Distance to Node 1 (source): 0
- Distance to Node 2: 4 (via 1→2)
- Distance to Node 3: 2 (via 1→3)
- Distance to Node 4: 9 (via 1→3→5→4)
- Distance to Node 5: 5 (via 1→3→5)
- Distance to Node 6: 11 (via 1→3→5→6)