CC-213L

Data Structures and Algorithms

Laboratory 11

Graphs and AVL

Version: 1.0.0

Release Date: 19-12-2024

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.

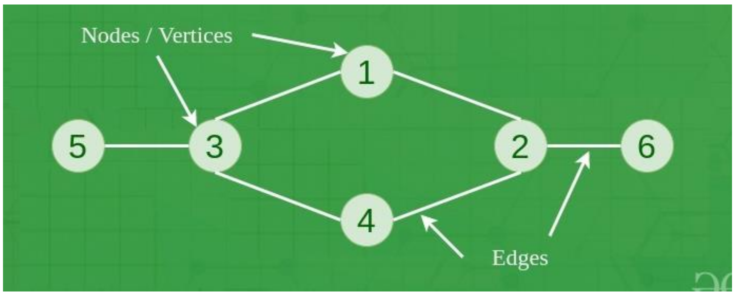


Figure 1(Graph)

There are two main types of graphs: directed and undirected.

1. Undirected Graph:

In an undirected graph, edges have no direction. If there is an edge between node A and node B, you can travel from A to B or from B to A along that edge. Mathematically, an undirected edge between nodes A and B can be represented as {A, B} or (A, B).

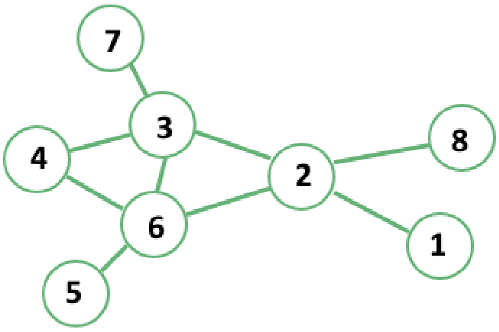


Figure 2(Undirected Graph)

2. Directed Graph (Digraph):

In a directed graph, edges have direction. If there is a directed edge from node A to node B, it means you can travel from A to B, but not necessarily from B to A. Mathematically, a directed edge from node A to node B can be represented as (A -> B).

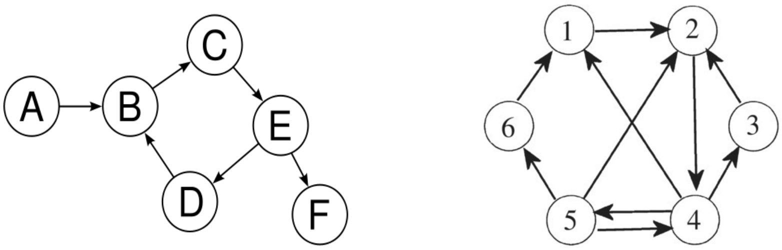


Figure 3(Directed Graphs)

Graphs can also have weighted edges, where each edge has an associated numerical value called a weight.

Common operations on graphs include adding or removing nodes and edges, traversing the graph to visit nodes in a specific order, and finding paths or cycles within the graph. Graphs can be classified into various types based on their properties and structures, such as:

3. Dense Graphs:

Definition: Dense graphs are graphs with many edges relative to the number of nodes.

Characteristics: In dense graphs, most pairs of nodes are connected by edges. The number of edges is close to the maximum possible for the given number of nodes.

Example: Complete graphs (where every pair of distinct nodes is connected by an edge) are an extreme example of dense graphs.

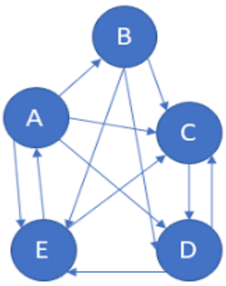


Figure 4(Dense Graph)

4. Sparse Graphs:

Definition: Sparse graphs are graphs with relatively few edges compared to the number of nodes.

Characteristics: In sparse graphs, only a small fraction of possible edges is present. The number of edges is much less than the maximum possible for the given number of nodes.

Example: Trees and forests are often sparse graphs.

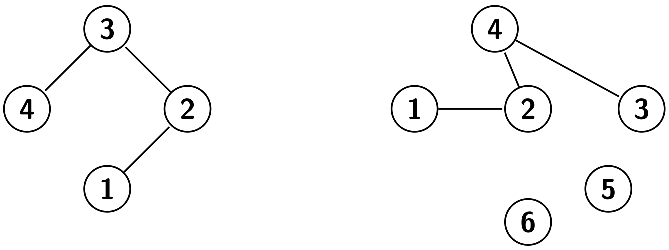


Figure 5(Sparse Graphs)

5. Cyclic Graphs:

Definition: Cyclic graphs are graphs that contain cycles, which are closed paths in the graph.

Characteristics: A cycle is a sequence of nodes where the first and last nodes are the same, and the edges form a closed loop.

Example: A simple example of a cyclic graph is a triangle (three nodes connected with edges in the shape of a triangle).

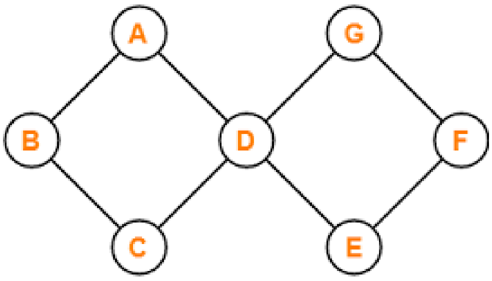


Figure 6(Cyclic Graph)

6. Acyclic Graphs:

Definition: Acyclic graphs are graphs that do not contain cycles.

Characteristics: There are no closed paths in acyclic graphs. These graphs have a natural hierarchy and are often used to represent relationships without circular dependencies.

Example: Trees are a common example of acyclic graphs. Directed acyclic graphs (DAGs) are often used in applications like task scheduling and dependency management.

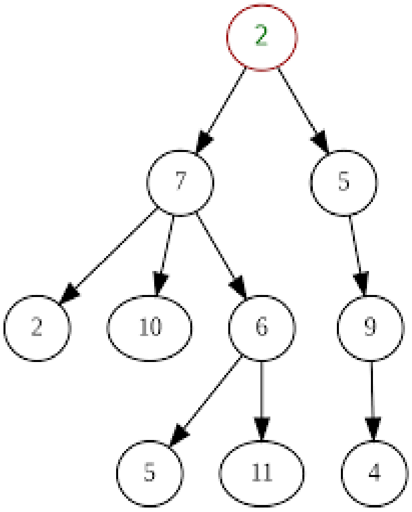


Figure 7(Acyclic Graph)

7. Connected Graphs:

Definition: Connected graphs are graphs where there is a path between every pair of nodes.

Characteristics: It is possible to travel from any node to any other node in the graph by following edges.

Example: A connected graph could be a fully connected graph where every node is directly or indirectly connected to every other node.

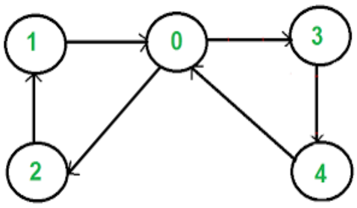


Figure 8(Connected Graphs)

8. Disconnected Graphs:

Definition: Disconnected graphs are graphs with components that are not connected to each other.

Characteristics: There are at least two nodes or sets of nodes in the graph that are not connected by any path.

Example: A graph consisting of two disconnected subgraphs (two separate groups of nodes) is an example of a.

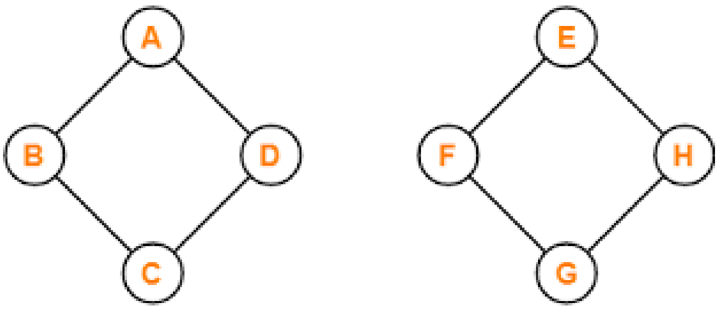


Figure 9(Disconnected Graphs)

Applications of Graphs:

Graphs have a wide range of applications across various domains due to their ability to model and represent relationships and connections between entities. Here are some common applications of graphs:

1. Social Network Analysis:

Graphs are used to model and analyze social networks, representing individuals as nodes and relationships as edges. This is crucial for understanding social structures, influence patterns, and information flow in networks like Facebook, Twitter, and LinkedIn.

2. Network Routing and Design:

Graphs play a key role in the design and optimization of computer networks. Nodes represent routers or computers, and edges represent communication links. Routing algorithms use graph theory concepts to find the most efficient paths for data transmission.

3. Recommendation Systems:

Graphs are employed in recommendation systems to model user-item interactions. Nodes represent users and items, and edges represent interactions or preferences. Algorithms analyze these graphs to suggest items that a user might be interested in based on their past behavior or preferences.

4. Web Page Ranking (PageRank):

Google's PageRank algorithm uses graph theory to rank web pages based on their importance. Web pages are represented as nodes, and hyperlinks between pages are represented as edges. Pages with more inbound links from important pages are considered more relevant and receive higher rankings in search results.

5. Transportation Networks:

Graphs model transportation systems such as road networks, subway systems, and airline routes. Nodes represent locations, and edges represent connections or routes. Algorithms can optimize travel routes, analyse traffic flow, and plan efficient transportation systems.

6. Epidemiology and Disease Spread Modelling:

Graphs are used to model the spread of diseases in populations. Nodes represent individuals, and edges represent possible paths of transmission. Analyzing the graph can help predict and control the spread of diseases.

7. Dependency Resolution and Task Scheduling:

Directed acyclic graphs (DAGs) are used to model dependencies between tasks in project management, software builds, and other scheduling applications. Nodes represent tasks, and edges represent dependencies. Efficient scheduling algorithms help optimize task execution.

8. Circuit Design:

Electrical circuits can be modeled as graphs, where components are nodes and connections between components are edges. Graph algorithms are used to analyze and optimize circuit design.

9. Recommendation Systems:

Graphs are utilized in recommendation systems to model user-item interactions. Nodes represent users and items, while edges represent interactions or preferences. Algorithms analyze these graphs to suggest items that a user might be interested in based on their past behavior or preferences.

10. Biological Networks:

Graphs are employed in bioinformatics to model biological interactions, such as protein-protein interactions, gene regulatory networks, and metabolic pathways. Nodes represent biological entities, and edges represent interactions or relationships.

11. Game Theory:

Graphs are used to model strategic interactions and dependencies in game theory. Nodes represent players, and edges represent relationships or interactions between players. This is applicable in areas like economics and political science.

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:

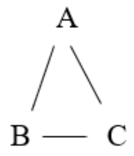
- The entry matrix [0][1] (or matrix [1][0]) is 1, indicating there is an edge between vertex A and vertex B.
- The entry matrix [0][2] (or matrix [2][0]) is 1, indicating there is an edge between vertex A and vertex C.
- The entry matrix [1][2] (or matrix [2][1]) is 1, indicating there is an edge between 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: Adjacency Matrix representation of the Graph

Given declaration of the class Graph. Add necessary variables in the class and implement all the given methods of the class.

```
class Graph {
        private:
            int **adjMat;    // Adjacency matrix
            int vertex;   // Number of vertices
            int edge;    // Number of edges
        public:
            Graph();    // Constructor
            bool isEmpty();
            void insertEdge(int u, int v);
            void deleteEdge(int u, int v);
};
```

→ `You can also make extra helper functions, to visualize the graph!`