National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Fall 2022, Lab Manual – 13

Course Code: CL-2001	**Course : Data Structures - Lab**
Instructor(s) :	**Abeer Gauher, Sobia Iftikhar**

LAB - 13

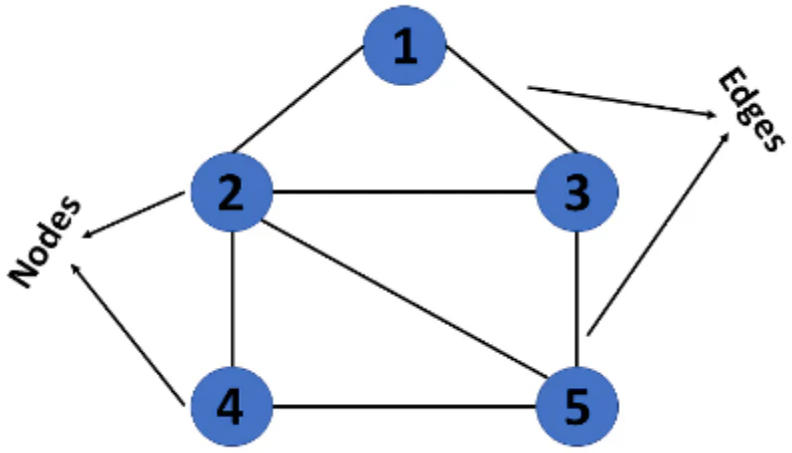
Graphs and it's applications

Introduction

- Graphs in data structures are non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them.
- Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks, and social networks.

What Are Graphs in Data Structure?

- A graph is a non-linear kind of data structure made up of nodes or vertices and edges.
- The edges connect any two nodes in the graph, and the nodes are also known as vertices.



This graph has a set of vertices V= { 1,2,3,4,5} and a set of edges E= { (1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(4,50 }.

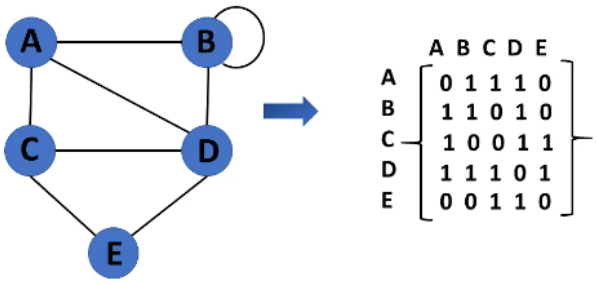
Representation of Graphs in Data Structures
The most frequent graph representations are the two that follow:

1. Adjacency matrix
2. Adjacency list

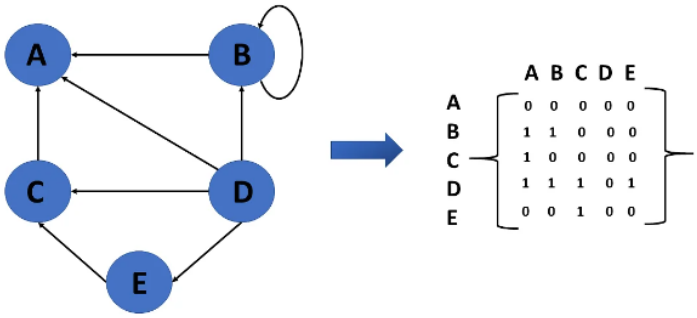
Adjacency Matrix

It's used to show which nodes are next to one another. I.e., is there any connection between nodes in a graph? If an edge exists between vertex a and vertex b, the corresponding element of G, gi,j = 1, otherwise gi,j = 0.

Undirected Graph Representation



Directed Graph Representation

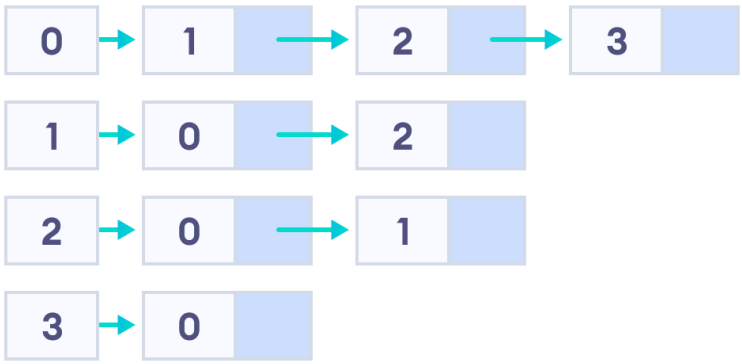


Adjacency List

A linked representation is an adjacency list.

You keep a list of neighbors for each vertex in the graph in this representation. It means that each vertex in the graph has a list of its neighboring vertices.

You have an array of vertices indexed by the vertex number, and the corresponding array member for each vertex x points to a singly linked list of x's neighbors.



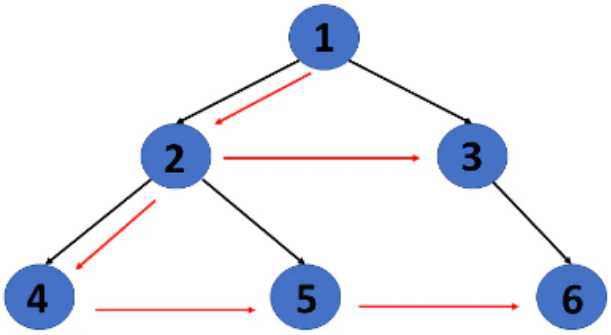
Graph Traversal Algorithm

There are two techniques to implement a graph traversal algorithm:

1. Breadth-first search
2. Depth-first search

Breadth-First Search or BFS

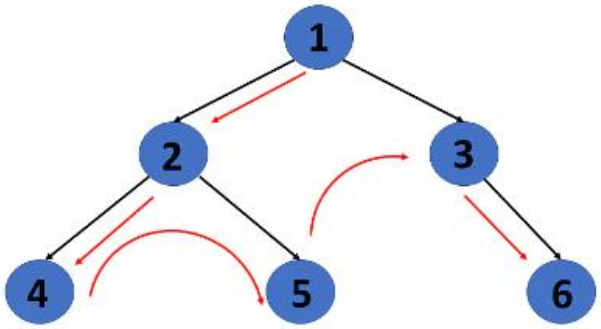
It begins at the root of the graph and investigates all nodes at the current depth level before moving on to nodes at the next depth level.
To maintain track of the child nodes that have been encountered but not yet inspected, more memory, generally you require a queue.



BFS	1	2	3	4	5	6

Depth-First Search or DFS

The DFS algorithm begins at the root node and examines each branch as far as feasible before backtracking.
To maintain track of the child nodes that have been encountered but not yet inspected, more memory, generally a stack, is required.

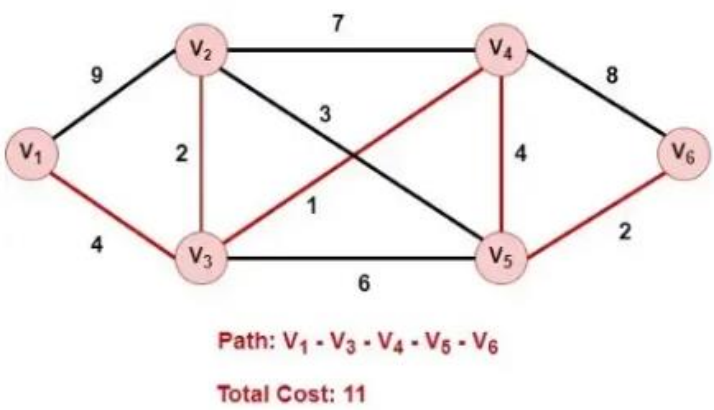


DFS	1	2	4	5	3	6

Graph Shortest Path

Dijkstra's Algorithm
Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.

It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.



	Distance	Previous Node	Visited
V1	0	None	True
V2	6	V3	True
V3	4	V1	True
V4	5	V3	True
V5	9	V4	True
V6	11	V5	True

Pseudocode:

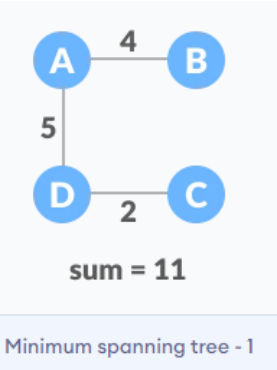
```
function Dijkstra(Graph, source, target):
    for each vertex v in Graph:
        define the visited state FALSE
        define the cost from the source node to infinity
        define the previous node to None
        if v == source:
          define the cost from the source node to zero

  while target is not visited:
      selected_node ← vertex in Graph with min distance from
                      the source node

      selected_node is characterized as visited

      for each neighbor v of selected_node not visited:
        alt ← selected_node distance from source node +
                length(selected_node, v)
        if alt < neighbor distance from source node:
          neighbor distance from source node ← alt
          neighbor previous node ← selected_node
    calculate the path
return path, target node distance from start
```

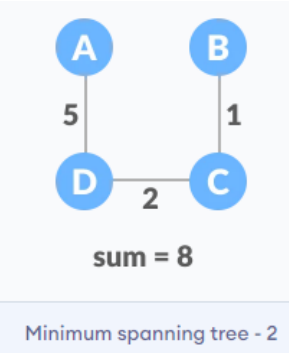
Spanning Trees

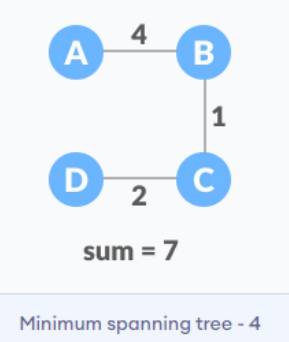
A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.

Minimum Spanning Tree

A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.



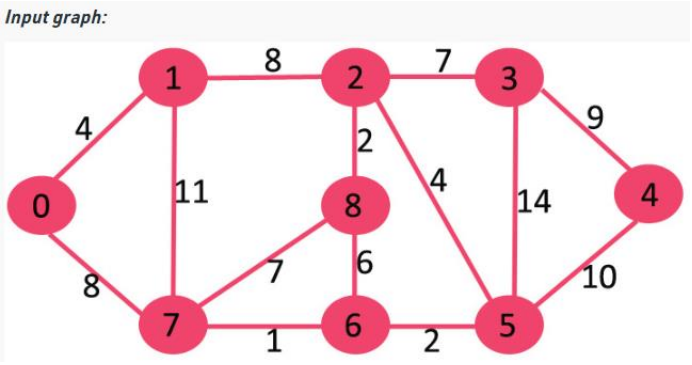




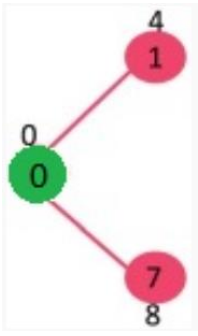
Prim's Algorithm

Prim's approach identifies the subset of edges that includes every vertex in the graph, and allows the sum of the edge weights to be minimized.

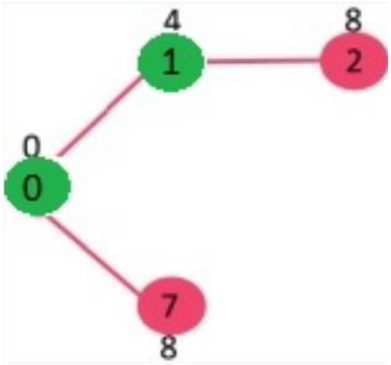
Prim's algorithm starts with a single node and works its way through several adjacent nodes, exploring all of the connected edges along the way. Edges with the minimum weights that do not cause cycles in the graph get selected for t inclusion in the MST structure.



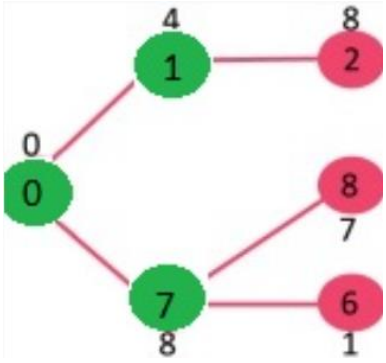
Step 1:



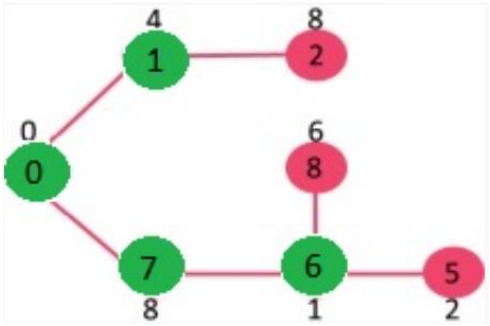
Step 2:



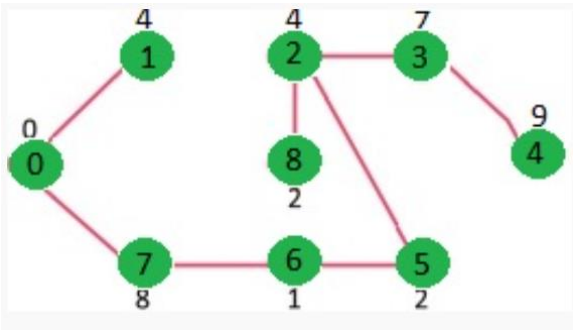
Step 3:



Step 4:



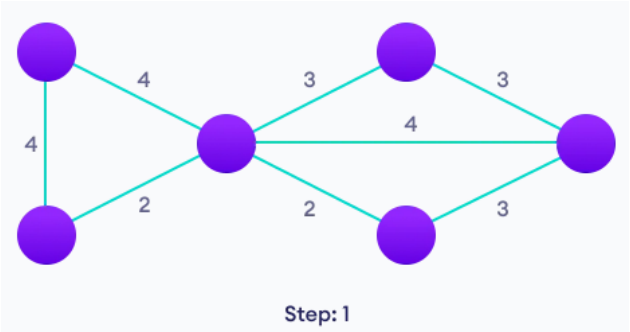
Step 5:

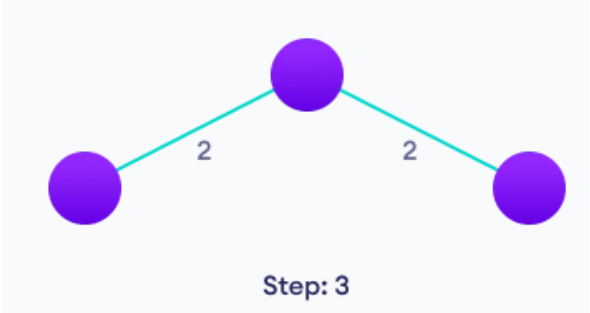


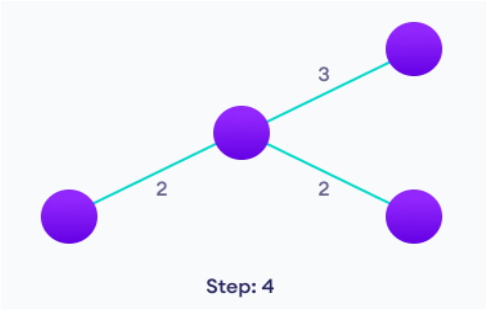
Kruskal's Algorithm

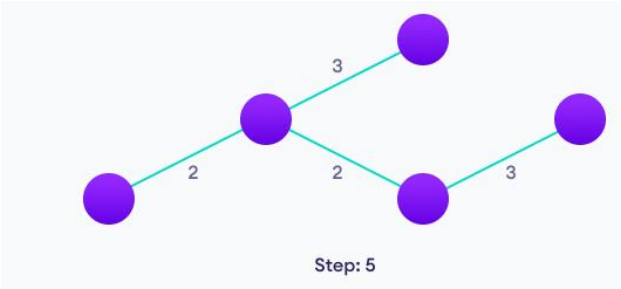
The steps for implementing Kruskal's algorithm are as follows:

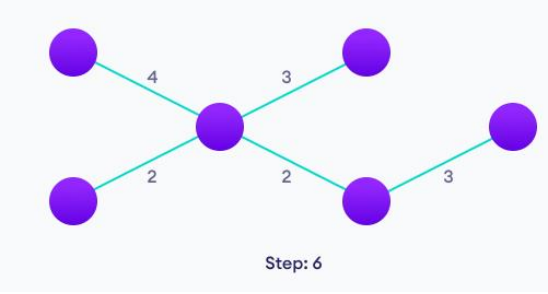
- Sort all the edges from low weight to high
- Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
- Keep adding edges until we reach all vertices.











Topological Sorting

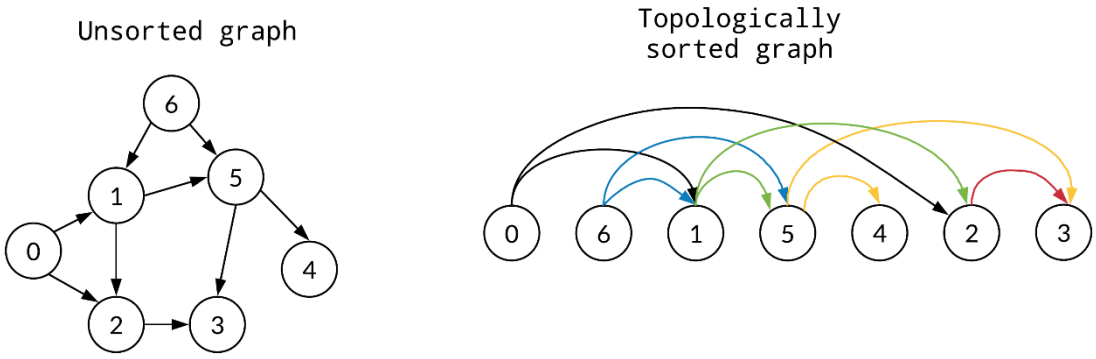
Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering.

Algorithm for Topological Sorting:

Prerequisite: DFS

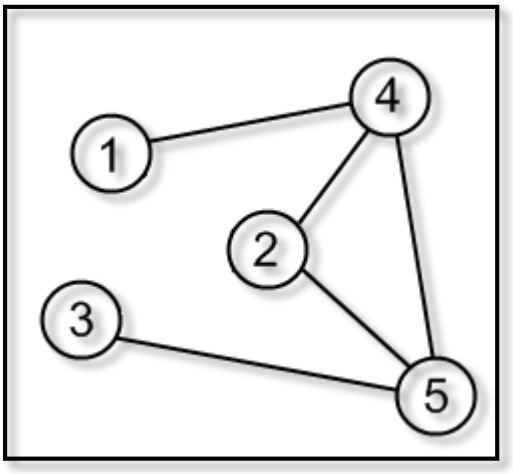
In topological sorting,

- We use a temporary stack.
- We don't print the vertex immediately,
- We first recursively call topological sorting for all its adjacent vertices, then push it to a stack.
- Finally, print the contents of the stack.



Lab Tasks:

Task 1:

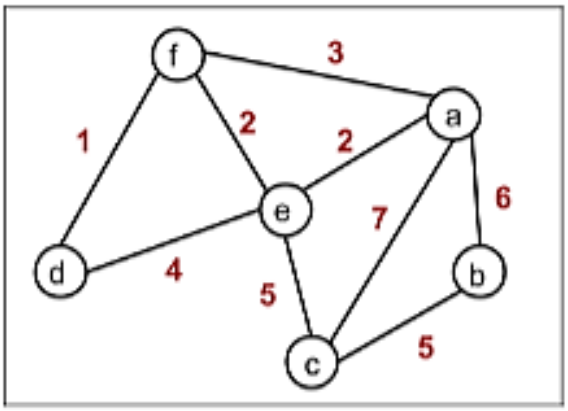


For the following graph implement the following:

- Create an Adjacency List and Adjacency Matrix. Display the list and matrix.
- Traverse the graph using BFS and DFS.
- Display the output for both the traversals.

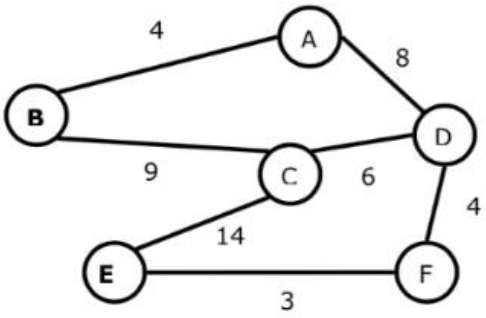
Task 2:

For the following graph, find the minimum spanning tree using Prim's and Kruskal's **(BOTH)** algorithm. Start with vertex A. Show your work at each set and indicate the order in which edges are added to the MST.



Task 3:

Use Dijkstra's algorithm to find the shortest path from node B to node E in the graph below.



Task 4:

Show the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the figure given below.