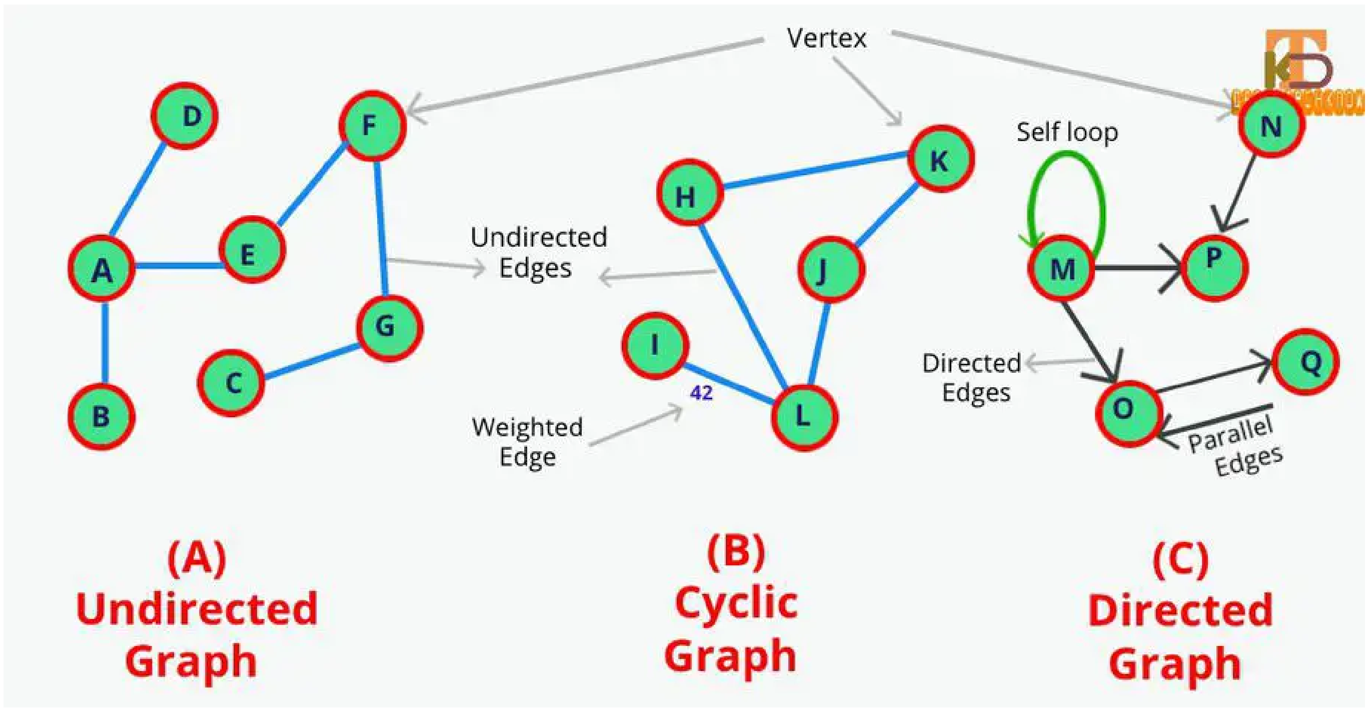
Sessions 13 & 14: Graphs & Applications

Graph theory is a branch of mathematics that deals with the study of graphs, which are mathematical structures used to represent relationships between objects. It has applications in various fields such as computer science, network analysis, social sciences, operations research, and more. Graph theory was pioneered by the Swiss mathematician Leonhard Euler in the 18th century with the famous problem known as the Seven Bridges of Königsberg.

A graph consists of two main components: vertices (also called nodes) and edges (also called links or arcs). Vertices represent the objects or entities, and edges represent the connections or relationships between those entities. The connections can be directed (pointing from one vertex to another) or undirected (bidirectional).



Here are some fundamental concepts in graph theory:

- **Vertex (Node):** Each object or entity is represented by a vertex in the graph. For example, in a social network, each person can be a vertex.
- **Edge (Link):** An edge is a connection between two vertices. It indicates a relationship or interaction between the corresponding entities. Edges can have weights or values associated with them, representing some quantitative measure of the relationship.
- **Directed and Undirected Graphs:** In an undirected graph, edges have no direction, meaning they represent symmetric relationships. In a directed graph, each edge has a specific direction from one vertex to another, indicating an asymmetric relationship.
- **Degree of a Vertex:** The degree of a vertex is the number of edges incident to it. In an undirected graph, it represents the number of connections a node has, while in a directed graph, it is split into in-degree (incoming edges) and out-degree (outgoing edges).
- **Path:** A path in a graph is a sequence of vertices connected by edges, indicating a route or a way to move from one vertex to another.
- **Cycle:** A cycle is a path that starts and ends at the same vertex, forming a closed loop.
- **Connected Graph:** A graph is said to be connected if there is a path between any two vertices in the graph. Otherwise, it is disconnected.

- **Weighted Graph:** A graph with numerical values (weights) assigned to its edges. Weighted graphs are commonly used to model situations where the edges represent distances, costs, time, etc.
- **Tree:** A tree is a connected acyclic graph, meaning it has no cycles. Trees have specific applications, such as in hierarchical data structures and searching algorithms.

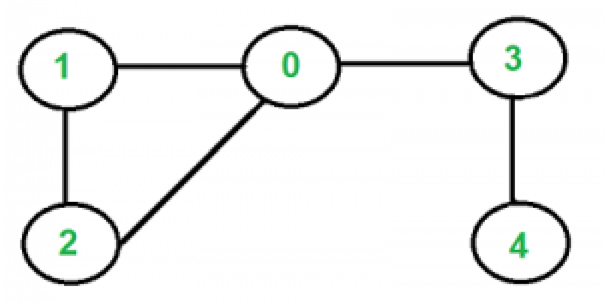
Graph theory provides a rich set of tools and algorithms for analyzing and solving problems related to networks, connectivity, optimization, and much more. Some of the essential algorithms in graph theory include Depth-First Search (DFS), Breadth-First Search (BFS), Dijkstra's algorithm, and Minimum Spanning Tree algorithms (e.g., Prim's and Kruskal's algorithms). These algorithms find applications in route planning, network routing, shortest path problems, and many other real-world scenarios.

Types of Graph

Graph theory encompasses various types of graphs, each with its unique characteristics and applications. Here are some of the most common types of graphs:

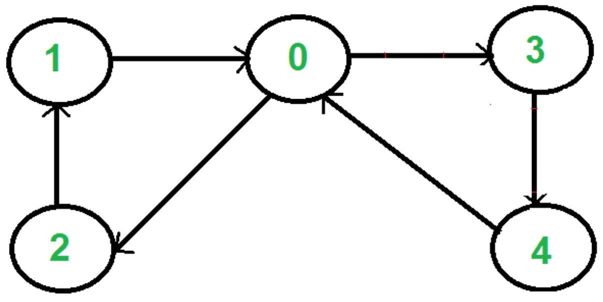
Undirected Graph:

- An undirected graph consists of vertices connected by undirected edges.
- Edges have no direction and represent symmetric relationships between vertices.
- Example: Social networks, where individuals are represented by vertices, and friendships are represented by undirected edges.



Directed Graph (Digraph):

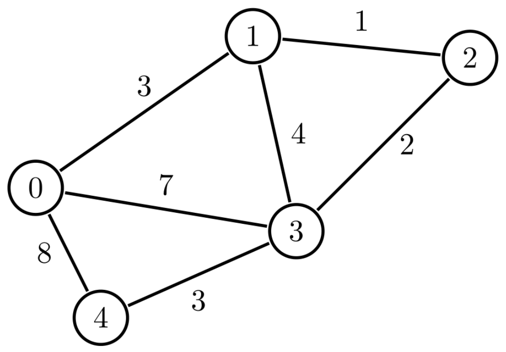
- In a directed graph, edges have a specific direction, indicating asymmetric relationships.
- Edges are represented by arrows from one vertex to another.
- Example: Web pages connected by hyperlinks, where the links have a clear direction from one page to another.



Weighted Graph:

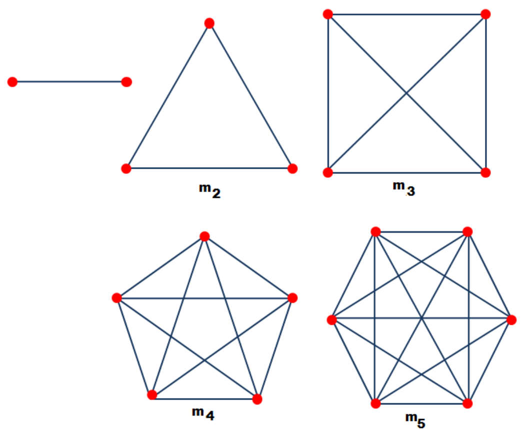
- In a weighted graph, edges have numerical values (weights) associated with them.

- Weights can represent distances, costs, time, or any other quantitative measure of the relationship between vertices.
- Example: A transportation network, where edges have weights representing the distances between cities.



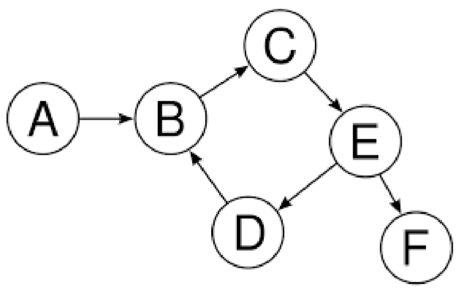
Complete Graph:

- In a complete graph, each vertex is directly connected to every other vertex by an edge.
- It has the maximum number of possible edges for its number of vertices.
- Example: A group of people, where each person is friends with every other person.



Cycle Graph:

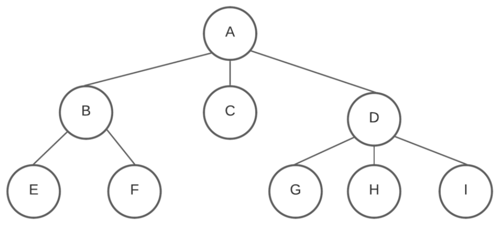
- A cycle graph is a closed loop where each vertex is connected to its adjacent vertices, forming a cycle.
- It has no branches or dead ends.
- Example: A circular road where each intersection is connected to the next one.



Tree:

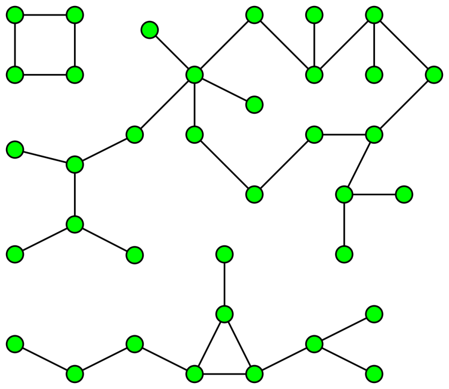
- A tree is a connected acyclic graph, meaning it has no cycles.

- It has one vertex, called the root, which has no incoming edges, and all other vertices have exactly one incoming edge.
- Trees have applications in hierarchical data structures and searching algorithms.
- Example: A family tree, where each node represents an individual, and the edges represent parent-child relationships.



Forest:

- A forest is a collection of disjoint trees, i.e., a set of trees without any common vertices or edges.
- Example: Multiple family trees representing different families.



These are just some of the many types of graphs in graph theory, each offering unique insights and modelling capabilities for various real-world problems and applications.

Representation of Graphs

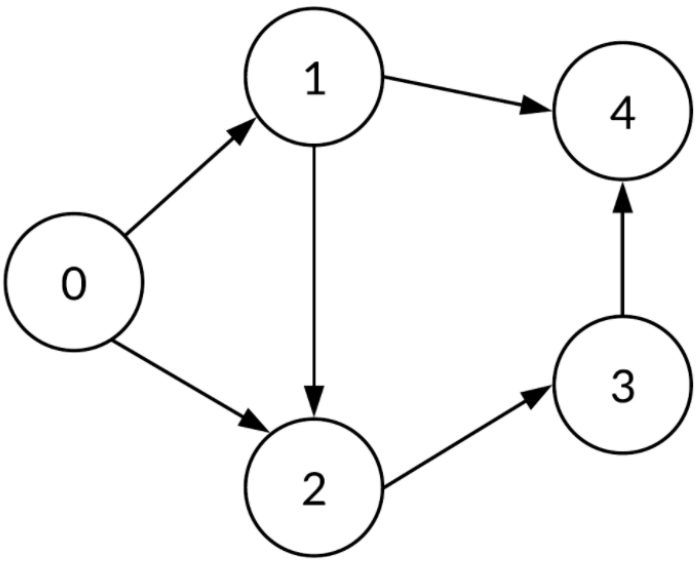
Graphs can be represented using two popular data structures: Adjacency Matrix and Adjacency List. Additionally, two common graph traversal algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS).

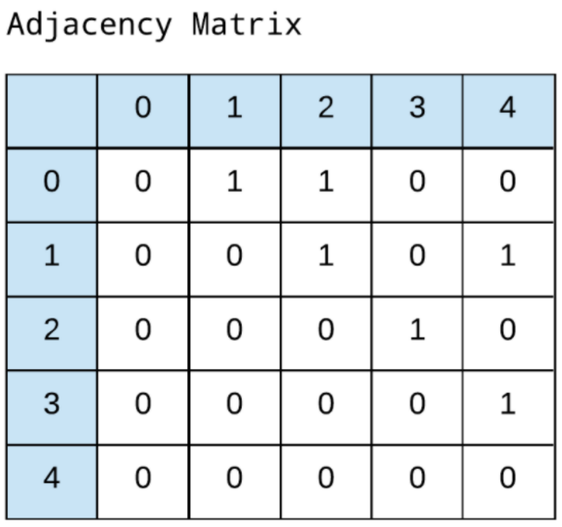
Adjacency Matrix

An adjacency matrix is a 2D array (or matrix) that represents the connections between vertices in a graph. For an undirected graph, the matrix is symmetric, while for a directed graph, it might not be symmetric.

In an adjacency matrix:

- Rows and columns represent vertices.
- The entry at matrix[i][j] is 1 if there is an edge between vertex i and vertex j, and 0 otherwise (for an unweighted graph).
- For a weighted graph, the entry at matrix[i][j] represents the weight of the edge between vertex i and vertex j, or it can be a special value (e.g., infinity) if there is no edge.





Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Advantages:

- Checking for the existence of an edge between two vertices is efficient (O(1)).
- It is easy to implement and understand.

Disadvantages:

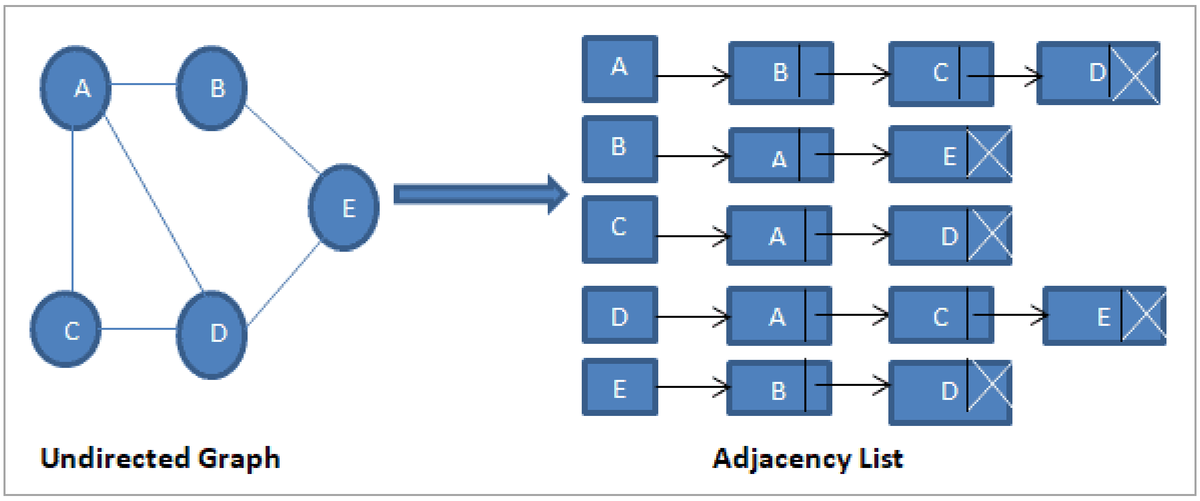
- It consumes more memory, especially for sparse graphs (graphs with few edges).
- Insertion and deletion of edges are less efficient (O(V^2)).

Adjacency List

An adjacency list is a collection of lists or arrays, where each list/array represents a vertex in the graph, and it contains its adjacent vertices.

In an adjacency list:

- Each vertex is associated with a list of its neighboring vertices.
- For an unweighted graph, the list may contain the actual vertices' identifiers.
- For a weighted graph, the list may contain pairs of (vertex, weight) to represent the edges and their weights.



Advantages:

- It efficiently represents sparse graphs (graphs with few edges) as it only uses space proportional to the number of edges.
- Insertion and deletion of edges are more efficient (O(1) or O(E)), where E is the number of edges.
- It is more memory-efficient for large graphs.

Disadvantages:

- Checking for the existence of an edge between two vertices can be less efficient (O(V)) as it requires searching through the list.
- It may consume more memory for dense graphs (graphs with many edges).

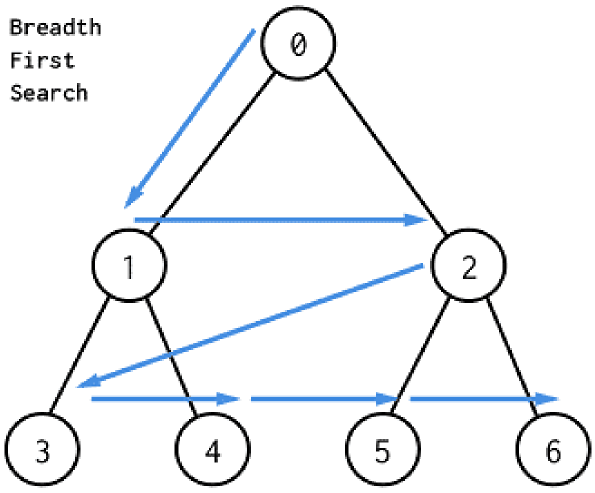
Graph Traversal Algorithms:

Breadth First Search

BFS is a graph traversal algorithm that visits all the vertices in a graph by exploring its neighbors level by level. It starts from a selected vertex (source) and explores all its neighbors before moving to their neighbors.

BFS Algorithm:

- Initialize a queue data structure and enqueue the source vertex.
- While the queue is not empty, dequeue a vertex and process it.
- Enqueue all its unvisited neighbors, mark them as visited, and continue until the queue is empty.



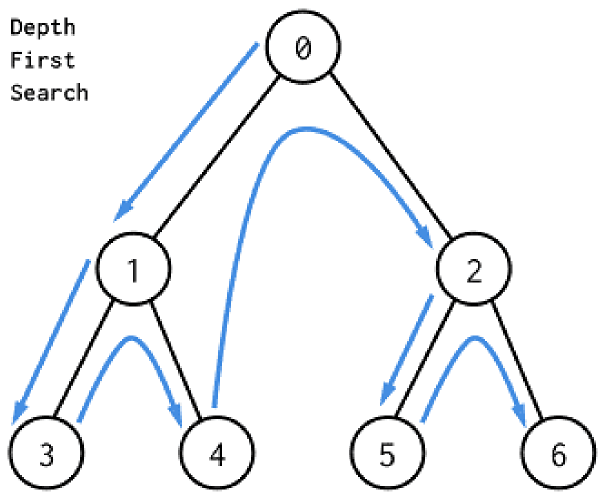
BFS ensures that all vertices at a certain distance (number of edges away) from the source vertex are visited before moving on to vertices farther away. It finds the shortest path in an unweighted graph.

Depth First Search

DFS is another graph traversal algorithm that explores as far as possible along each branch before backtracking. It starts from a selected vertex (source) and explores one branch as far as possible before backtracking.

DFS Algorithm (recursive version):

- Mark the source vertex as visited and process it.
- For each unvisited neighbor of the current vertex, recursively apply the DFS algorithm to it.



DFS explores deeper into the graph before visiting its neighbors. It is used for various applications like topological sorting, connected component analysis, and cycle detection.

Both BFS and DFS are essential algorithms in graph theory, and their choice depends on the specific requirements of the problem being solved.

Shortest Path

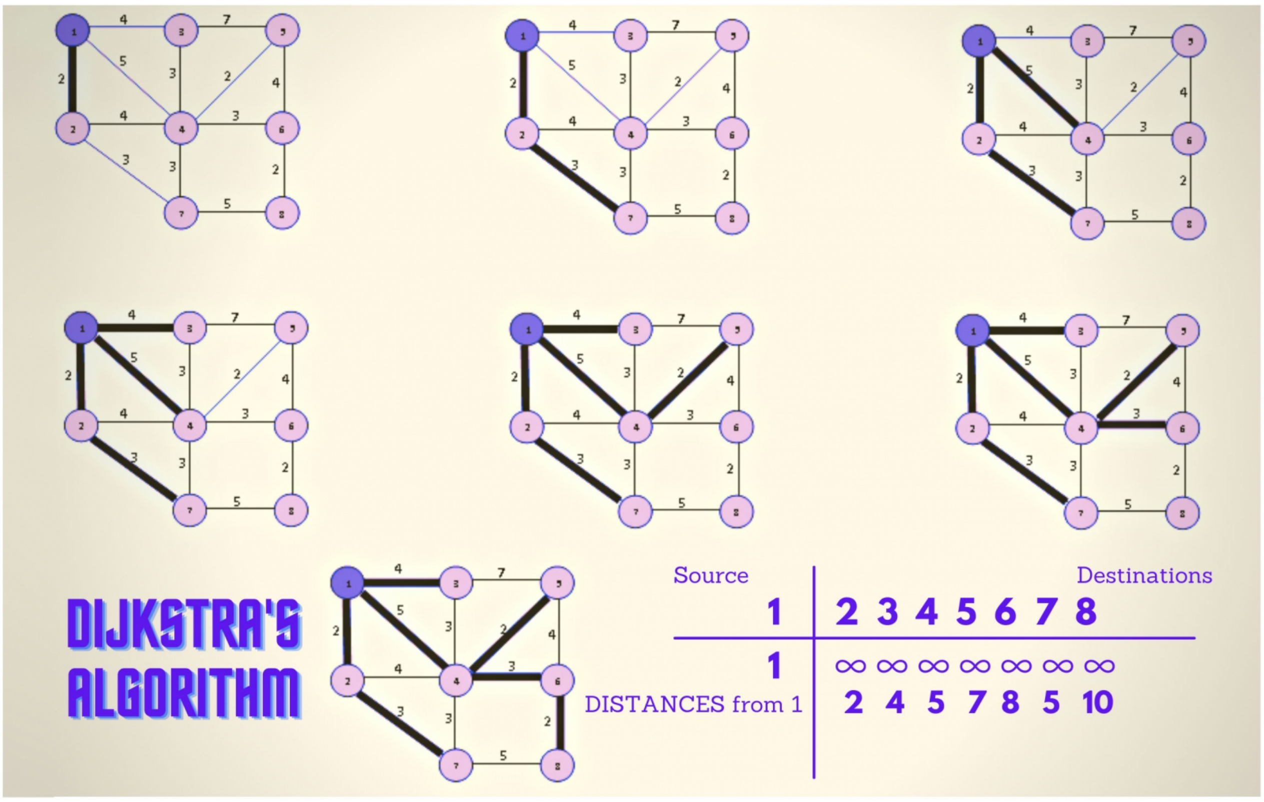
Shortest Path algorithms are used to find the shortest distance between two vertices in a graph. They are crucial for various applications like route planning, network routing, and optimization problems. Two commonly used shortest path algorithms are Dijkstra's algorithm for single-source shortest path and the Floyd-Warshall algorithm for all-pairs shortest path.

Dijkstra's Algorithm:

Dijkstra's algorithm is a greedy algorithm used to find the shortest path from a single source vertex to all other vertices in a graph with non-negative edge weights. It works well for sparse graphs and is commonly used in routing and navigation applications.

Algorithm steps:

- Initialize distances from the source vertex to all other vertices as infinity, except for the source itself, which is set to 0.
- Create a priority queue (min heap) to keep track of the next vertex to visit based on the current distance.
- Start with the source vertex and explore its neighbours.
- For each neighbour, update the distance from the source if a shorter path is found.
- Repeat step 3 and 4 until all vertices are visited.



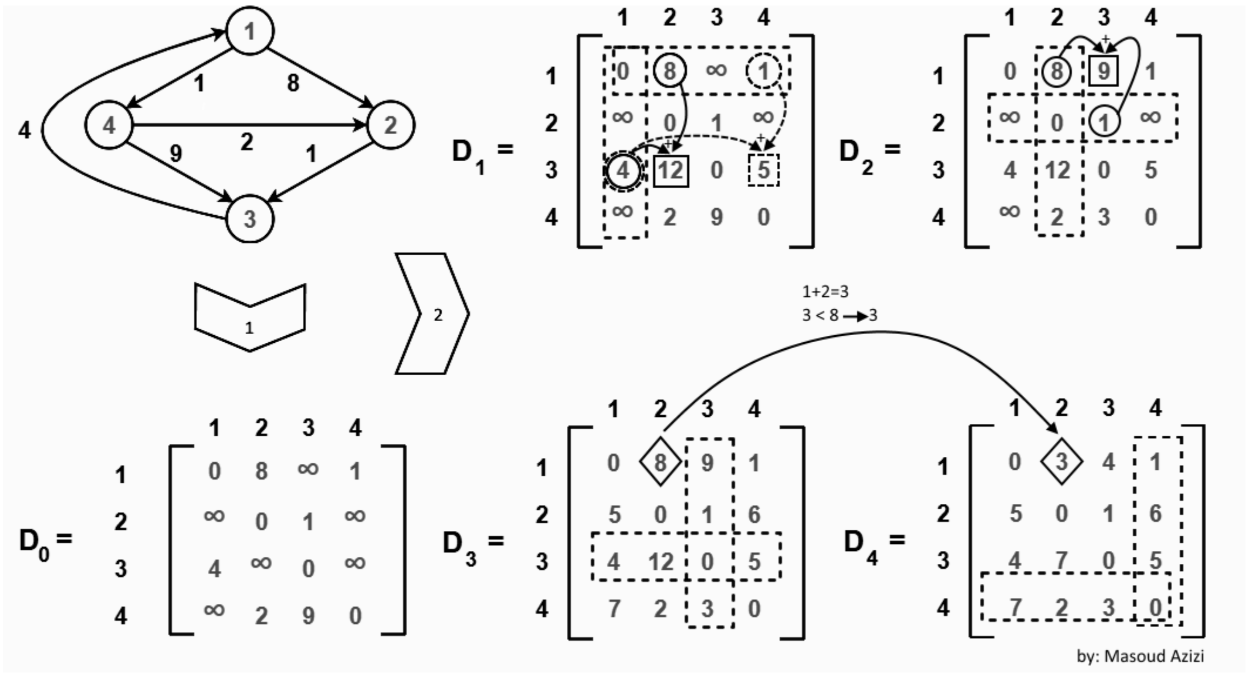
Source	Destinations						
1	2	3	4	5	6	7	8
1	∞	∞	∞	∞	∞	∞	∞
DISTANCES from 1	2	4	5	7	8	5	10

Floyd-Warshall Algorithm

The Floyd-Warshall algorithm is a dynamic programming algorithm used to find the shortest paths between all pairs of vertices in a graph, including graphs with negative edge weights. It is more efficient than running Dijkstra's algorithm for each source vertex separately when dealing with dense graphs.

Algorithm steps:

- Initialize a 2D matrix (let's call it dist) to represent the distances between all pairs of vertices. Set the diagonal elements (i.e., distance from a vertex to itself) to 0, and initialize all other elements to infinity.
- For each edge (u, v) with weight w, update the dist[u][v] to w.
- For each vertex k, consider all pairs of vertices (i, j), and if the path from i to k to j is shorter than the current distance dist[i][j], update dist[i][j] to the new shorter distance.
- After iterating through all vertices k, the dist matrix will contain the shortest path distances between all pairs of vertices.



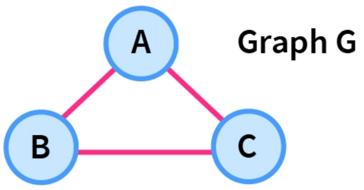
Comparison:

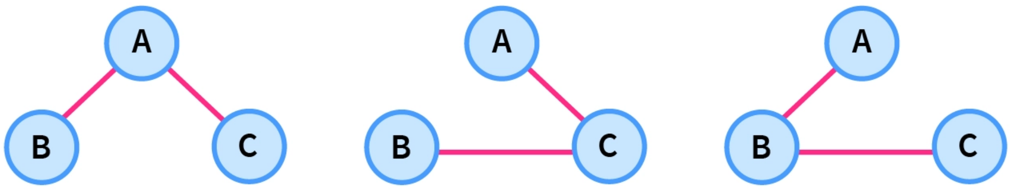
- Dijkstra's algorithm is used to find the shortest path from a single source vertex to all other vertices in a graph with non-negative edge weights. It is suitable for sparse graphs.

- The All-pairs shortest path and Floyd-Warshall algorithms find the shortest paths between all pairs of vertices, with the latter also handling negative edge weights. However, the Floyd-Warshall algorithm is more efficient for dense graphs.

Spanning Trees

Spanning trees are special subgraphs of a connected graph that include all the vertices of the original graph while having the minimum possible total edge weight. Minimum Spanning Trees (MSTs) are used to find the most cost-efficient way to connect all nodes in a weighted graph. Two well-known algorithms to find the MST are Prim's algorithm and Kruskal's algorithm.

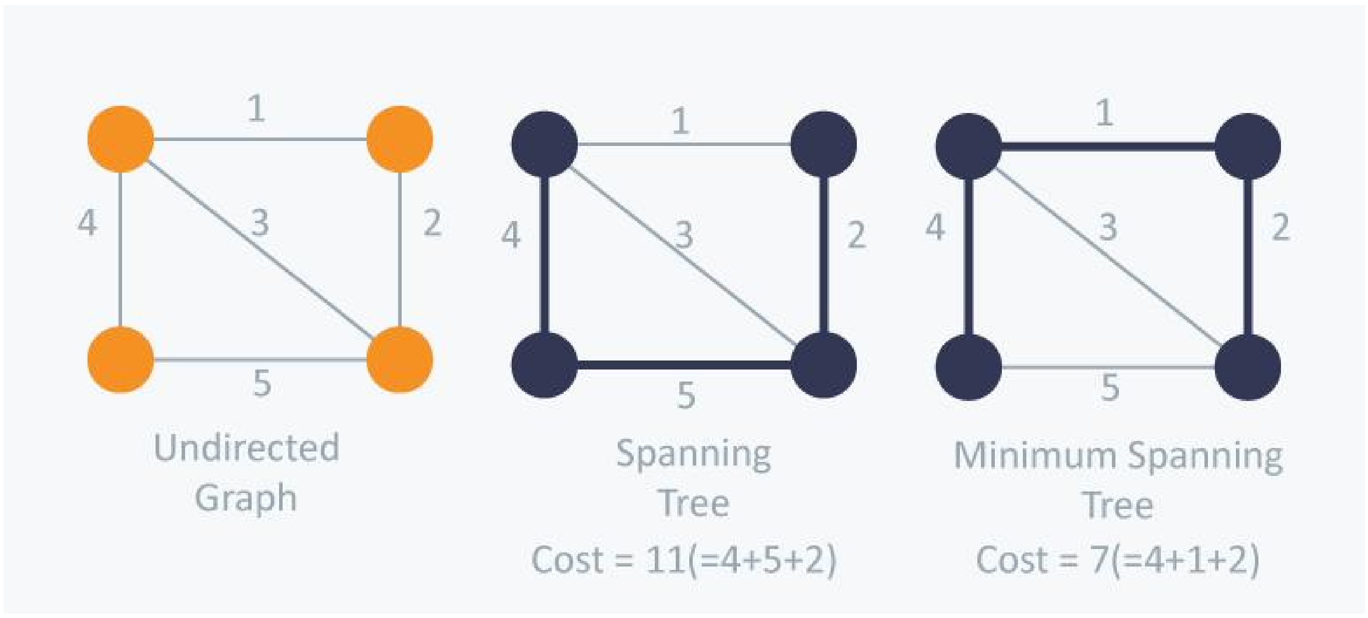




Spanning Trees, subgraph of G

Minimum Spanning Tree (MST):

A minimum spanning tree is a tree that connects all vertices in a connected, undirected graph with the minimum possible total edge weight. MSTs do not contain any cycles, and the number of edges in an MST is always one less than the number of vertices in the graph.

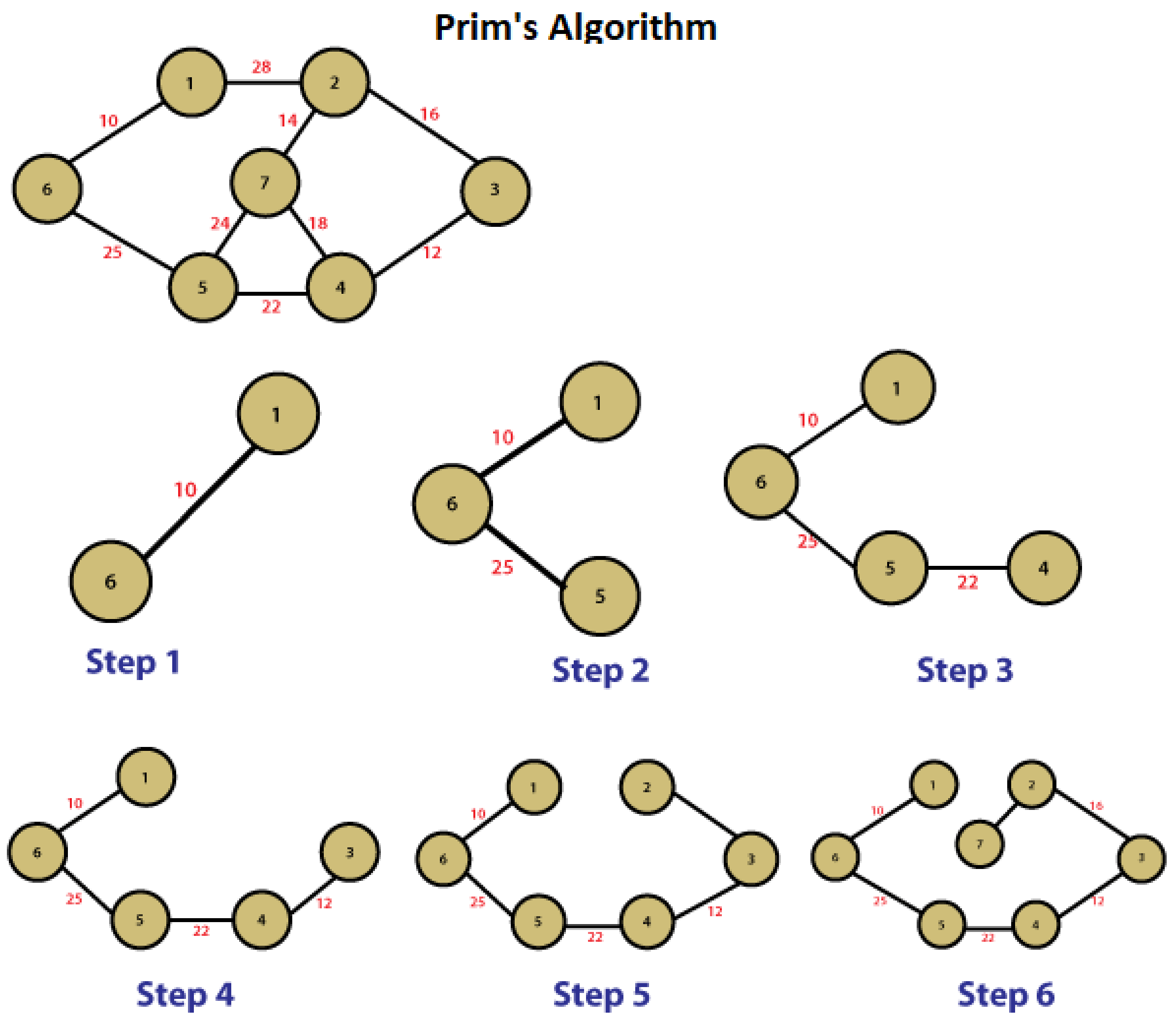


Prim's Algorithm:

Prim's algorithm is a greedy algorithm that finds the minimum spanning tree for a connected, undirected graph. It starts with an arbitrary node and repeatedly adds the cheapest edge that connects a vertex in the MST to a vertex outside of it until all vertices are included in the MST.

Algorithm steps:

- Start with an arbitrary node as the initial MST.
- Create a priority queue (min heap) containing all edges connected to the current MST.



While there are still vertices not included in the MST, do the following:

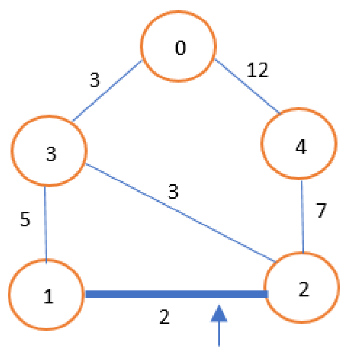
a. Extract the edge with the minimum weight from the priority queue.
b. If one of the edge's vertices is not in the MST, add the edge to the MST and add all edges connected to the newly added vertex to the priority queue.

Kruskal's Algorithm:

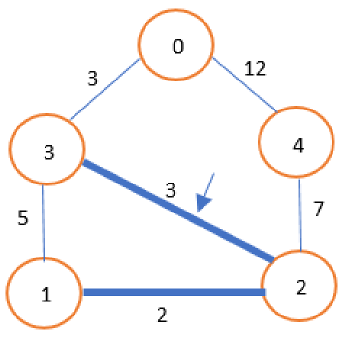
Kruskal's algorithm is another greedy algorithm that also finds the minimum spanning tree for a connected, undirected graph. Instead of starting with a single node, Kruskal's algorithm starts with an empty MST and adds edges one by one, always choosing the edge with the minimum weight that does not create a cycle.

Algorithm steps:

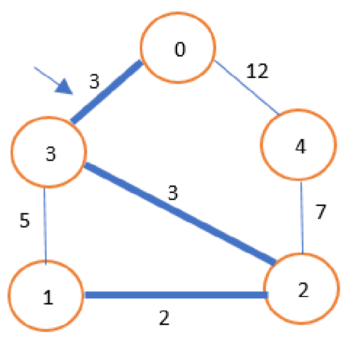
- Create a priority queue (min heap) containing all edges in the graph.
- Initialize an empty MST.



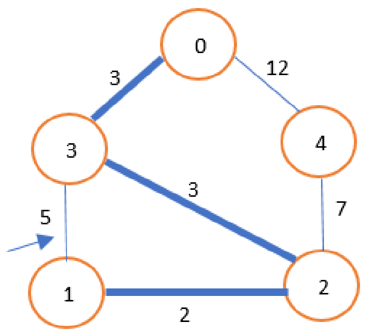
1



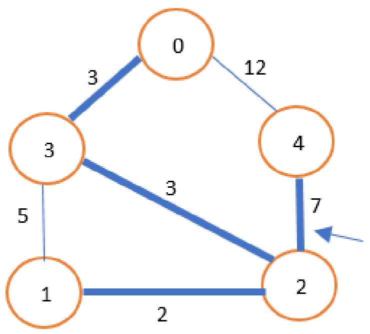
2



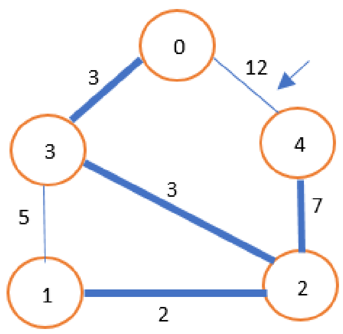
3



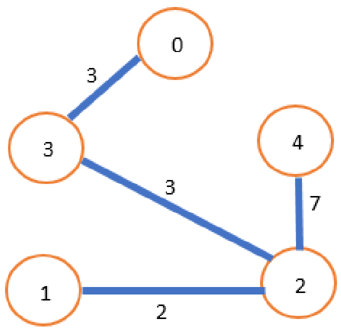
4



5



6



7

While the MST does not contain all vertices and the priority queue is not empty, do the following:

a. Extract the edge with the minimum weight from the priority queue.
b. If adding the edge to the MST does not create a cycle, add the edge to the MST.

Comparison:

- Both Prim's and Kruskal's algorithms find the minimum spanning tree of a connected, undirected graph.
- Prim's algorithm starts with a single node and grows the MST by adding the cheapest edge that connects the current MST to an outside vertex, while Kruskal's algorithm starts with an empty MST and adds edges one by one, avoiding cycles.
- Kruskal's algorithm can handle graphs with disconnected components, while Prim's algorithm requires the graph to be connected.

- In terms of time complexity, Kruskal's algorithm is generally faster than Prim's algorithm for sparse graphs since it uses a priority queue, while Prim's algorithm can be faster for dense graphs when implemented with a Fibonacci heap or other efficient data structures.