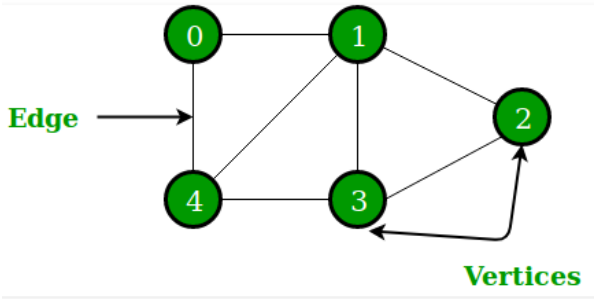
Graph

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as,

A Graph consists of a finite set of vertices (or nodes) and set of Edges which connect a pair of nodes.



In the above Graph, the set of vertices V = {0, 1, 2, 3, 4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

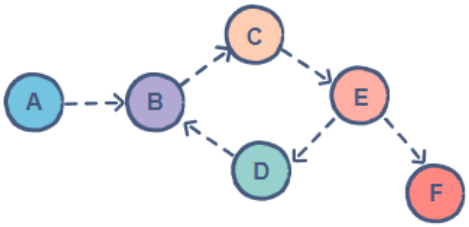
Why Use Graphs?

Using graphs, we can clearly and precisely model a wide range of problems. For example, we can use graphs for:

- Coloring maps (modeling cities and roads)
- Social Relations (sociology)
- Protein interactions (biology)
- Social networking (e.g. Facebook and Twitter)

Directed Graph

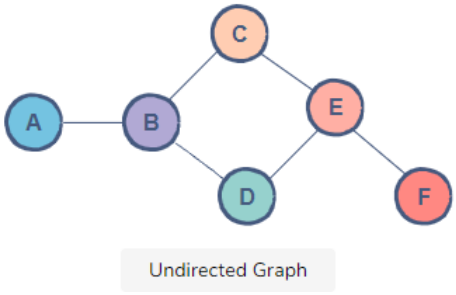
A directed graph is a set of vertices (nodes) connected by edges, with each node having a direction associated with it.



Directed Graph

Undirected Graph

In an undirected graph the edges are unidirectional, with no direction associated with them. Hence, the graph can be traversed in either direction. The absence of an arrow tells us that the graph is undirected.



Undirected Graph

Weighted Graph

In a weighted graph, each edge is assigned a weight or cost. Figure 1 outlines an example of directed and undirected weighted graphs.

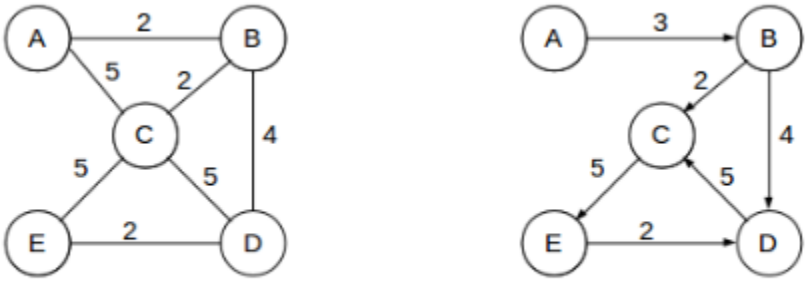
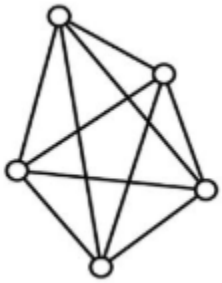


Figure 1: Examples of directed and undirected weighted graphs.

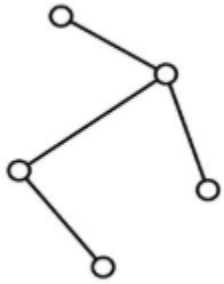
Complete Graph

In a **complete graph**, there is an edge between every single pair of vertices in the graph.



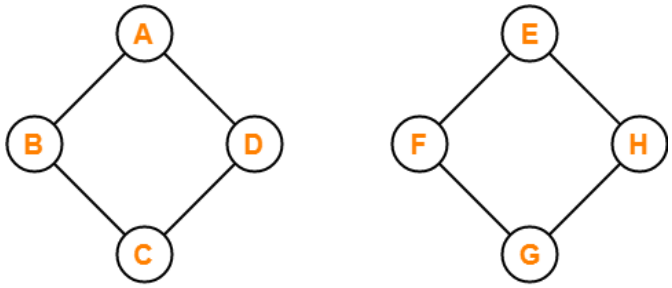
Connected Graph

In a **connected graph**, it's possible to get from every vertex in the graph to every other vertex in the graph through a series of edges, called a **path**.



Disconnected Graph

A graph is disconnected if at least two vertices of the graph are not connected by a path.



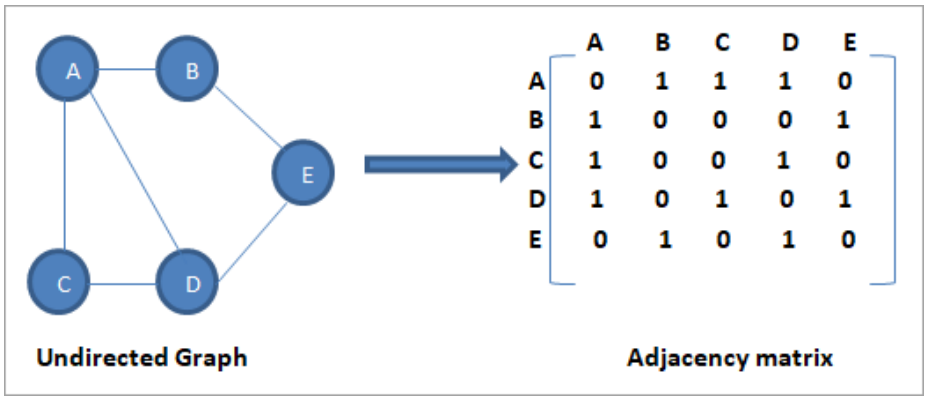
Example of Disconnected Graph

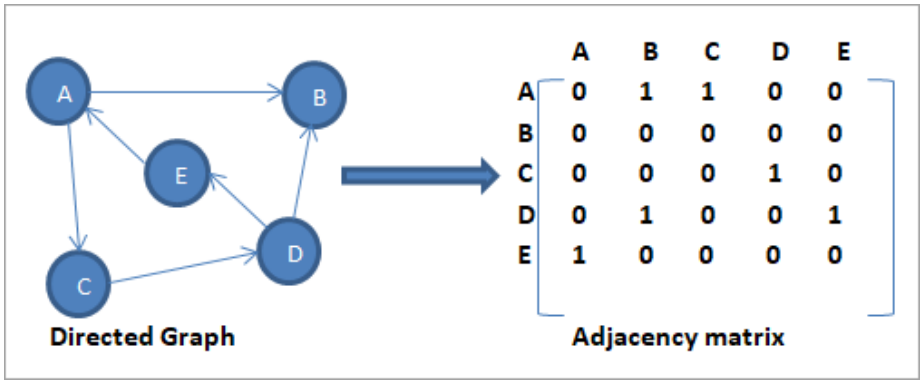
Graph Representation

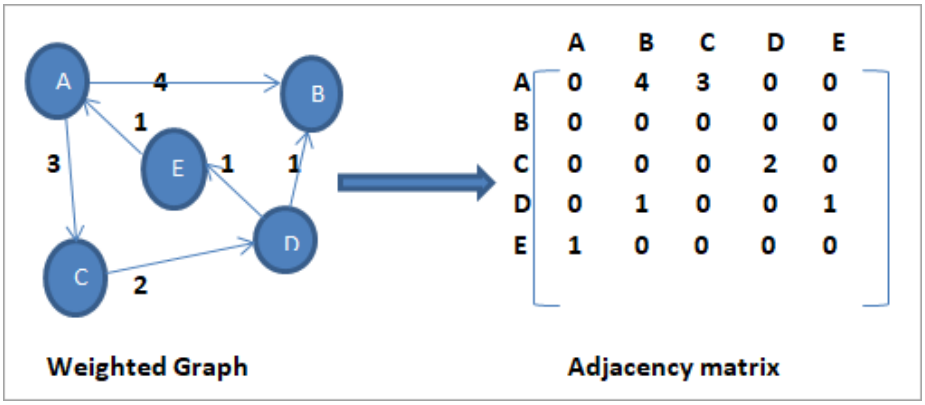
Graphs are commonly represented in two ways:

1. Adjacency Matrix

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex. If the value of any element `a[i][j]` is 1, it represents that there is an edge connecting vertex i and vertex j.

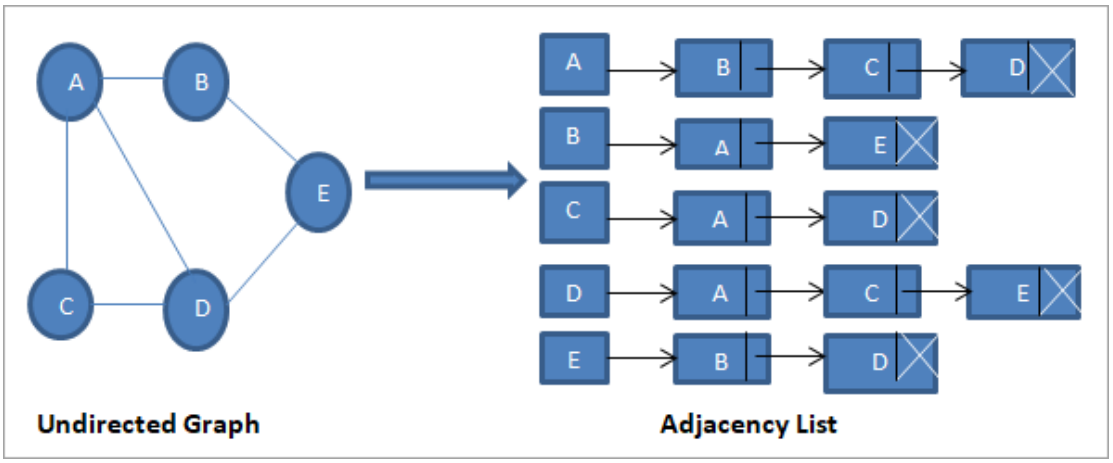


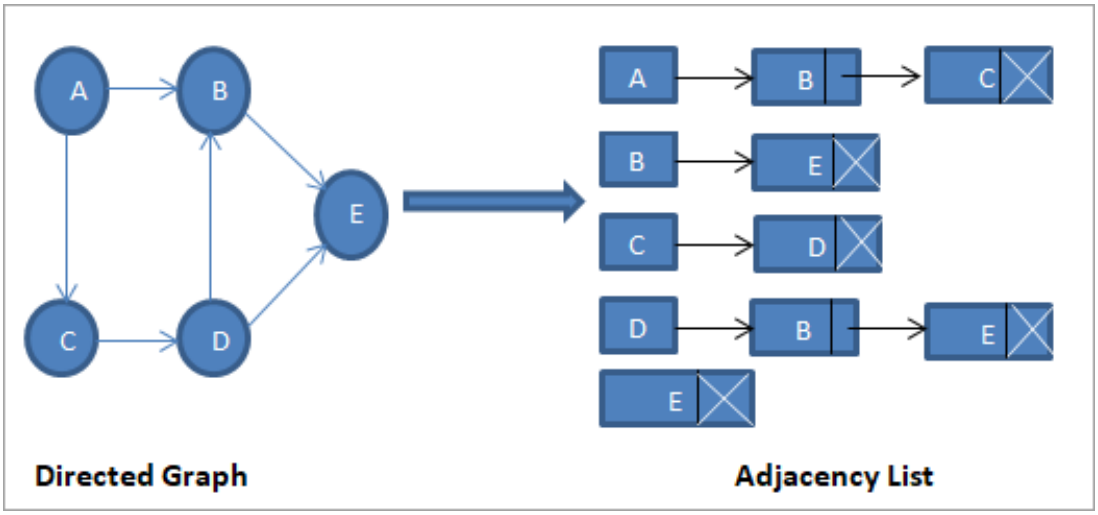


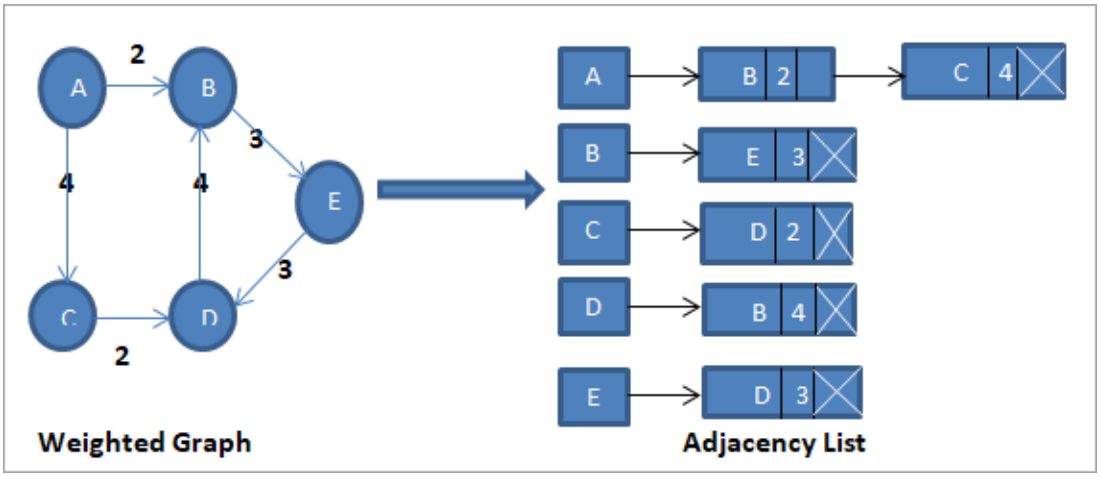


2. Adjacency List

The adjacency list representation maintains each node of the graph and a link to the nodes that are adjacent to this node. When we traverse all the adjacent nodes, we set the next pointer to null at the end of the list.


A
B
C
D
E
A
B
C
D
B
A
E
C
A
D
D
A
C
E
E
B
D
Undirected Graph
Adjacency List


A
B
C
D
E
A
B
C
B
E
C
D
D
B
E
E
Directed Graph
Adjacency List


A
B
C
D
E
2
3
4
4
3
2
A
B 2
C 4
B
E 3
C
D 2
D
B 4
E
D 3
Weighted Graph
Adjacency List

Graph Traversal

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

1. DFS (Depth First Search
2. BFS (Breadth First Search)

1. DFS (Depth First Search)

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

We use the following steps to implement DFS traversal...

Step 1- Define a Stack of size total number of vertices in the graph.

Step 2- Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.

Step 3- Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.

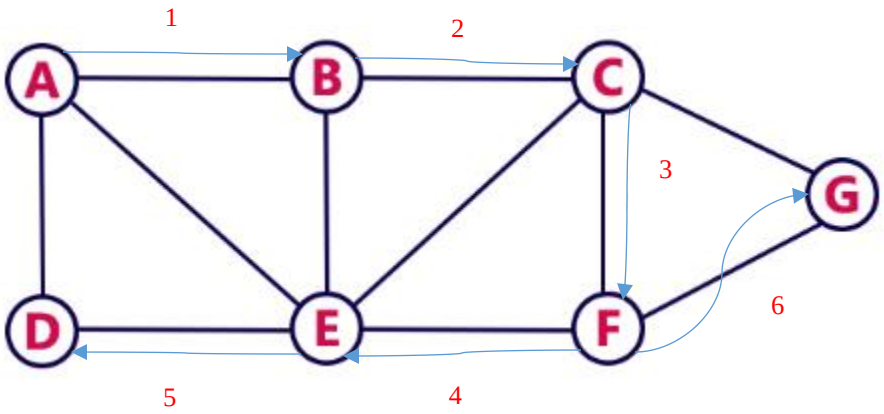
Step 4- Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.

Step 5- When there is no new vertex to visit then use back tracking and pop one vertex from the stack.

Step 6- Repeat steps 3, 4 and 5 until stack becomes Empty.

Step 7- When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

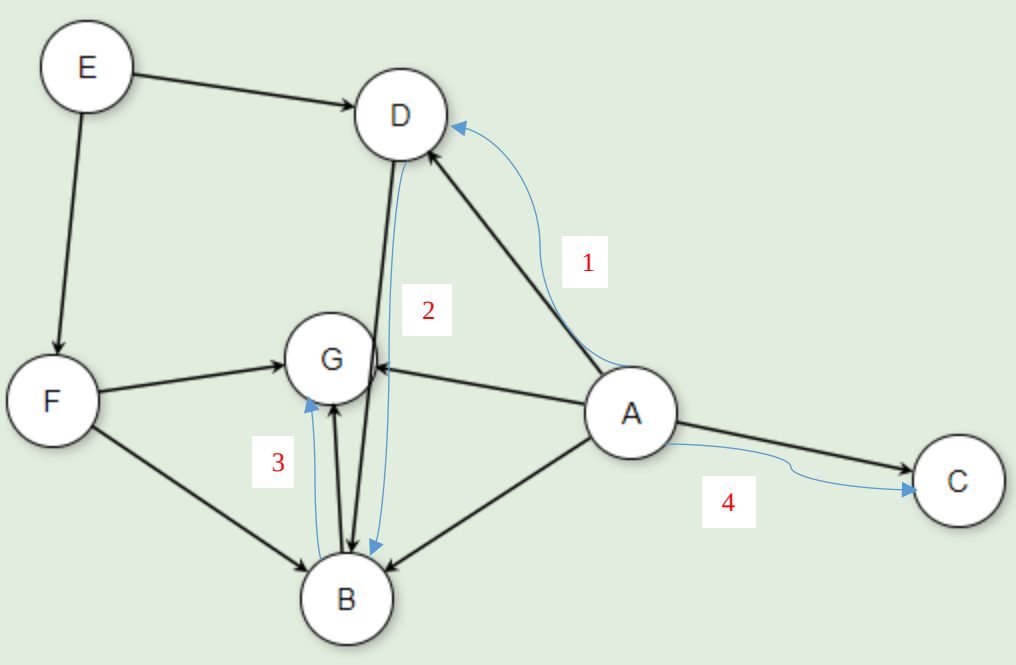
Consider the following example graph to perform DFS traversal

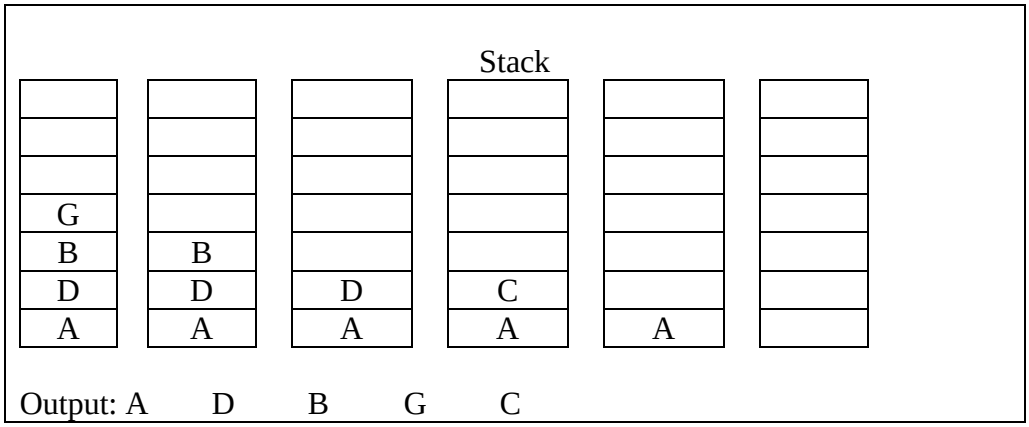


Stack

D							
E	E	G					
F	F	F	F				
C	C	C	C	C			
B	B	B	B	B	B		
A	A	A	A	A	A	A	

Output: A B C F E D G





Note that this traversal does not reach all of the nodes, due to the directions on the edges making some nodes unreachable from A. This is why DFS is typically done in the context of stating the traversal from every node.

2. BFS (Breadth First Search)

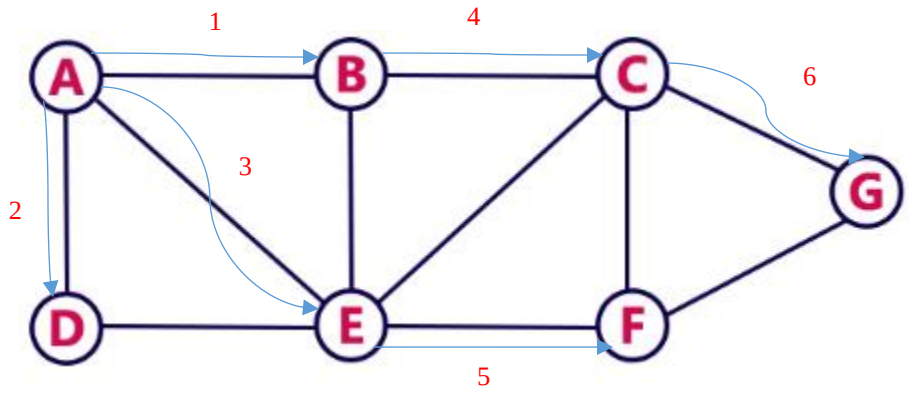
Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

We use the following steps to implement BFS traversal...

Step 1 - Define a Queue of size total number of vertices in the graph.
Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
Step 3 - Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.

Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
Step 5 - Repeat steps 3 and 4 until queue becomes empty.
Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

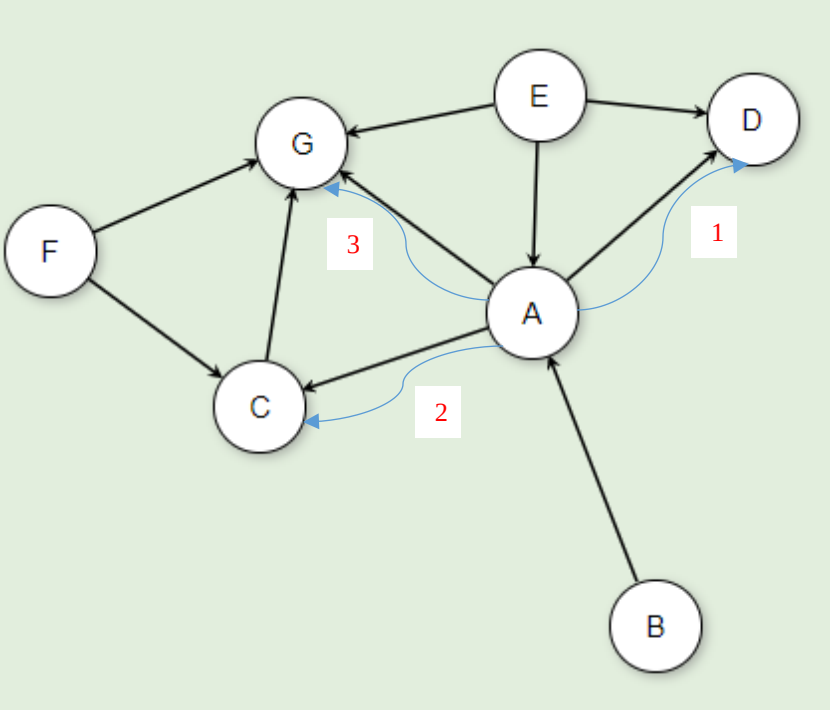
Consider the following example graph to perform BFS traversal



Queue

A							
	B						
	D	D					
	E	E	E				
		C	C	C			
				F	F		
					G	G	

Output: A B D E C F G



Queue

A				
	D			
	C	C		
	G	G	G	

Output: A D C G

Spanning Subgraph: A subgraph of a graph, which contains all the vertices of the graph.
Spanning Tree: A spanning tree is a subgraph (spanning subgraph) of a graph, which contains all the vertices of the graph and contains no cycle.
Minimum Cost Spanning Tree: It is a spanning tree whose cost is minimum.
There are two well-known algorithms to build minimum cost spanning tree. One is Prim's algorithm and another is Kruskal's algorithm.

Prim's Algorithm
Step-01:

- Randomly choose any vertex.
- The vertex connecting to the edge having least weight is usually selected.

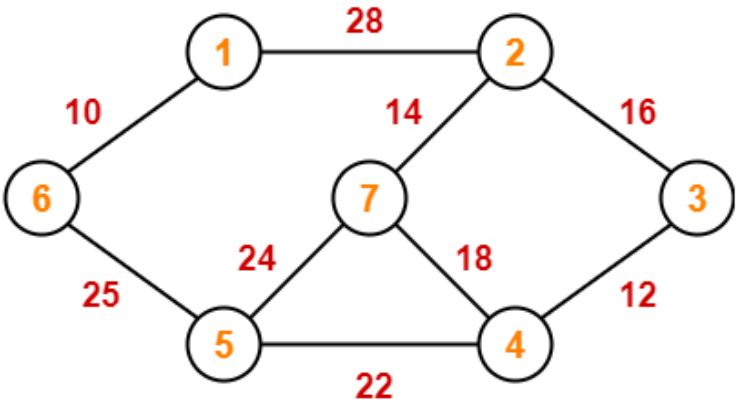
Step-02:

- Find all the edges that connect the tree to new vertices.
- Find the least weight edge among those edges and include it in the existing tree.
- If including that edge creates a cycle, then reject that edge and look for the next least weight edge.

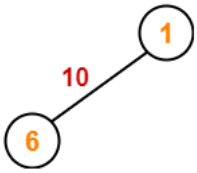
Step-03:

- Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

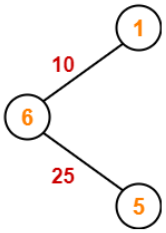
Ex-01: Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-



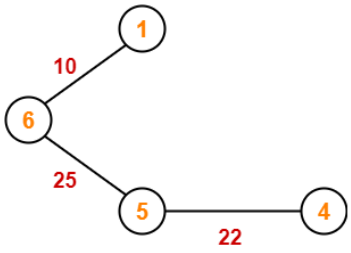
Solution:



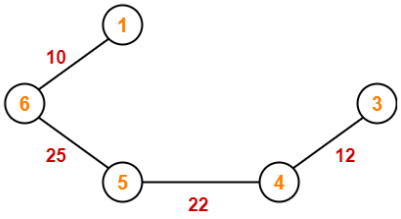
Step-01



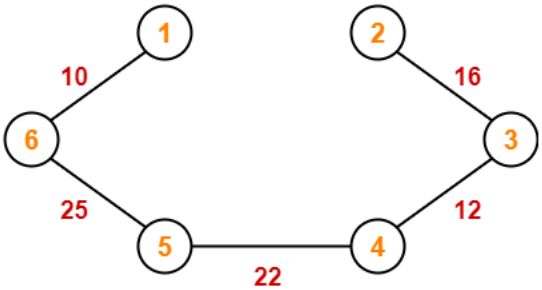
Step-02



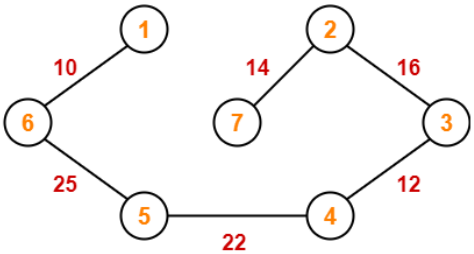
Step-03



Step-04



Step-05



Step-06

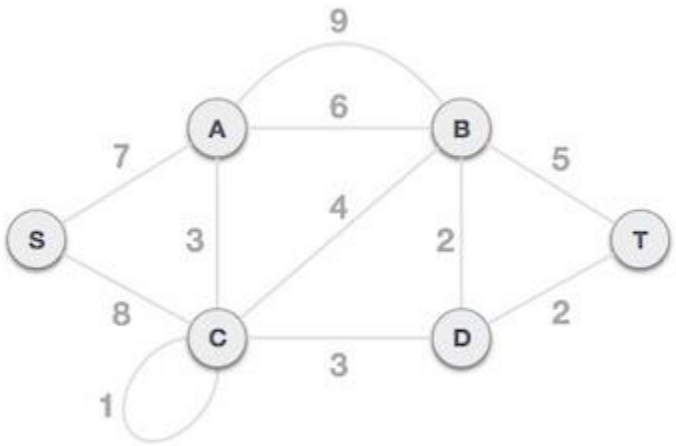
Now, Cost of Minimum Spanning Tree

= Sum of all edge weights

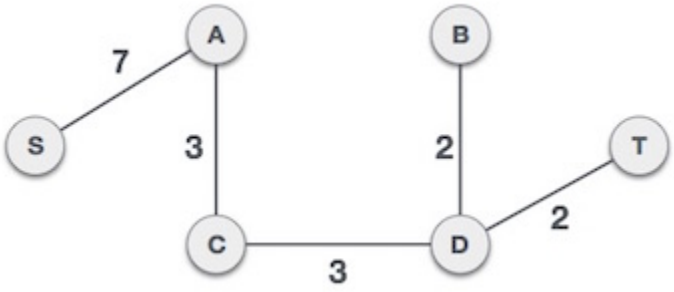
= 10 + 25 + 22 + 12 + 16 + 14

= 99 units

Ex-02



Solution: Remove all loops



Now, Cost of Minimum Spanning Tree

= Sum of all edge weights

= 7 + 3 + 3 + 2 + 2

= 17 units

Kruskal's Algorithm

Step-01:

- Sort all the edges from low weight to high weight.

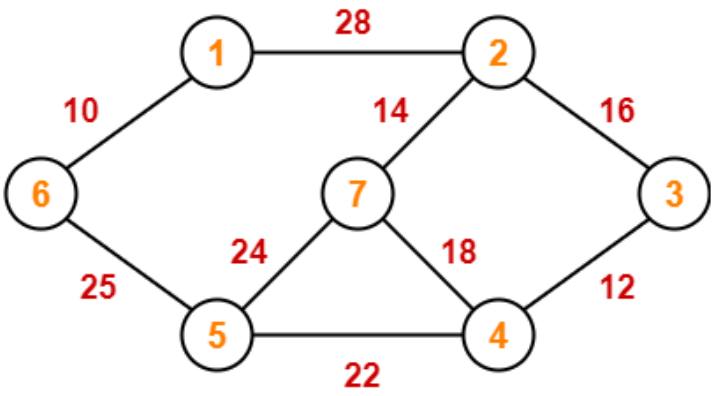
Step-02:

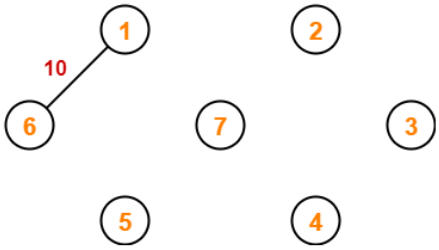
- Take the edge with the lowest weight and use it to connect the vertices of graph.
- If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step-03:

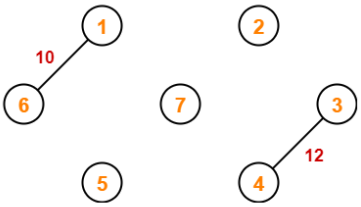
- Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.

Ex-01: Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm-

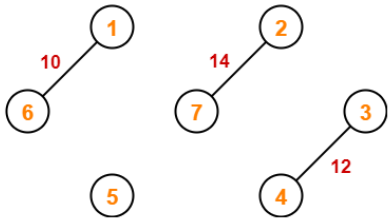




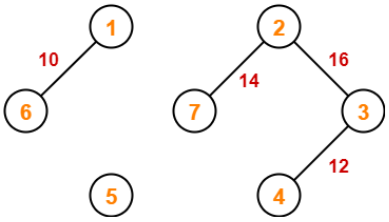
Step-01



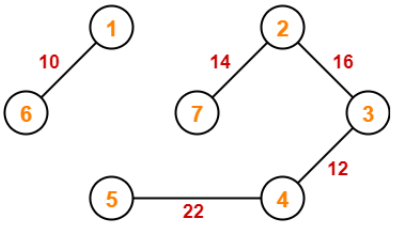
Step-02



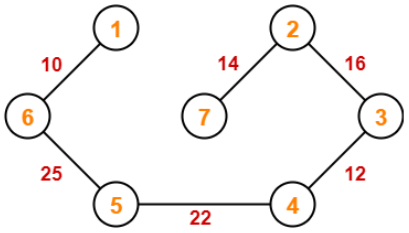
Step-03



Step-04



Step-05



Step-06

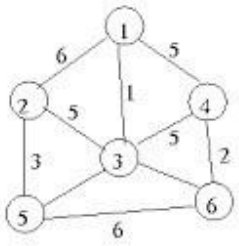
Weight of the MST

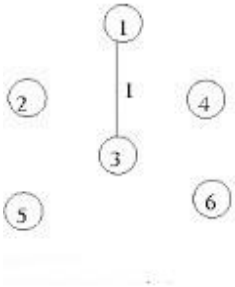
= Sum of all edge weights

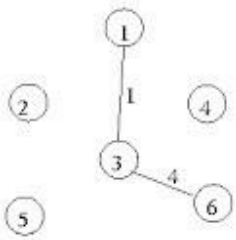
= 10 + 25 + 22 + 12 + 16 + 14

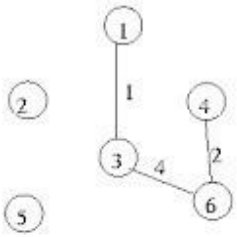
= 99 units

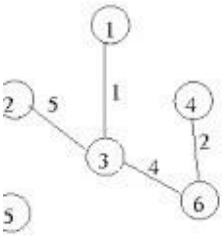
Prim's and Kruskal's Algorithm

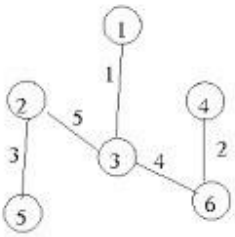






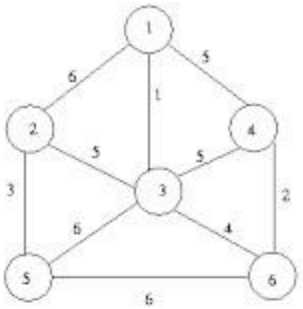


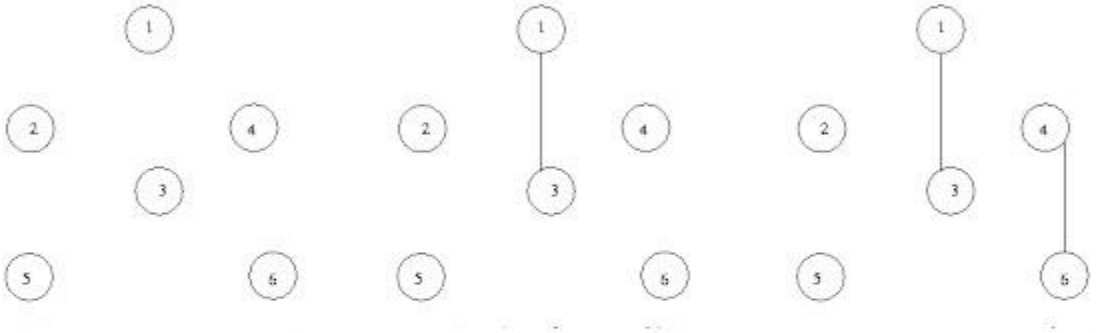


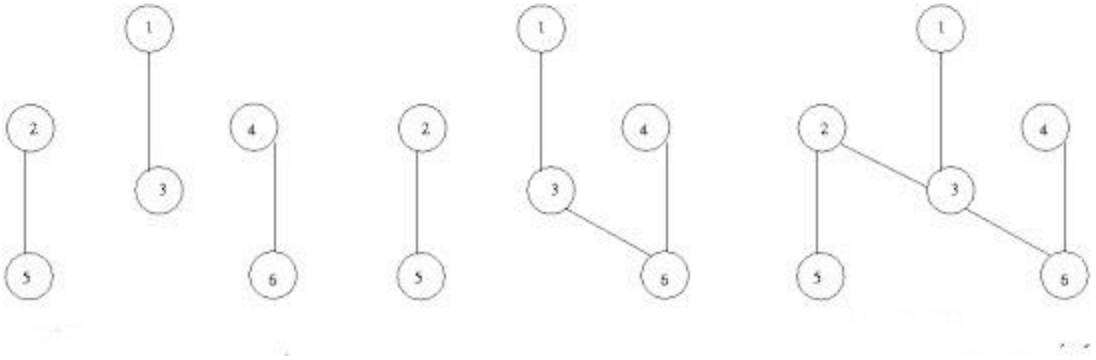


Weight of the MST = 1 + 4 + 2 + 5 + 3

= 18 units







Weight of the MST = 1 + 2 + 3 + 4 + 5

= 15 units