Analysis of Algorithm

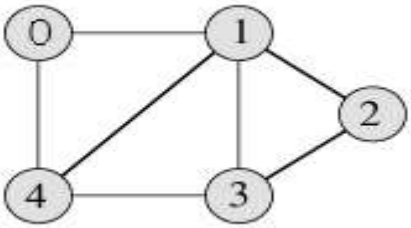
Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010

Week 13

❖ What is Graph

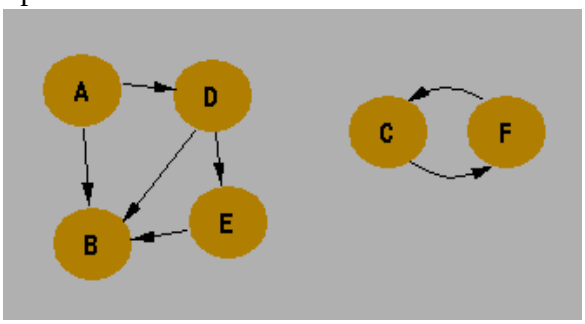
- Graph is a data structure.
- A person can use graph to save information.
- Graph is a data structure that consists of following two components:
  1. A finite set of nodes known as vertices.
  2. A finite set of edges that relate the nodes to each other.
- The set of edges describes relationships among the vertices.

Following is an example undirected graph with 5 vertices:



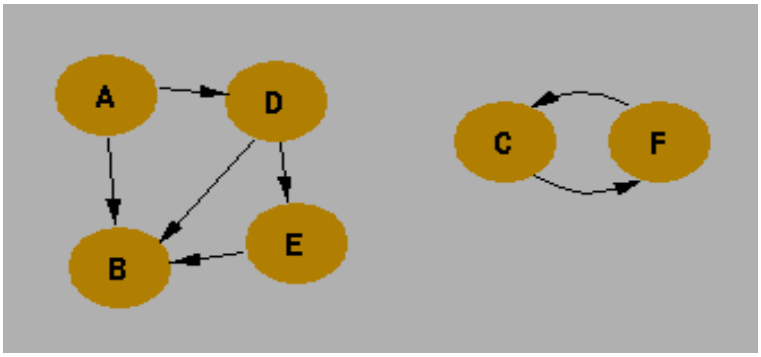
Graph vocabulary:

- The letters are held in what are called the ***vertices*** of the graph.
- Vertices can be connected to other vertices.
- A connection between 2 vertices is called an ***edge***.
- This example graph is a ***directed graph***.
- This just means that each edge in the graph is ***unidirectional***,
- i.e., it goes from one vertex to another.
  For example, there is an *edge* from **D** to **B**,
  but there is in no edge representing the reverse relationship (from **B** to **D**).
- Here is a simple graph that stores letters:



- Also, all the vertices aren't connected in this example graph.
- I.e., there are connections between **A, B, D** and **E**,
  but there is no way to *get to* vertex **C** from any of those vertices.

Thus, **A, B, D** and **E** from their own *component*.
A second component is made up of **C** and **F**.



Why we use graph?

- Some types of information are naturally represented in a graph.
- For example, the graph could be viewed as a map of what cities are connected by **train routes**.
  Viewing it this way;
  1. Each *vertex* represents a particular city
  2. Each *edge* represents whether there is a train route from one city to another.
  3. We can imagine that the edges are unidirectional
     since trains are only allowed to go in one direction on the track.
- The *edges* in our graph represent a very simple relationship.
- For example; one city is connected to another.
- We can store information at vertices (e.g., the city name)
- We can also store information at each edge.
  For example,
  we want to store the ***distance*** between cities at edges OR
  the ***time*** the trip takes OR
  the ***cost*** of the ticket OR
  all of those pieces of information.

Real Life Example:

Graphs are used to represent many real life applications:

1. Train route
2. Networks
3. Network – can be path in a city, telephone network or circuit network
4. Social Networking – such as facebook; in facebook each person is a vertex (node). Each node is a structure and contains information like person id, name, gender and location.
5. Communication networks – i.e. Network topologies.

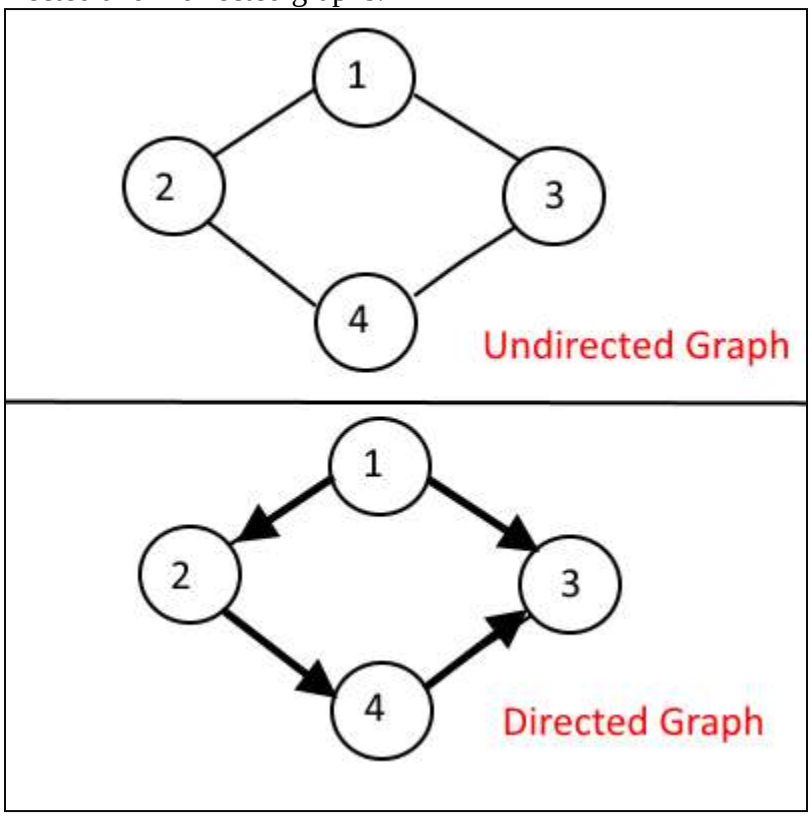
6. Computer networks and the Internet. Often nodes will represent end-systems or routers, while edges represent connections between these systems.
7. **Molecules:** Graphs can be used to model atoms and molecules for studying their interaction and structure among other things.

Representation of graph:

1. Adjacency Matrix

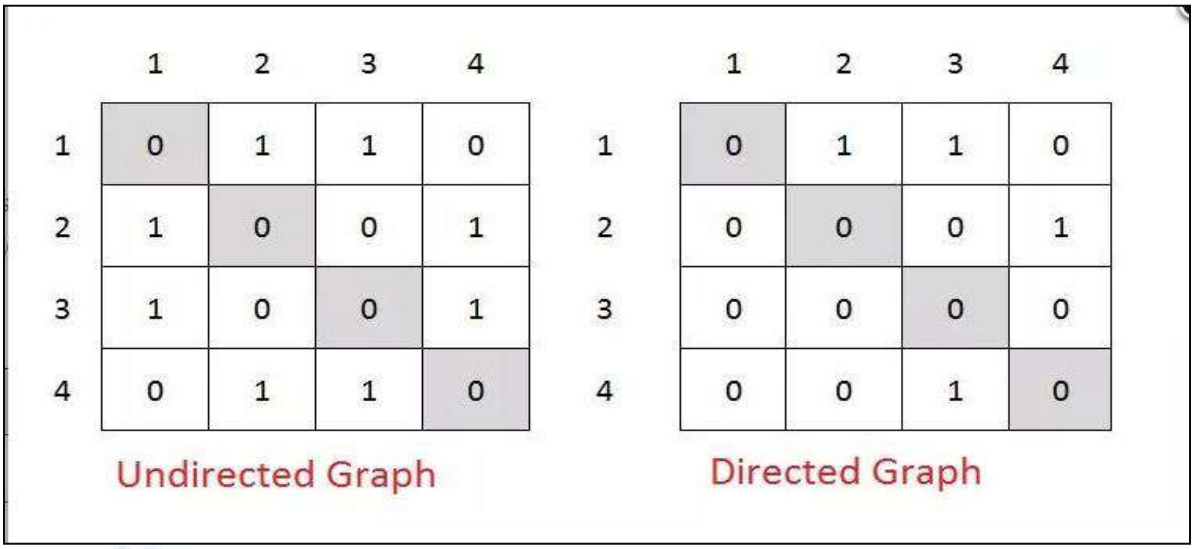
- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
- Let the 2D array be A[][], a blockA[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
- Adjacency matrix for undirected graph is always symmetric.
- Adjacency Matrix is also used to represent weighted graphs.
- If A[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

These are the directed and indirected graphs:



The adjacency matrix of the following diected and undirected graph is:

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010



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

Undirected Graph

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

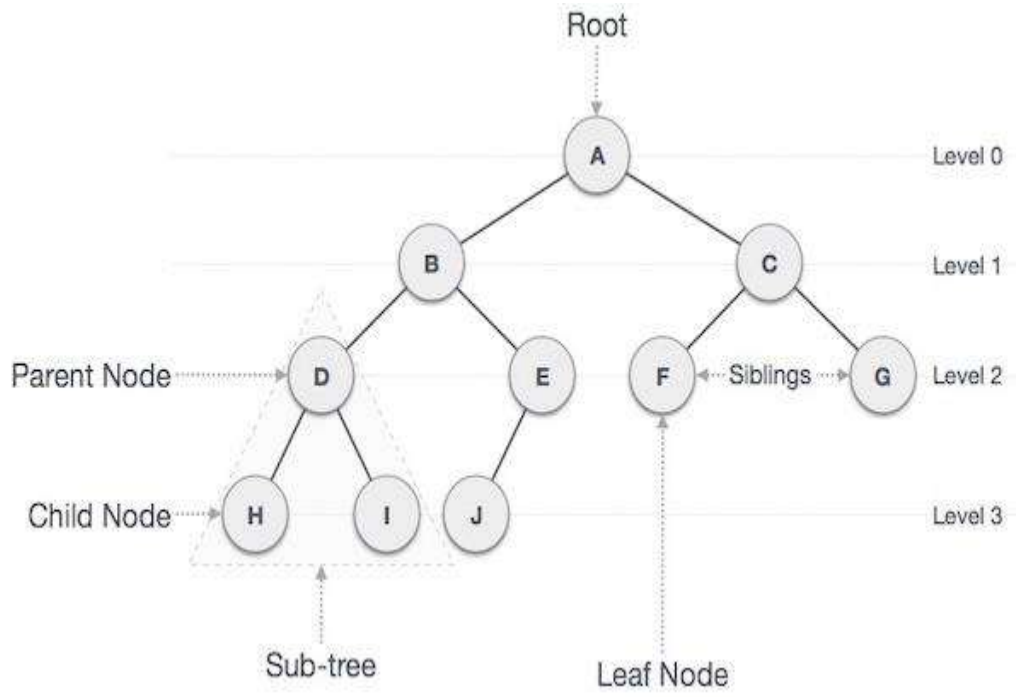
Directed Graph

❖ Binary Search Tree and basic Terms

Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010



Important Terms

Following are the important terms with respect to tree.

- **Path** − Path refers to the sequence of nodes along the edges of a tree.
- **Root** − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** − Any node except the root node has one edge upward to a node called parent.
- **Child** − The node below a given node connected by its edge downward is called its child node.
- **Leaf** − The node which does not have any child node is called the leaf node.
- **Subtree** − Subtree represents the descendants of a node.
- **Visiting** − Visiting refers to checking the value of a node when control is on the node.
- **Traversing** − Traversing means passing through nodes in a specific order.
- **Levels** − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- **keys** − Key represents a value of a node based on which a search operation is to be carried out for a node.

Binary Search Tree (BST)

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

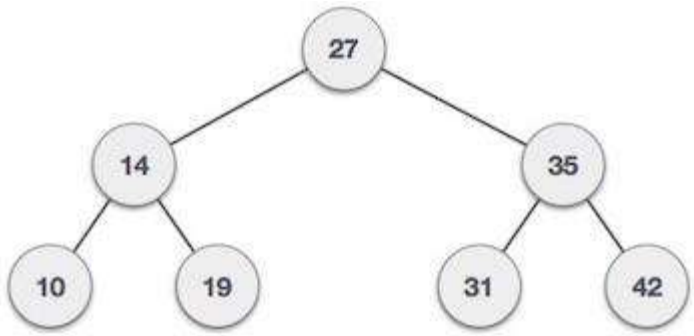
Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as

left_subtree(keys) ≤ node (key) ≤ right_subtree (keys)

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST −



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Basic Operations

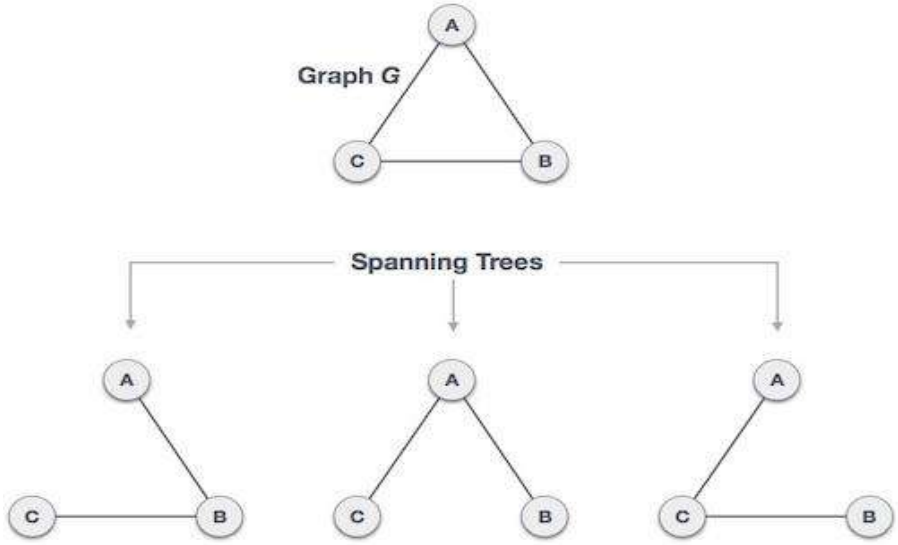
Following are the basic operations of a tree −

- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.
- **In-order Traversal** − Traverses a tree in an in-order manner.
- **Post-order Traversal** − Traverses a tree in a post-order manner.

What is Spanning Tree?

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



We found three spanning trees off one complete graph. A complete undirected graph can have maximum n(n-2) number of spanning trees, where n is the number of nodes. In the above addressed example, 3(3−2) = 3 spanning trees are possible.

Applications of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture

Mathematical Properties of Spanning Tree

- Spanning tree has n-1 edges, where n is the number of nodes (vertices).
- From a complete graph, by removing maximum e - n + 1 edges, we can construct a spanning tree.

- A complete graph can have maximum n(n-2) number of spanning trees.

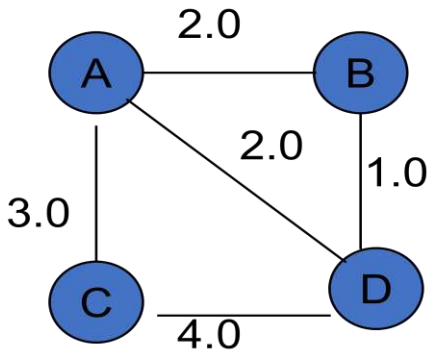
Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

❖ Minimum Spanning Tree (MST)

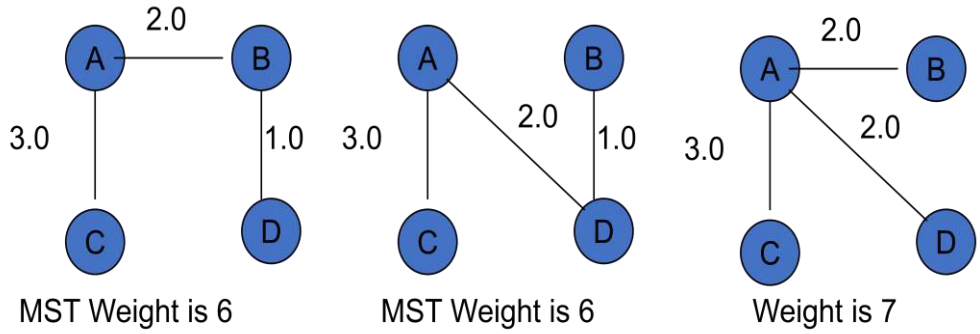
In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

- A Spanning Tree for a connected, undirected graph, G = (V, E), is a subgraph of G that is an undirected tree and contains all the vertices of G.
- In a weighted graph G = (V, E, W), the weight of a subgraph is the sum of the weights of the edges in the subgraph.
- A minimum spanning tree (MST) for a weighted graph is a spanning tree with minimum weight.

Consider the following graph



The possible spanning trees for above graph are as follow



Applications of MST

Minimum spanning trees are useful when we want to find the cheapest way to connect a

- Set of cities by roads
- Set of electrical terminals or computers by wires or telephone lines

Etc…

Minimum Spanning-Tree Algorithms

We shall learn about two most important spanning tree algorithms here −

- Kruskal's Algorithm
- Prim's Algorithm

Both are greedy algorithms.

❖ Prim's Algorithm for Minimum Spanning Tree

- Prim's algorithm begins by selecting an arbitrary starting vertex and then "**branches out**" form the past of the tree constructed so far by choosing a new vertex and edge at each iteration.
- The new edge connects the new vertex to the previous tree.
- the vertices may be thought of as divided into three (disjoint) categories as follows:

1. **Tree Vertices:** in the tree constructed so far
2. **Fringe Vertices:** Not in the tree, but adjacent to some vertex in the tree.
3. **Unseen vertices:** all others

- The key step in the algorithm is the selection of a vertex from the fringe
- Prim's algorithm always chooses an edge of minimum weight from a tree vertex to a fringe vertex.

The general algorithm structure is

Prim MST(G, n)

Initialize all the vertices as unseen

Select an arbitrary vertex s to start the tree; reclassify it as tree.

Reclassify all the vertices adjacent to s as fringe.

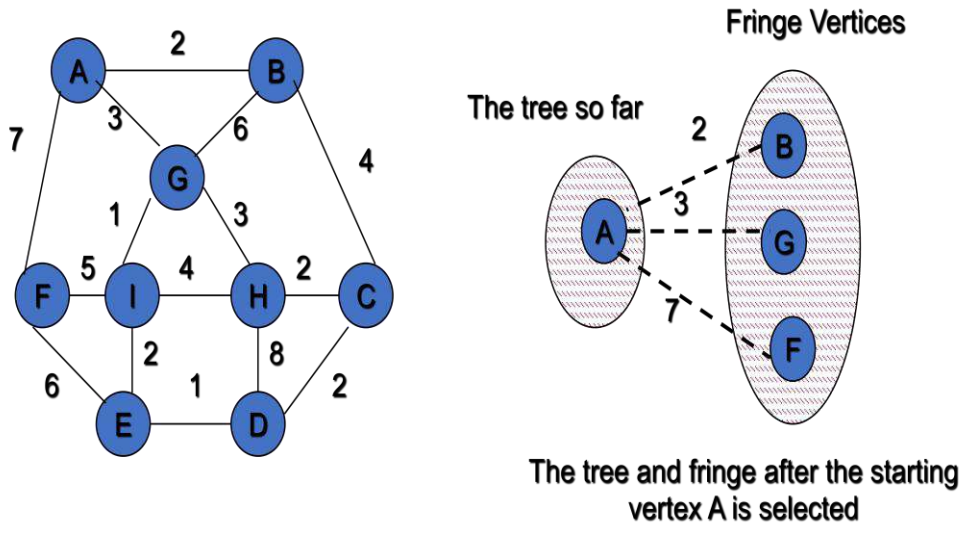
While there are fringe vertices

Select an edge of minimum weight between a tree vertex t and a fringe vertex v.

Reclassify v as tree; add edge tv to the tree;

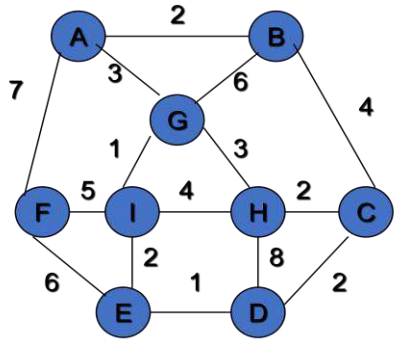
Reclassify all unseen vertices adjacent to v as fringe.

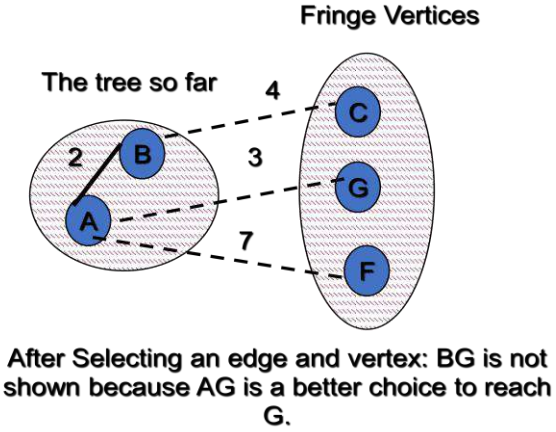
Example



In above diagram we have started from node A, there are three possible node to there we can move (B, G, F). Minimum weight is for B, so B will be selected as shown in next diagram.

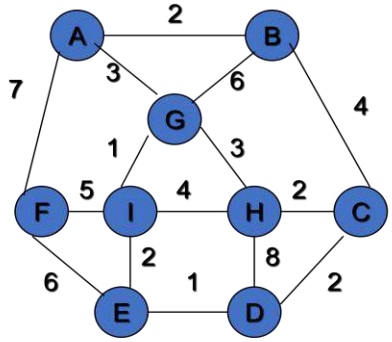
Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010

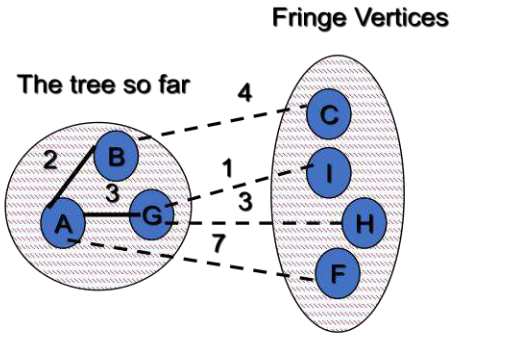




After Selecting an edge and vertex: BG is not shown because AG is a better choice to reach G.

Now 3 will be selected as in diagram



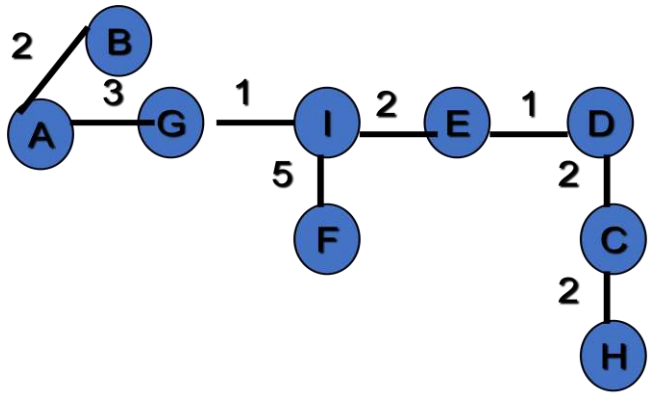


After Selecting an edge AG : GB is not shown because vertex B is already include in a tree.

This process will continue and finally we will get following spanning tree which will be referred as minimum spanning tree.

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010

The final **Minimum Spanning tree** after prim's algorithm is



❖ Kruskal's Algorithm for Minimum Spanning Tree

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

Steps

- Sort the edges of the graph in non decreasing order.
- Start making the minimum spanning tree by taking the first edge from sorted edges.
- Continue taking the edges from the list of sorted edges.
- If the inclusion of a selected edge in minimum spanning tree makes a cycle then reject that edge, otherwise include that edge.
- Given a directed graph, and a starting vertex **S, find the shortest path from S to every other vertex in the graph**.
    - Un-weighted Directed Graph
    - Weighted Directed Graph

Un-weighted Shortest Path

- No. of edges on the path
- Process vertices in layers
- Vertices closest to start s are evaluated first
- Most distant vertices are evaluated last

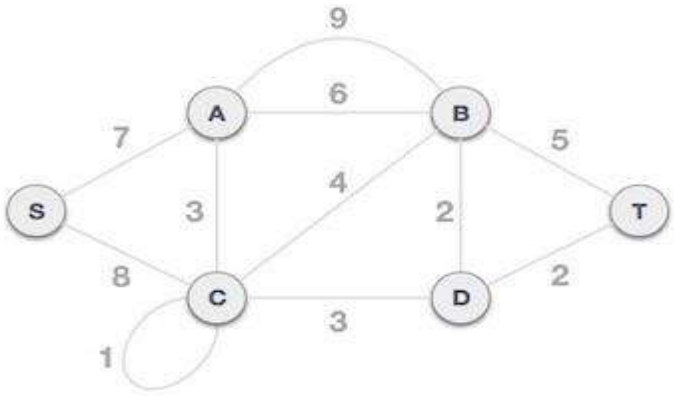
Algorithm

1. Ds = 0, for all other vertices Dw = ∝ ; Dv = Distance from s to v
2. Create a Queue (Q) ; Enqueue(s, Q); // Insert starting vertex //

3. While (Q Not Empty) // for all vertices //
4. v = Dequeue(Q) // Delete vertex from Queue //
5. for each w adjacent to v
    a. if (Dw = = ∝) {
        i. Dw = Dv + 1; // increase no. of edges on path //
        ii. Pw = v ; // predecessor vertex for Path //
        iii. Enqueue(w, Q); //insert vertex for processing //
    b. }

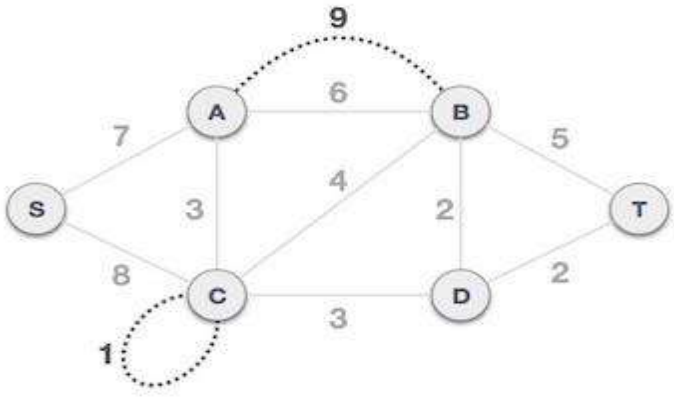
Example

To understand Kruskal's algorithm let us consider the following example −

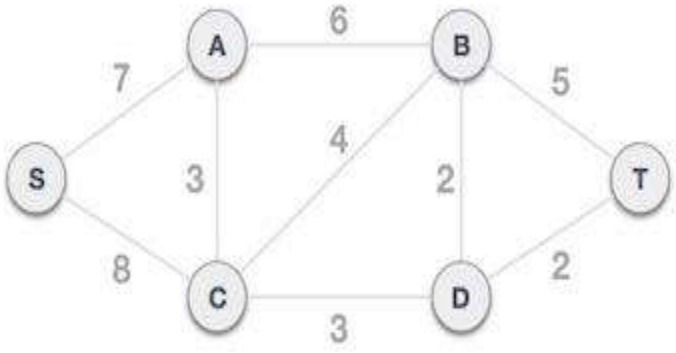


Step 1 - Remove all loops and Parallel Edges

Remove all loops and parallel edges from the given graph.



In case of parallel edges, keep the one which has the least cost associated and remove all others.



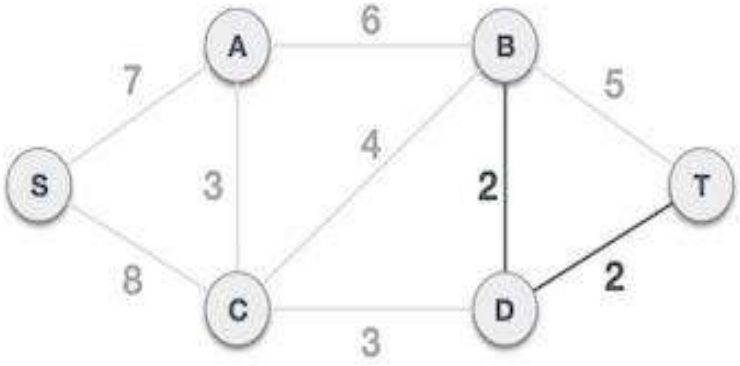
Step 2 - Arrange all edges in their increasing order of weight

The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

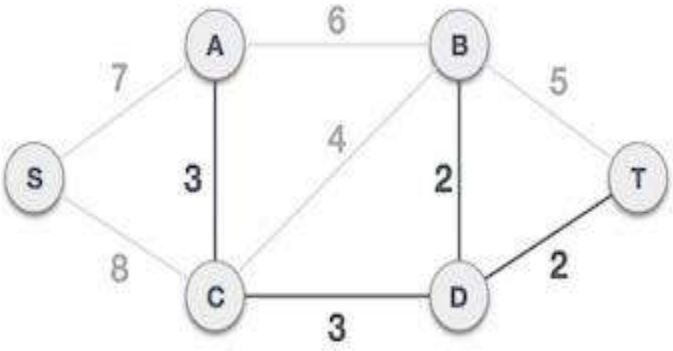
Step 3 - Add the edge which has the least weightage

Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not including the edge in the graph.

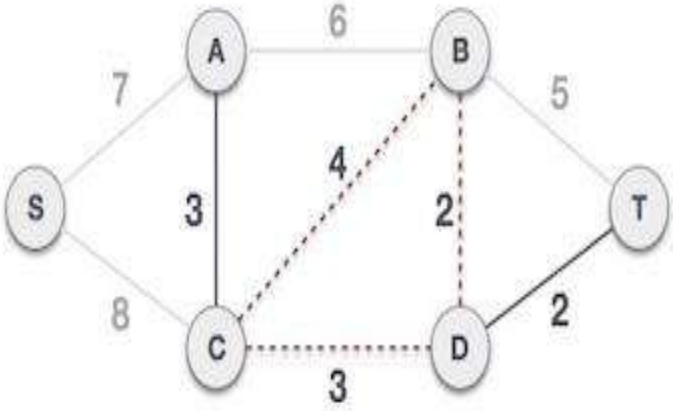


The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.

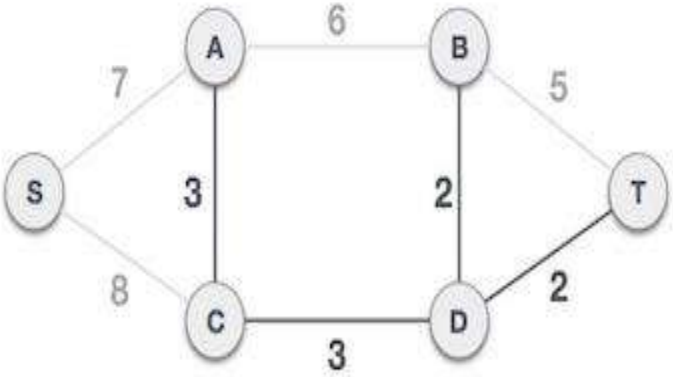
Next cost is 3, and associated edges are A,C and C,D. We add them again



Next cost in the table is 4, and we observe that adding it will create a circuit in the graph.

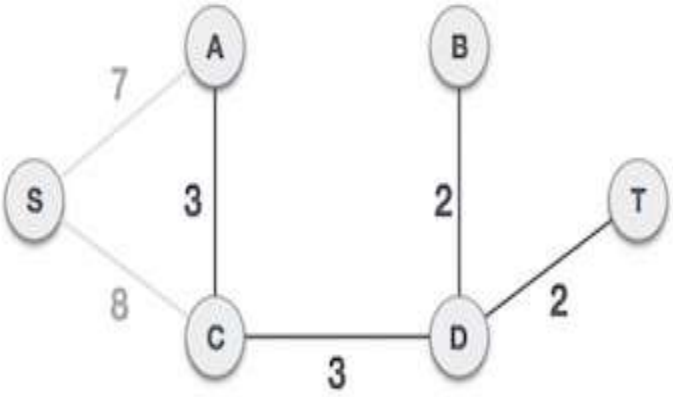


We ignore it. In the process we shall ignore/avoid all edges that create a circuit.

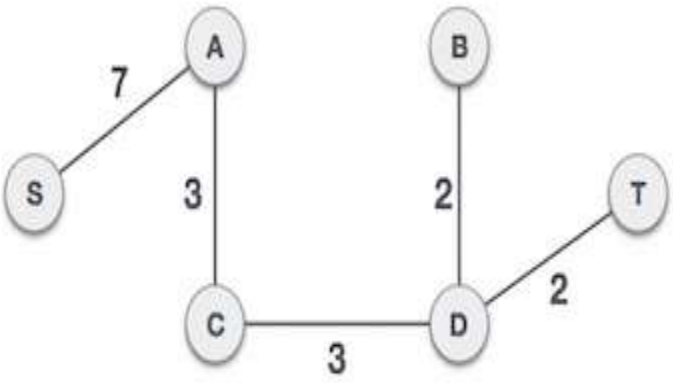


Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk WhatsApp# 0346-5100010

We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.



Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.



By adding edge S, A we have included all the nodes of the graph and we now have minimum cost spanning tree.