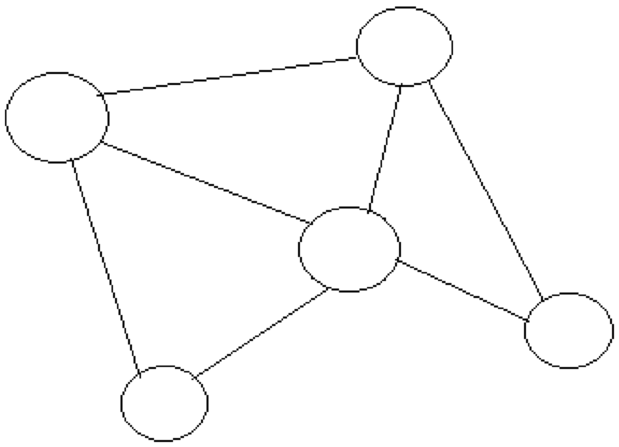
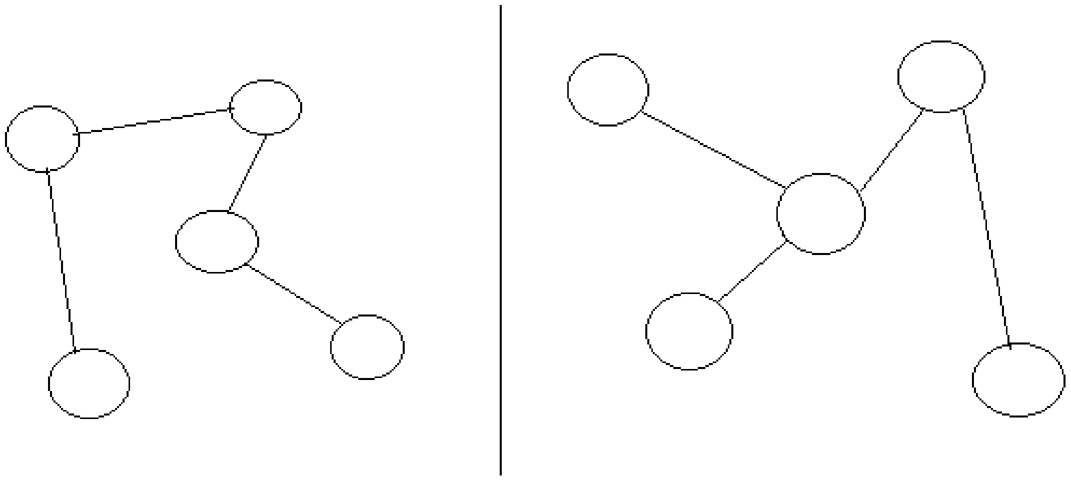
<u>Minimum Spanning Tree (Prim's Algorithm)</u>

Before we understand what minimum spanning tree of a graph is let us understand what a spanning tree is.
Spanning Tree: The term spanning tree consists of two words **- spanning & tree.** Spanning means include all vertices of a graph and tree means a graph without cycles. Therefore we can conclude that a spanning tree of graph *g* is a tree of *g* which will have all the vertices of g but will have no cycles. A pictorial example will make things easier.

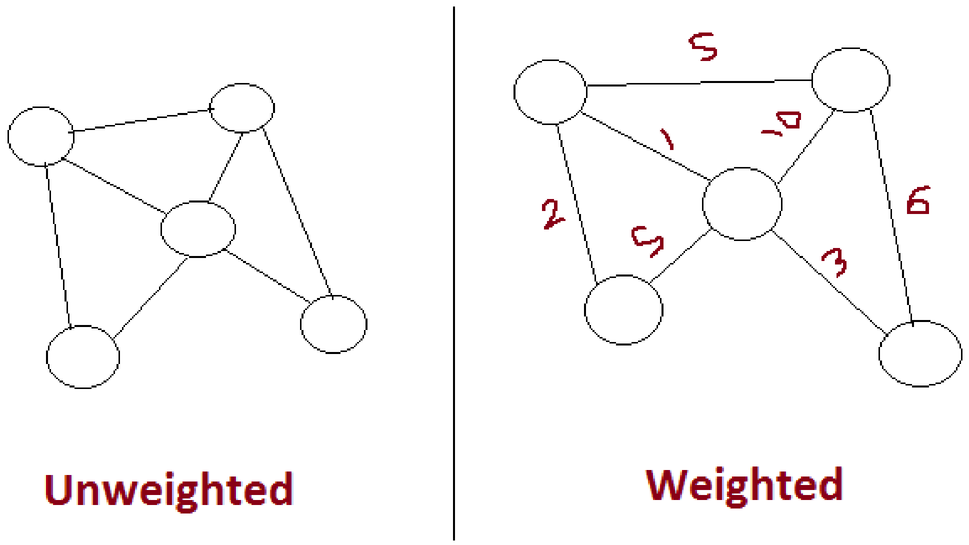


For the above graph, its spanning trees looks like following diagram.



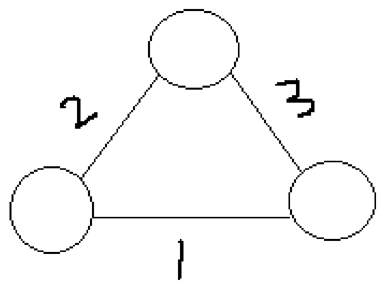
A graph can have multiple spanning trees. The given graph has five vertices and seven edges and they formed cycles. Spanning tree of that graph, as mentioned, is a tree which will contain all the vertices of that graph but with zero cycles. The cycles are removed from the graph by eliminating those edges which create them. Note that in a spanning tree the number of edges will be number of vertices - 1.

Weighted Graph: All the graphs we have seen so far were unweighted graphs, meaning the edges of the graphs did not carry a value. All the edges had the same power. Now we are going to look at weighted graphs where each edge carries a value and the power of all the edges will be different. Look at the following diagram. It shows an unweighted and a weighted graph respectively.

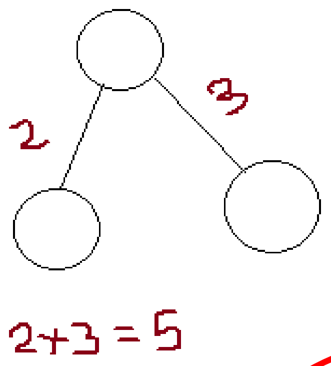


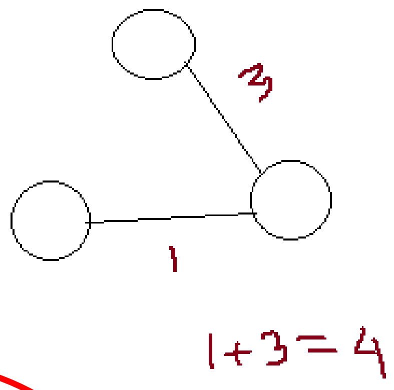
In real life scenario we can consider the vertices as cities and the edges as paths between them. Weights on edges indicate the level of traffic on each path.

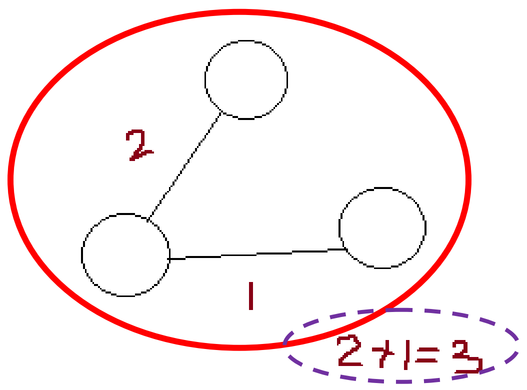
Minimum Spanning Tree: It is only applicable for weighted graphs. Minimum spanning tree is the derivation of that tree from a given graph, whose summation of the weights of the edges is the least. It means at first we will derive all possible spanning tree of a graph in the same way as shown earlier. Then we will add up all the weights of each spanning tree. The minimum spanning tree will be that spanning tree whose addition of the weights is the minimum. The pictorial explanation is given below.



This a simple graph. Its all possible spanning trees along with the summation of weights are given below.







All the trees above are valid spanning trees of the given graph. Among them the minimum spanning tree is the one circled red because it has the minimum total weight, which is 3. The other spanning trees have total weights of 5 and 4 respectively.
Finding all the possible spanning trees and deducing the minimum is not an efficient way. Hence algorithms have been developed to find the MST without finding all the possibilities. They are Prim’s and Kruskal’s MST algorithms.
This lecture explains Prim’s algorithm. Below is the pseudocode.

```
Prim(G)
for each vertex v in the graph G{
        key[v] = ∞
        parent[v] = null
}
// select a source and let’s call it r
key[r]=0;
B= put all the vertices of the graph
        while (B! empty){
        u = the vertex with the smallest key in B;
        remove u from B
        foreach vertex v adjacent[u]{
                if (v is in B && weight(u,v)<key[v])
                parent[v]=u
                key [v] = weigth(u,v)
                }
        }
```

Explanation:

```
for each vertex v in the graph G{
        key[v] = ∞
        parent[v] = null
}
```

Each vertex of the graph will have two properties – key and parent – and initially we will set them to ∞ and null respectively for all vertices.

```
key[r]=0;
```

We will select a source randomly and let us call it r. We will change the key of r to 0.

```
B= put all the vertices of the graph
```

We will keep all the vertices into an array and let us name the array as B.

```
while (B! empty){
u = the vertex with the smallest key in B;
remove u from B
```

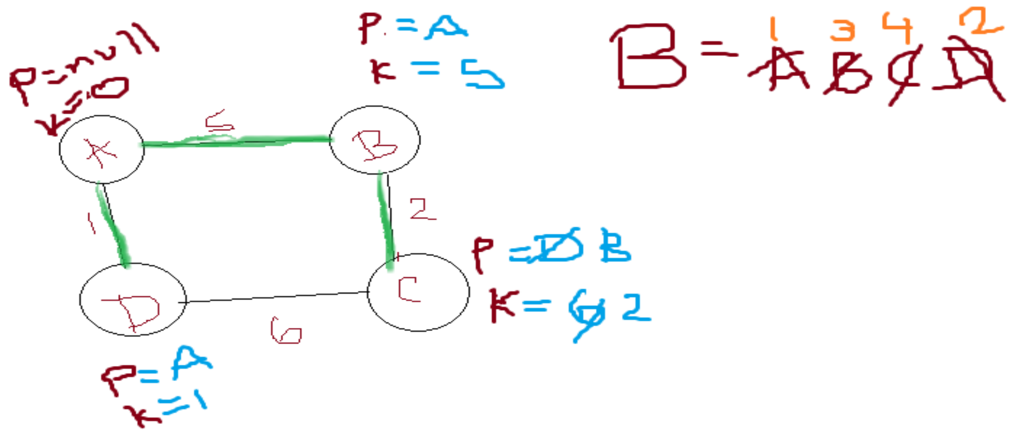
We will run a loop until the array B is empty. If not empty then we will pick the vertex with the smallest key. If there are multiple vertices with the same key, pick anyone. The picked vertex will be stored in a variable called u. Then remove the selected vertex from B.

```
foreach vertex v adjacent[u]{
if (v is in B && weight(u,v)<key[v])
parent[v]=u
key [v] = weigth(u,v)
```

Now for the picked vertex, we will deal with its neighbors or adjacent vertices one at a time. For each adjacent vertex, v, we will check two conditions: 1) whether it is in the array B 2) the weight on the edge from the picked vertex, u, to this adjacent vertex, v, is less than the key of v or not. It both the conditions are satisfied then we will replace the parent of v with u and the key of v as the weight on the edge. When all the adjacent vertices have been dealt with, we will go the previous loop and pick another vertex from B is the array is not empty. This will continue until becomes empty.

Now draw the resultant tree looking the parent of each vertex. This tree is guaranteed to a MST.

Below is the simulation of Prim's on a graph. The green edges are to be considered for obtaining an MST.



P=null
k=0
P=A
k=5
B= A B C D
1 3 4 2
A
B
C
D
5
2
1
6
P=D B
K=6 2
P=A
k=1