PROGRAM 9: SPANNING TREE IMPLEMENTATION

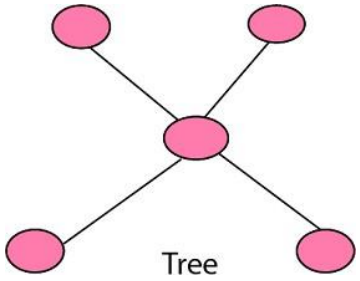
Aim: To write a program in C++ for implementing minimum spanning tree.

Description:

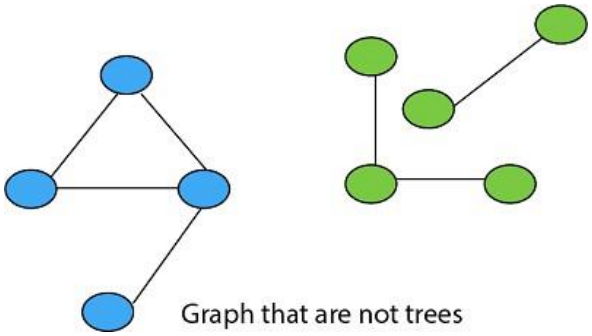
Tree:

A tree is a graph with the following properties:

1. The graph is connected (can go from anywhere to anywhere)
2. There are no cyclic (Acyclic)



Tree



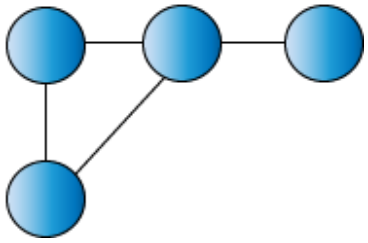
Graph that are not trees

Spanning Tree:

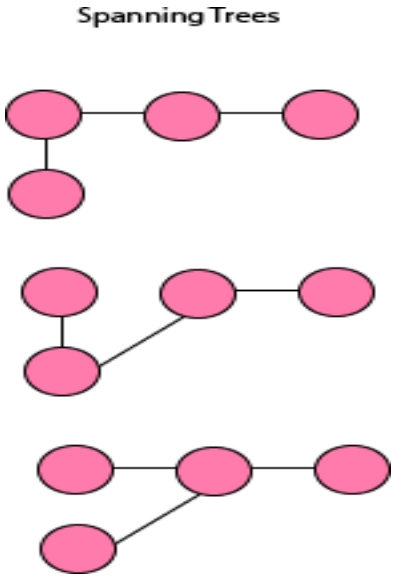
Given a connected undirected graph, a spanning tree of that graph is a subgraph that is connected and does not have a cycle in it (i.e a tree that has all vertices joined). A single graph can have many spanning trees.

For Example:

Connected Undirected Graph

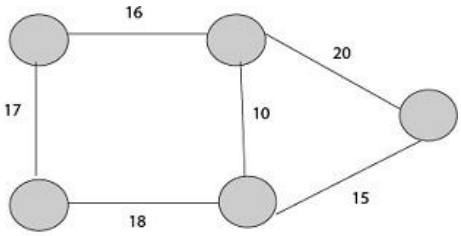


For the above-connected graph. There can be multiple spanning Trees like

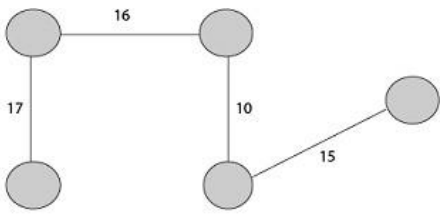


Minimum Spanning Tree:

Minimum Spanning Tree is a Spanning Tree which has minimum total cost. If we have a linked undirected graph with a weight (or cost) combine with each edge. Then the cost of spanning tree would be the sum of the cost of its edges.



Connected , Undirected Graph



Minimum Cost Spanning Tree
Total Cost = 17+16+10+15=58

The minimum spanning tree from a graph is found using the following algorithms:

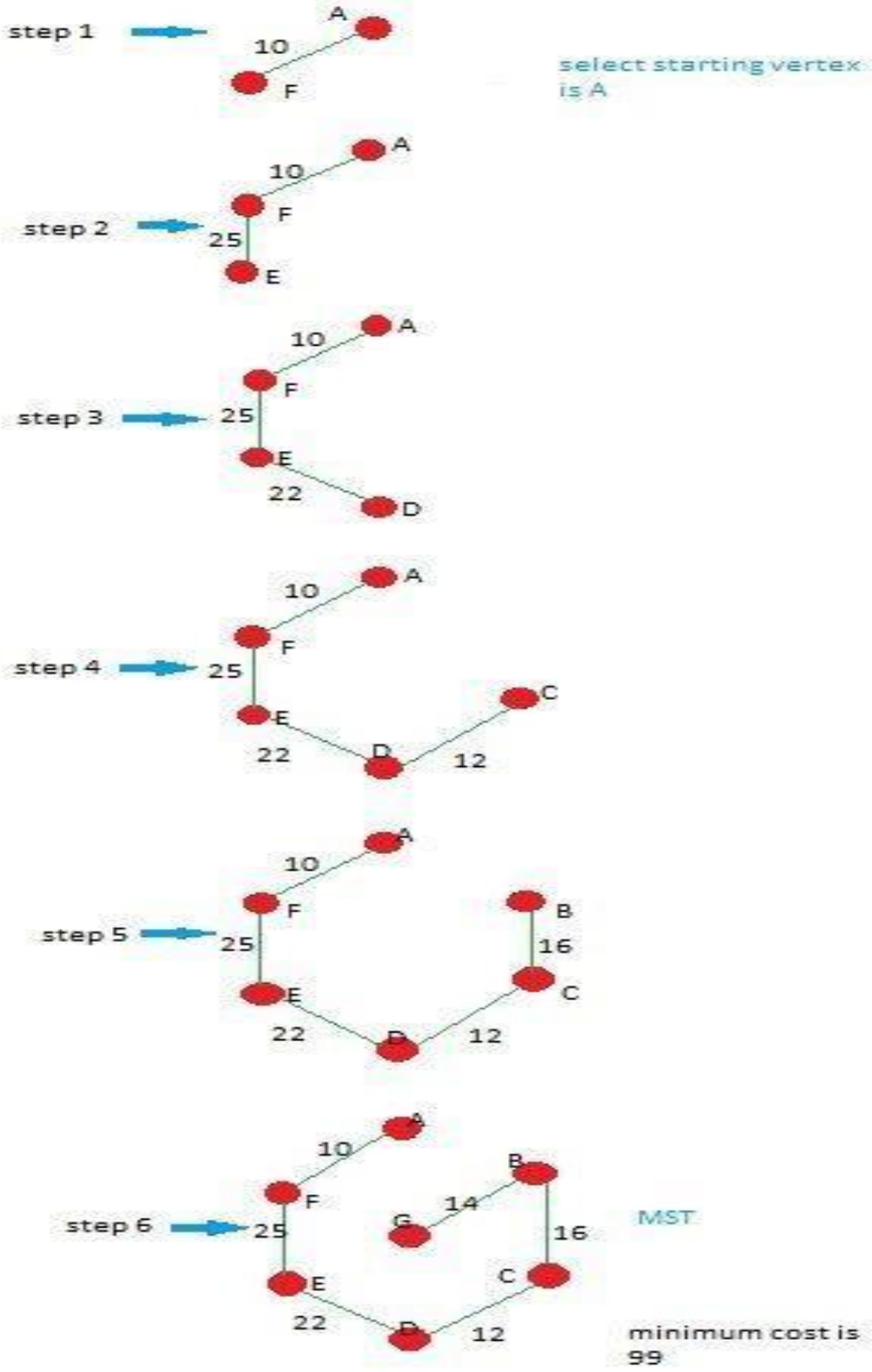
1. Prim's Algorithm
2. Kruskal's Algorithm

Spanning Tree Applications

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

Minimum Spanning tree Applications

- To find paths in the map
- To design networks like telecommunication networks, water supply networks, and electrical grids.


step 1
A
10
F
select starting vertex is A
step 2
A
10
F
25
E
step 3
A
10
F
25
E
22
D
step 4
A
10
F
25
E
22
D
12
C
step 5
A
10
F
25
E
22
D
12
C
B
16
step 6
A
10
F
25
G
14
B
16
E
22
D
12
C
MST
minimum cost is 99

Here we look that the cost of the minimum spanning tree is 99 and the number of edges in minimum spanning tree is 6.

In above diagram we take alphabet A,B,C,D,E,F,G for vertex which is similiar to 0,1,2,3,4,5,6 for vertex

Algorithm:

Step 1: S={ }; //initialize spanning tree set with NULL

Step 2: P={starting vertex}; //contain the starting vertex

Step 3: Visit[ ] ; //initialize the visit array to false

Step 4: Visit[starting vertex]=true ; //make starting vertex visit true

Step 5: While(P!=V) //loop until all the vertex is not visited

Select the least cost edge(u,v) where u belongs to P and v belongs to V-P ;

Visit[v]=true ;

S=S U {(u,v)} ;

P=P U {v} ;

Program:

```
#include <iostream>
#include<bits/stdc++.h>
#include <cstring>
    // number of vertices in graph
#define V 7
    // create a 2d array of size 7x7
```

```
	//for adjacency matrix to represent graph
int main () {
	// create a 2d array of size 7x7
	//for adjacency matrix to represent graph
int G[V][V] = {
{0,28,0,0,0,10,0},
{28,0,16,0,0,0,14},
{0,16,0,12,0,0,0},
{0,0,12,22,0,18},
{0,0,0,22,0,25,24},
{10,0,0,0,25,0,0},
{0,14,0,18,24,0,0}
};
int edge; // number of edge
	// create an array to check visited vertex
int visit[V];
	//initialise the visit array to false
for(int i=0;i<V;i++)
	{
		visit[i]=false;
	}
	// set number of edge to 0
edge = 0;
	// the number of edges in minimum spanning tree will be
	// always less than (V -1), where V is the number of vertices in
	//graph
```

```
        // choose 0th vertex and make it true
visit[0] = true;
int x; // row number
int y; // col number
        // print for edge and weight
cout << "Edge" << " : " << "Weight";
cout << endl;
while (edge < V - 1) {//in spanning tree consist the V-1 number of edges
        //For every vertex in the set S, find the all adjacent vertices
        // , calculate the distance from the vertex selected.
        // if the vertex is already visited, discard it otherwise
        //choose another vertex nearest to selected vertex.
int min = INT_MAX;
x = 0;
y = 0;
for (int i = 0; i < V; i++)
{
        if (visit[i])
        {
                for (int j = 0; j < V; j++)
                {
                        if (!visit[j] && G[i][j])
                        { // not in selected and there is an edge
                                if (min > G[i][j])
                                {
                                        min = G[i][j];
```

```
                                x = i;
                                y = j;
                            }
                        }
                    }
                }
cout << x << " ---> " << y << " : " << G[x][y];
cout << endl;
visit[y] = true;
edge++;
}
return 0;
}
```

Output:

```
Edge : Weight
0 ---> 5 : 10
5 ---> 4 : 25
4 ---> 3 : 22
3 ---> 2 : 12
2 ---> 1 : 16
1 ---> 6 : 14
```

Result:

Thus a C++ program has been written and executed for implementing minimum spanning tree.