- **Graph.h**

```
#ifndef GRAPH_H_
#define GRAPH_H_
struct Edge{
	int u,v,wt;
};
class Graph {
	Edge edge[20];
	int weight[20][20];
	int dist[20];
	int path[20];
	int vn,en;
	std::string str[20];
public:
	Graph();
	void createGrapgh();
	void displayGraph();
	void prims();
	void kruskals();
	void sort();
	virtual ~Graph();
};

#endif /* GRAPH_H_ */
```

- **Graph.cpp**

```
#include<iostream>
#include "Graph.h"
using namespace std;
Graph::Graph() {
	// TODO Auto-generated constructor stub
	do{
		cout<<"Enter number of vertice:";
		cin>>vn;
		cout<<"Enter number of edges:";
		cin>>en;
	}while(vn<1 || vn>20 || en<1 ||en>20);

	for(int i=0;i<vn;i++)
	{
		for(int j=0;j<en;j++)
		{
			weight[i][j]=0;
		}
	}

}
//==============definition of create graph==========================
void Graph::createGrapgh(){
	int a,b,w;
	cout<<"Enter department for following vartices:"<<endl;
	for(int i=0;i<vn;i++)
	{
		cout<<"Vertex "<<i<<":";
```

```
            cin>>str[i];
        }
        cout<<"=============================================================="<<e
ndl;
        cout<<"You entered:"<<endl;
        for(int i=0;i<vn;i++)
            cout<<i<<":"<<str[i]<<endl;
        cout<<"=============================================================="<<e
ndl;
        cout<<"Enter edges in graph::"<<endl;
        for(int i=0;i<en;i++){
            cout<<"Enter verices and weight:";
            cin>>a>>b>>w;
            edge[i].u=a;
            edge[i].v=b;
            edge[i].wt=w;
            weight[a][b]=w;
            weight[b][a]=w;
        }
}
//========prims algorithm ======================================
void Graph::prims(){
        int totalVisited=0;
        int visited[vn];
        //initialize default values
        for(int i=0;i<vn;i++){
            dist[i]=5000;
            visited[i]=0;
            path[i]=0;
```

```
}
//display initial contents
cout<<"Vertex\tDist\tVisited";
for(int i=0;i<vn;i++){
	cout<<endl;
	cout<<i<<"\t";
	cout<<dist[i]<<"\t";
	cout<<visited[i]<<" ";
}
//start from 0th vertex
int current=0;
visited[0]=1;
totalVisited=1;
dist[current]=0;

//repeate till all verices are visited
while(totalVisited!=vn){

	cout<<endl<<"=========================================="<<endl;
	cout<<"Current Vertex:"<<current<<endl;
	cout<<"Total Visited:"<<totalVisited<<endl;
	//find distance from current vertex to all other connected vertices which are not visited
	for(int i=0;i<vn;i++){
		if(weight[current][i]!=0){
			if(visited[i]==0)
			{
				//if current distance is smaller than previos, replace
				if(weight[current][i]<dist[i]){
					dist[i]=weight[current][i];
```

```
                                path[i]=current;
                        }
                }
        }
}
//display distance from current to all other verices
cout<<"From"<<current<<endl;
cout<<"Vertex\tDist\tVisited";
for(int i=0;i<vn;i++){
        cout<<endl;
        cout<<i<<"\t";
        cout<<dist[i]<<"\t";
        cout<<visited[i]<<" ";
}
int minCost=32767;
//find minimum distance from available
for(int i=0;i<vn;i++){
        if(visited[i]==0){
                if(dist[i]<minCost){
                        minCost=dist[i];
                        current=i;
                }
        }
}

//marks the visited of current as 1
visited[current]=1;
totalVisited++;
//display selected vertex and its cost
```

```
            cout<<endl<<"Selected vertex:"<<current<<endl;
            cout<<"Mincost:"<<minCost<<endl;

            cout<<"Vertex\tDist\tVisited";
            for(int i=0;i<vn;i++){
                  cout<<endl;
                  cout<<i<<"\t";
                  cout<<dist[i]<<"\t";
                  cout<<visited[i]<<" ";
            }
      }
      //display mst
      int cost=0;
      cout<<endl<<"============================================================";
      cout<<endl<<"Minimum Spanning tree is:"<<endl;
      cout<<"Department\tPath\tDistance"<<endl;
      for(int i=0;i<vn;i++)
      {
            cout<<endl;
            cout<<str[i]<<"\t\t";
            cout<<str[path[i]]<<"\t";
            cout<<dist[i]<<"\t";
            cost+=dist[i];
      }
      cout<<endl<<"Total Cost="<<cost<<endl;
}
//=========kruskals algorithm=====================================
void Graph::kruskals(){
      cout<<"Eges are:"<<endl;
```

```
        cout<<"U V Weight"<<endl;
        for(int i=0;i<en;i++)
                cout<<edge[i].u<<" "<<edge[i].v<<" "<<edge[i].wt<<endl;
        //cout<<"After sorting:"<<endl;
        sort();
        int connt[en];
        int val=1,s,l;
        int cnt=0;
        int j=0;
        Edge temp[20];
cout<<endl<<"============================================================="<<endl
;
while(cnt<en-1 && j<en){
        //if both vertices are not visited
                if(connt[edge[j].u]==0 && connt[edge[j].v]==0){
                        cout<<"Edge selected:"<<edge[j].u<<" "<<edge[j].v<<" "<<edge[j].wt<<endl;
                        temp[cnt]=edge[j];
                        connt[edge[j].u]=connt[edge[j].v]=val;
                        val++;
                        cnt++;
                }
                //if both vertices have different connection value
                else if(connt[edge[j].u]!=connt[edge[j].v]){
                        cout<<"Edge selected:"<<edge[j].u<<" "<<edge[j].v<<" "<<edge[j].wt<<endl;
                        temp[cnt]=edge[j];
                        //if both vertices are visited
                        if(connt[edge[j].u]!=0 && connt[edge[j].v]!=0){

                                //replace smaller connt value among both with grater one
```

```
                //if first vertex is having less connt value
                if(connt[edge[j].u] < connt[edge[j].v])
                {
                        s=connt[edge[j].u];
                        l=connt[edge[j].v];
                }
                //if connt value of second is less
                else{
                        s=connt[edge[j].v];
                        l=connt[edge[j].u];
                }
                //replace large value with smaller
                for(int i=0;i<en;i++){
                        if(connt[i]==l)
                                connt[i]=s;
                }
                cnt++;
        }
        //if only first vertex is visited
        else if(connt[edge[j].u]!=0 && connt[edge[j].v]==0){
                connt[edge[j].v]=connt[edge[j].u];
                cnt++;
        }
        //if only second vertex is visited
        else{
                connt[edge[j].u]=connt[edge[j].v];
                cnt++;
        }
}
```

```
                //if both vertices have same connt values, reject it
                else
                        cout<<"Edge Rejected:"<<edge[j].u<<" "<<edge[j].v<<" "<<edge[j].wt<<endl;
                j++;
        }
        cout<<endl<<"============================================================="
<<endl;
        cout<<"Minimum spanning tree with kruskal's algorithm:"<<endl;
        int cost=0;
        cout<<"Dept1\tDept2\tWeight"<<endl;
        for(int i=0;i<cnt;i++){
                cout<<str[temp[i].u]<<"\t"<<str[temp[i].v]<<"\t"<<temp[i].wt<<endl;
                cost+=temp[i].wt;
        }
        cout<<endl<<"Total cost:"<<cost;
}
//========definition of sort=======================================
void Graph::sort(){
        bool swapped=false;
        for(int i=0;i<en;i++){
                for(int j=0;j<en-i-1;j++){
                        if(edge[j].wt > edge[j+1].wt){
                                swap(edge[j],edge[j+1]);
                                swapped=true;
                        }
                }
                if(!swapped)
                        break;
        }
```

```
}
//=======display graph============================================
void Graph::displayGraph(){
	cout<<endl<<"Matrix is:"<<endl;
	for(int i=0;i<vn;i++){
		cout<<" ";
		for(int j=0;j<vn;j++){
			cout<<weight[i][j]<<" ";
		}
		cout<<endl;
	}
}
Graph::~Graph() {
	// TODO Auto-generated destructor stub
}
```

- **Assignment7.cpp**

```
//============================================================================
// Name        : Assignmet7.cpp
// Author      : Megha Sonavane
// Description : Minimum Spanning Tree
//============================================================================

#include <iostream>
#include "Graph.h"
using namespace std;

int main() {
	cout<<"***Minimum spanning tree***"<<endl;
	Graph g;
	g.createGrapgh();
	g.displayGraph();
	int ch;
	do{
		cout<<endl<<"1:MST with Prim's Algoritm"<<endl<<"2:MST with Kruskal's
Algorithm"<<endl<<"3:Display Original Graph"<<"0:Exit"<<endl;
		cout<<"Enter choice:";
		cin>>ch;
		switch(ch){
		case 1:
			g.prims();
			break;
		case 2:
			g.kruskals();
			break;
```

```
            case 3:
                g.displayGraph();
                break;
            case 0:
                cout<<"Thank You..";
                break;
            default:
                cout<<"Invalid choice.."<<endl;
            }
        }while(ch!=0);
        return 0;
}
```

- **Output**

Minimum spanning tree
Enter number of vertice:5
Enter number of edges:7
Enter department for following vartices:
Vertex 0:Comp
Vertex 1:IT
Vertex 2:ETC
Vertex 3:Libary
Vertex 4:office
==========
You entered:
0:Comp
1:IT
2:ETC
3:Libary
4:office
==========
Enter edges in graph::
Enter verices and weight:0
1
7
Enter verices and weight:1
2
5
Enter verices and weight:2
3
2
Enter verices and weight:3

```
4
12
Enter verices and weight:4
0
9
Enter verices and weight:1
4
3
Enter verices and weight:2
4
1

Matrix is:
 0 7 0 0 9
 7 0 5 0 3
 0 5 0 2 1
 0 0 2 0 12
 9 3 1 12 0

1:MST with Prim's Algoritm
2:MST with Kruskal's Algorithm
3:Display Original Graph0:Exit
Enter choice:1
Vertex Dist  Visited
0       5000  0
1       5000  0
2       5000  0
3       5000  0
4       5000  0
```

```
========================================
Current Vertex:0
Total Visited:1
From0
Vertex Dist  Visited
0       0       1
1       7       0
2       5000    0
3       5000    0
4       9       0
Selected vertex:1
Mincost:7
Vertex Dist  Visited
0       0       1
1       7       1
2       5000    0
3       5000    0
4       9       0
========================================
Current Vertex:1
Total Visited:2
From1
Vertex Dist  Visited
0       0       1
1       7       1
2       5       0
3       5000    0
4       3       0
Selected vertex:4
```

Mincost:3
Vertex Dist Visited
0 0 1
1 7 1
2 5 0
3 5000 0
4 3 1
==========
Current Vertex:4
Total Visited:3
From4
Vertex Dist Visited
0 0 1
1 7 1
2 1 0
3 12 0
4 3 1
Selected vertex:2
Mincost:1
Vertex Dist Visited
0 0 1
1 7 1
2 1 1
3 12 0
4 3 1
==========
Current Vertex:2
Total Visited:4
From2

```
Vertex Dist  Visited
0       0       1
1       7       1
2       1       1
3       2       0
4       3       1
Selected vertex:3
Mincost:2
Vertex Dist  Visited
0       0       1
1       7       1
2       1       1
3       2       1
4       3       1
==========================================================
Minimum Spanning tree is:
Department  Path   Distance

Comp        Comp        0
IT          Comp        7
ETC         office      1
Libary      ETC         2
office      IT          3
Total Cost=13

1:MST with Prim's Algoritm
2:MST with Kruskal's Algorithm
3:Display Original Graph0:Exit
Enter choice:2
```

Eges are:
U V Weight
0 1 7
1 2 5
2 3 2
3 4 12
4 0 9
1 4 3
2 4 1

==========

Edge selected:2 4 1
Edge selected:2 3 2
Edge selected:1 4 3
Edge Rejected:1 2 5
Edge selected:0 1 7
Edge Rejected:4 0 9
Edge Rejected:3 4 12

==========

Minimum spanning tree with kruskal's algorithm:

Dept1	Dept2	Weight
ETC	office	1
ETC	Libary	2
IT	office	3
Comp	IT	7

Total cost:13
1:MST with Prim's Algoritm

2:MST with Kruskal's Algorithm
3:Display Original Graph
0:Exit
Enter choice:0
Thank You..