ASSIGNMENT 5

Aim :

You have a business with several offices; you want to lease phone lines to connect them up with each other and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Objective :

To understand the concept of minimum spanning tree and finding the minimum cost of tree using Kruskals algorithm.

Theory :

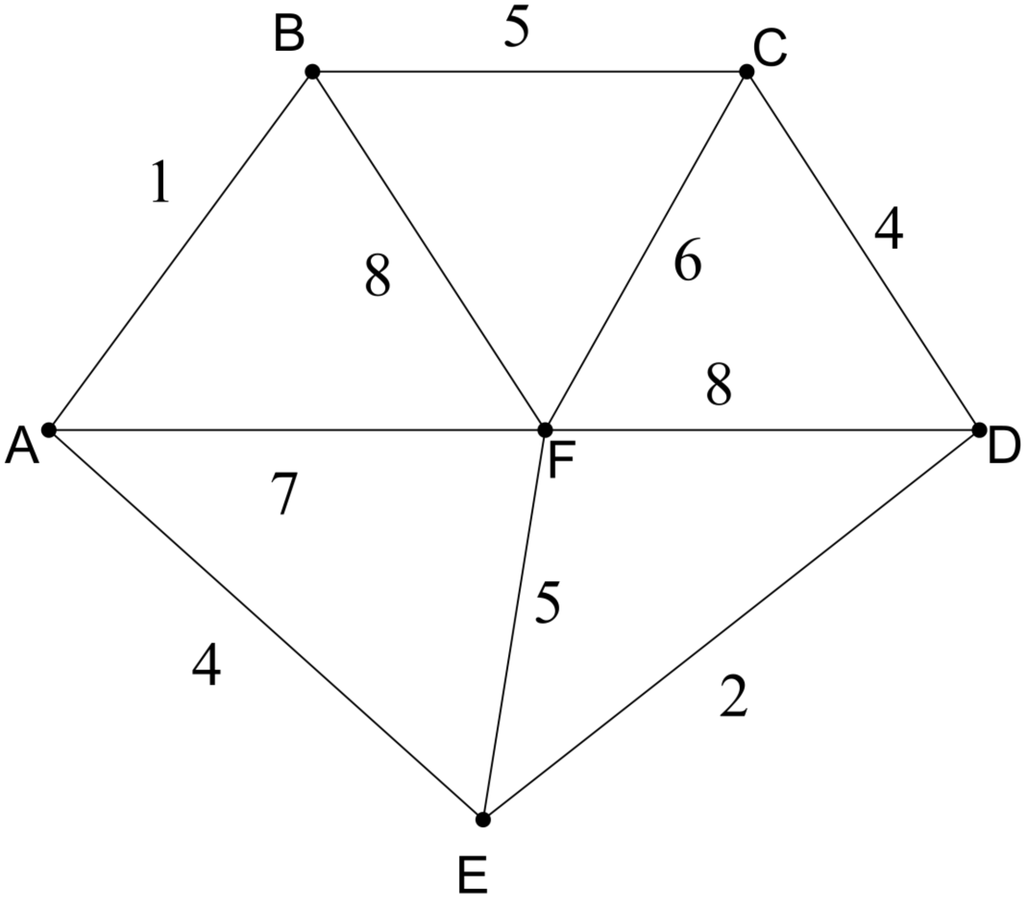
A spanning tree of the graph is a connected (if there is at least one path between every pair of vertices in a graph) subgraph in which there are no cycle. Suppose you have a connected undirected graph with a weight (or cost) associated with each edge. The cost of a spanning tree would be the sum of the costs of its edges. A minimum-cost spanning tree is a spanning tree that has the lowest cost. There are two basic algorithms for finding minimum-cost spanning trees: 1. Prim's Algorithm 2. Kruskal's Algorithm .

Kruskals's algorithm: It tarts with no nodes or edges in the spanning tree, and repeatedly add the cheapest edge that does not create a cycle.

Steps of Kruskal's Algorithm to find minimum spanning tree:

1. Select the shortest edge in a network
2. Select the next shortest edge which does not create a cycle
3. Repeat step 2 untill spanning tree has n-1 edges.

Example:



The solution is

AB	1
ED	2
CD	4
AE	4
EF	5

Total weight of tree: 16

Algorithm:

- Algorithm kruskal(G,V,E,T)

{

```
 1.Sort E in increasing order of weight
 2.let G=(V,E) and T=(A,B),A=V ,B is null
   set and let n =count(V)
3.Initialize n set ,each containing a different element of v.
4.while(|B|<n-1) do
     begin
      e=<u,v>the shortest edge not yet considered
      U=Member(u)
       V=Member(v)
     if( Union(U,V))
         update in B and add the cost
      } }
   end
5.T is the minimum spanning tree
}
```

Program code:

```
#include <iostream>


using namespace std;


class Edge{
private:
   int startNode,endNode,weight;
public:
   Edge(){};
```

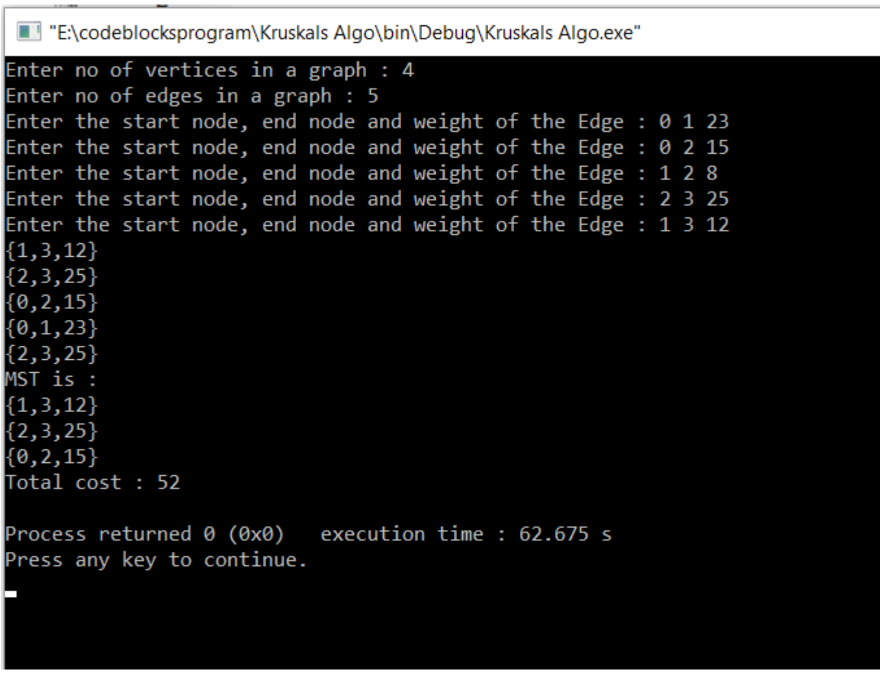
```
    Edge(int sn,int en,int w){
        startNode = sn;
        endNode = en;
        weight =w;
    }

    Edge(const Edge& e){
        startNode = e.startNode;
        endNode = e.endNode;
        weight = e.weight;
    }
    void setParameters(int sn,int en,int w){
        startNode = sn;
        endNode = en;
        weight =w;
    }
    int getStartNode(){return startNode;}
    int getEndNode(){return endNode;}
    int getWeight(){return weight;}
};

class Graph{
private:
    Edge* edgeArr;
    bool* vertices;
    int noOfVertices,noOfEdges;
```

```
public:
    Graph(){};
    Graph(int noOfVertices,int noOfEdges){
        edgeArr = new Edge[noOfEdges];
        vertices = new bool[noOfVertices];
        this->noOfEdges =noOfEdges;
        this->noOfVertices = noOfVertices;
        for(int i=0;i<noOfVertices;i++){
            vertices[i] = false;
        }
    }

    void initiateGraph(int noOfVertices,int noOfEdges){
        edgeArr = new Edge[noOfEdges];
        vertices = new bool[noOfVertices];
        this->noOfEdges =noOfEdges;
        this->noOfVertices = noOfVertices;
        for(int i=0;i<noOfVertices;i++){
            vertices[i] = false;
        }
    }

    Edge* getEdgeArr(){return edgeArr;}
    bool* getVertices(){return vertices;}
    int getNoOfVertices(){return noOfVertices;}
    int getNoOfEdges(){return noOfEdges;}
```

```
};



Graph inGraph;

Graph MST;

int totCost = 0;


void createGraph(){

    cout<<"Enter no of vertices in a graph : ";

    int noOfVertices;

    cin>>noOfVertices;

    cout<<"Enter no of edges in a graph : ";

    int noOfEdges;

    cin>>noOfEdges;

    inGraph.initiateGraph(noOfVertices,noOfEdges);

    for(int i =0; i<noOfEdges;i++){

        cout<<"Enter the start node, end node and weight of the Edge : ";

        int sN,eN,w;

        cin>>sN>>eN>>w;

        inGraph.getEdgeArr()[i].setParameters(sN, eN, w);

    }

}


void displayGraph(Graph graph){

    for(int i=0;i<graph.getNoOfEdges();i++){
```

```
cout<<"{"<<graph.getEdgeArr()[i].getStartNode()<<","<<graph.getEdgeArr()[i].getEndNode()
<<","<<graph.getEdgeArr()[i].getWeight()<<"}"<<endl;
    }
}

void graphSort(Graph graph){
    for(int i=1; i< graph.getNoOfEdges(); i++){
        Edge key = graph.getEdgeArr()[i];
        int j;
        for(j =i-1; j>=0; j--){
            if(key.getWeight() < graph.getEdgeArr()[j].getWeight()){
                graph.getEdgeArr()[j+1] = graph.getEdgeArr()[j];
            }
        }
        graph.getEdgeArr()[j+1] = key;
    }
}

void createMST(){
    MST.initiateGraph(inGraph.getNoOfVertices(), inGraph.getNoOfVertices()-1);
    int j = 0;
    for(int i=0;i< inGraph.getNoOfEdges(); i++){
        if(inGraph.getVertices()[inGraph.getEdgeArr()[i].getStartNode()] == true &&
inGraph.getVertices()[inGraph.getEdgeArr()[i].getEndNode()] == true){}
        else{
            MST.getEdgeArr()[j] = inGraph.getEdgeArr()[i];
            totCost += MST.getEdgeArr()[j].getWeight();

            inGraph.getVertices()[inGraph.getEdgeArr()[i].getStartNode()] = true;

            inGraph.getVertices()[inGraph.getEdgeArr()[i].getEndNode()] = true;

            j++;

        }

    }

}


int main(){

    createGraph();

    graphSort(inGraph);

    displayGraph(inGraph);

    createMST();

    cout<<"MST is :"<<endl;

    displayGraph(MST);

    cout<<"Total cost : "<<totCost<<endl;

}
```

Output:



"E:\codeblocksprogram\Kruskals Algo\bin\Debug\Kruskals Algo.exe"

```
Enter no of vertices in a graph : 4
Enter no of edges in a graph : 5
Enter the start node, end node and weight of the Edge : 0 1 23
Enter the start node, end node and weight of the Edge : 0 2 15
Enter the start node, end node and weight of the Edge : 1 2 8
Enter the start node, end node and weight of the Edge : 2 3 25
Enter the start node, end node and weight of the Edge : 1 3 12
{1,3,12}
{2,3,25}
{0,2,15}
{0,1,23}
{2,3,25}
MST is :
{1,3,12}
{2,3,25}
{0,2,15}
Total cost : 52

Process returned 0 (0x0)   execution time : 62.675 s
Press any key to continue.
```

Conclusion: Kruskal's algorithm can be shown to run in O(**E log E**) time, where E is the number of edges in the graph. Thus we have connected all the offices with a total minimum cost using kruskal’s algorithm.