

ASSIGNMENT NO.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 study the use of kruskal's and prims algorithm in given problem.

Theory:-

What is Minimum Spanning Tree?

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

What are the applications of Minimum Spanning Tree?

See [this](#) for applications of MST.

Below are the steps for finding MST using Kruskal's algorithm

- 1.** Sort all the edges in non-decreasing order of their weight.
- 2.** Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
- 3.** Repeat step#2 until there are $(V-1)$ edges in the spanning tree.

The step#2 uses [Union-Find algorithm](#) to detect cycle.

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Algorithm:-

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty and F is not yet spanning
 - remove an edge with minimum weight from S
 - if the removed edge connects two different trees then add it to the forest F , combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Program Code:-

```
#include <iostream>

#include<iomanip>

using namespace std;

const int MAX=10;

class EdgeList; //forward declaration

class Edge      //USED IN KRUSKAL

{

    int u,v,w;

public:

    Edge(){} //Empty Constructor
```

```
        Edge(int a,int b,int weight)
        {
            u=a;
            v=b;
            w=weight;
        }

        friend class EdgeList;

        friend class PhoneGraph;
};

//---- EdgeList Class -----

class EdgeList
{
    Edge data[MAX];

    int n;

public:

    friend class PhoneGraph;

    EdgeList()
    { n=0;}

    void sort();

    void print();
```

```
};
```

```
//----Bubble Sort for sorting edges in increasing weights' order ---//
```

```
void EdgeList::sort()
```

```
{
```

```
    Edge temp;
```

```
    for(int i=1;i<n;i++)
```

```
        for(int j=0;j<n-1;j++)
```

```
            if(data[j].w>data[j+1].w)
```

```
            {
```

```
                temp=data[j];
```

```
                data[j]=data[j+1];
```

```
                data[j+1]=temp;
```

```
            }
```

```
}
```

```
void EdgeList::print()
```

```
{
```

```
    int cost=0;
```

```
    for(int i=0;i<n;i++)
```

```
    {
```

```
        cout<<"\n"<<i+1<<" "<<data[i].u<<"--"<<data[i].v<<" = "<<data[i].w;
```

```
        cost=cost+data[i].w;
```

```
    }

    cout<<"\nMinimum cost of Telephone Graph = "<<cost;

}

//----- Phone Graph Class-----

class PhoneGraph
{
    int data[MAX][MAX]={0, 28, 0, 0, 0,10,0},
    {28,0,16,0,0,0,14},
    {0,16,0,12,0,0,0},
    {0,0,12,0,22,0,18},
    {0,0,0,22,0,25,24},
    {10,0,0,0,25,0,0},
    {0,14,0,18,24,0,0},
    };

    int n;

public:

    PhoneGraph(int num)

    {

        n=num;

    }
```

```
void readgraph();

void printGraph();

int mincost(int cost[],bool visited[]);

int prim();

void kruskal(EdgeList &spanlist);

int find(int belongs[], int vertexno);

void unionComp(int belongs[], int c1,int c2);

};

void PhoneGraph::readgraph()
{
    cout<<"Enter Adjacency(Cost) Matrix: \n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n; j++)
            cin>>data[i][j];
    }
}

void PhoneGraph::printGraph()
{
    cout<<"\nAdjacency (COST) Matrix: \n";

    for(int i=0;i<n;i++)
```

```
        {  
            for(int j=0;j<n;j++)  
            {  
                cout<<setw(3)<<data[i][j];  
            }  
            cout<<endl;  
        }  
    }  
  
int PhoneGraph::mincost(int cost[],bool visited[]) //finding vertex with minimum  
cost  
{  
    int min=9999,min_index; //initialize min to MAX value(ANY) as temporary  
    for(int i=0;i<n;i++)  
    {  
        if(visited[i]==0 && cost[i]<min)  
        {  
            min=cost[i];  
            min_index=i;  
        }  
    }  
}
```

```
        return min_index; //return index of vertex which is not visited and having  
        minimum cost
```

```
    }
```

```
int PhoneGraph::prim()
```

```
{
```

```
    bool visited[MAX];
```

```
    int parents[MAX]; //storing vertices
```

```
    int cost[MAX]; //saving minimum cost
```

```
    for(int i=0;i<n;i++)
```

```
    {
```

```
        cost[i]=9999; //set cost as infinity/MAX_VALUE
```

```
        visited[i]=0; //initialize visited array to false
```

```
    }
```

```
    cost[0]=0; //starting vertex cost
```

```
    parents[0]=-1; //make first vertex as a root
```

```
    for(int i=0;i<n-1;i++)
```

```
    {
```

```
        int k=mincost(cost,visited); //minimum cost elements index
```



```
        visited[k]=1; //set visited

    for(int j=0;j<n;j++)//for adjacent verices comparision
    {
        if(data[k][j] && visited[j]==0 && data[k][j] < cost[j])
        {
            parents[j]=k;
            cost[j]=data[k][j];
        }
    }
}

cout<<"Minimum Cost Telephone Map:\n";

for(int i=1;i<n;i++)
{
    cout<<i<<" -- "<<parents[i]<<" = "<<cost[i]<<endl;
}

int mincost=0;

for (int i = 1; i < n; i++)

    mincost+=cost[i];                //data[i][parents[i]];

return mincost;

}
```

//----- Kruskal's Algorithm

void PhoneGraph::kruskal(EdgeList &spanlist)

```
{
    int belongs[MAX]; //Separate Components at start (No Edges, Only vertices)

    int cno1,cno2;    //Component 1 & 2

    EdgeList elist;

    for(int i=1;i<n;i++)
        for(int j=0;j<i;j++)
        {
            if(data[i][j]!=0)
            {
                elist.data[elist.n]=Edge(i,j,data[i][j]); //constructor for
initializing edge

                elist.n++;
            }
        }

    elist.sort(); //sorting in increasing weight order

    for(int i=0;i<n;i++)
        belongs[i]=i;

    for(int i=0;i<elist.n;i++)
```

```
        {

            cno1=find(belongs,elist.data[i].u); //find set of u

            cno2=find(belongs,elist.data[i].v); ////find set of v

            if(cno1!=cno2) //if u & v belongs to different sets

            {

                spanlist.data[spanlist.n]=elist.data[i]; //ADD Edge to spanlist

                spanlist.n=spanlist.n+1;

                unionComp(belongs,cno1,cno2); //ADD both components to

same set

            }

        }

    }

void PhoneGraph::unionComp(int belongs[],int c1,int c2)

{

    for(int i=0;i<n;i++)

    {

        if(belongs[i]==c2)

            belongs[i]=c1;

    }

}
```

```
int PhoneGraph::find(int belongs[],int vertexno)
```

```
{  
  
    return belongs[vertexno];  
  
}  
  
//----- MAIN PROGRAM-----  
  
int main() {  
  
    int vertices,choice;  
  
    EdgeList spantree;  
  
    cout<<"Enter Number of cities: ";  
  
    cin>>vertices;  
  
    PhoneGraph p1(vertices);  
  
    //p1.readgraph();  
  
    do  
  
    {  
  
        cout<<"\n1.Find Minimum Total Cost(By Prim's Algorithm)"  
  
            <<"\n2.Find Minimum Total Cost(by Kruskal's  
Algorithms)"  
  
            <<"\n3.Re-Read Graph(INPUT)"  
  
            <<"\n4.Print Graph"  
  
            <<"\n0. Exit"  
  
            <<"\nEnter your choice: ";
```

```
        cin>>choice;

        switch(choice)
        {
        case 1:

                cout<<" Minimum cost of Phone Line to cities is: "<<p1.prim();

                break;

        case 2:

                p1.kruskal(spantree);

                spantree.print();

                break;

        case 3:

                p1.readgraph();

                break;

        case 4:

                p1.printGraph();

                break;

        default:

                cout<<"\nWrong Choice!!!";

        }

    }while(choice!=0);
```

```
        return 0;

    }

/*    Sample INPUT: vertices =7
*        {{0, 28, 0, 0, 0,10,0},
        {28,0,16,0,0,0,14},
        {0,16,0,12,0,0,0},
        {0,0,12,0,22,0,18},
        {0,0,0,22,0,25,24},
        {10,0,0,0,25,0,0},
        {0,14,0,18,24,0,0},
        };

    Minimum Cost: 99

*/
```

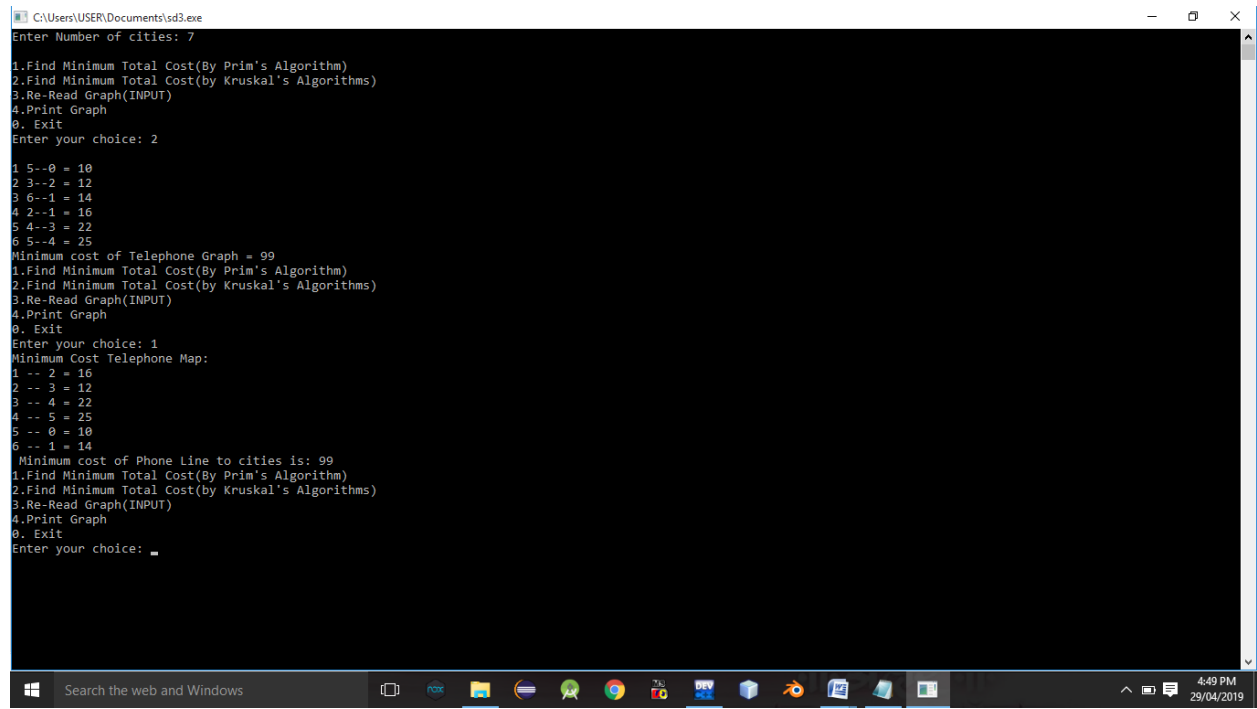
Output Screenshots:-

Skill Development Lab-2 ,2018-19

Name: Omkar Dhaigude

Roll no: 223015

Gr no:17U052



```
CAUsers\USER\Documents\sd3.exe
Enter Number of cities: 7
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 2
1 5--0 = 10
2 3--2 = 12
3 6--1 = 14
4 2--1 = 16
5 4--3 = 22
6 5--4 = 25
Minimum cost of Telephone Graph = 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 1
Minimum Cost Telephone Map:
1 -- 2 = 16
2 -- 3 = 12
3 -- 4 = 22
4 -- 5 = 25
5 -- 0 = 10
6 -- 1 = 14
Minimum cost of Phone Line to cities is: 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: _
```

Conclusion:- Thus,we have studied implementation of kruskal's algorithm.