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:-

You want a set of lines that connects all your offices with a minimum total cost. To Solve the problem by suggesting appropriate data structures

Theory:-

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.[1] It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.[1] This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component).

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.

Code:-

```
        #include <iostream>
using namespace std;
const int MAX=10;
class edge
{
    friend class graph;
    friend class edgelist;
    int u,v,wt;
public:
    edge()
    {}
    edge(int x,int y, int w)
```

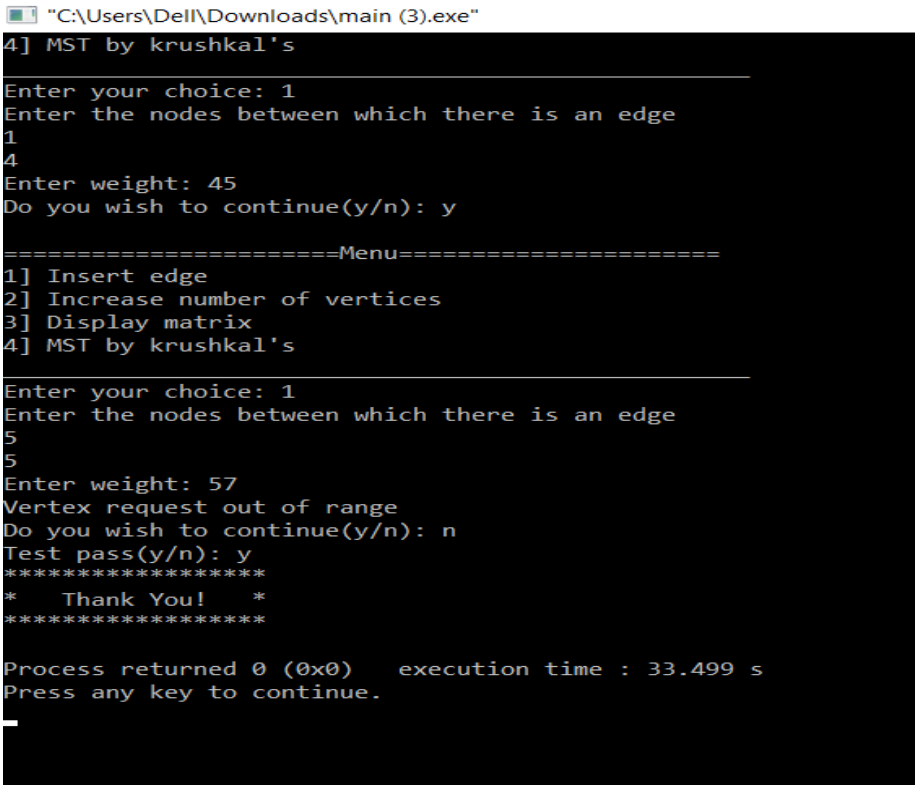
```
    {
        u=x;
        v=y;
        wt=w;
    }
};
class edgelist
{
    friend class graph;
    edge data[MAX];
    int n;
public:
    edgelist()
    {
        n=0;
    }
    void sort()
    {
        edge temp;
        for(int i=0;i<n-1;i++)
        {
            for(int j=0;j<n-i-1;i++)
            {
                if(data[j].wt>data[j].wt)
                {
                    temp=data[j];
                    data[j]=data[j+1];
                    data[j+1]=temp;
                }
            }
        }
    }
    void print()
    {
        cout<<n<<endl;
        int cost=0;
        for(int i=0;i<n;i++)
        {
            cout<<"\n"<<i+1<<" "<<data[i].u<<" --> "<<data[i].v<<" =
"<<data[i].wt;
            cost=cost+data[i].wt;
        }
        cout<<"\nThe minimum cost of the minimum spanning tree is
"<<cost<<endl;
    }
};
class graph
{
    int g[MAX][MAX];
    int v;
public:
    graph()
    {
        for(int i=0;i<v;i++)
```

```
            for(int j=0;j<v;j++)
                g[i][j]=0;
    }
    void insert_edge(int n1,int n2,int wt)
    {
        if(n1-1>=v||n2-1>=v)
            cout<<"Vertex request out of range\n";
        else
        {
            g[n1-1][n2-1]=wt;
            g[n2-1][n1-1]=wt;
        }
    }
    void display()
    {
        for(int i=0;i<v;i++)
        {
            for(int j=0;j<v;j++)
            {
                cout<<g[i][j]<<"\t";
            }
            cout<<endl;
        }
    }
    void update_v(int n)
    {
        v=n;
    }
    void krushkal(edgelist mst)
    {
        edgelist list;
        int belongs[v];
        int c1,c2;
        for(int i=0;i<v;i++)
        {
            for(int j=0;j<v;j++)
            {
                if(g[i][j]!=0)
                {
                    list.data[list.n]=edge(i,j,g[i][j]);
                    list.n++;
                }
            }
        }
        list.sort();
        for(int i=0;i<v;i++)
            belongs[i]=i;
        for(int i=0;i<list.n;i++)
        {
            c1=find(belongs,list.data[i].u);
            c2=find(belongs,list.data[i].v);
            if(c1!=c2)
            {
                mst.data[mst.n]=list.data[i];
```

```
                mst.n++;
                uni(belongs,c1,c2);
            }
        }
        mst.print();
    }
    int find(int belongs[],int x)
    {
        return belongs[x];
    }
    void uni(int belongs[],int c1,int c2)
    {
        for(int i=0;i<v;i++)
        {
            if(belongs[i]==c2)
                belongs[i]=c1;
        }
    }
};
int main()
{
    char r;
    do
    {
        graph g;
        char op;
        int v;
        cout<<"Enter number of vertices: ";
        cin>>v;
        g.update_v(v);
        do
        {
            int c;
            cout<<"\n=======================Menu=======================\n";
            cout<<"1] Insert edge\n2] Increase number of vertices\n3]
Display matrix\n4] MST by krushkal's\n";
            cout<<"__________________________________________________\n";
            cout<<"Enter your choice: ";
            cin>>c;
            switch(c)
            {
                case 1: {
                            int n1,n2,wt;
                            cout<<"Enter the nodes between which there is
an edge\n";
                            cin>>n1>>n2;
                            cout<<"Enter weight: ";
                            cin>>wt;
                            g.insert_edge(n1,n2,wt);
                        }
                        break;
                case 2: {
                            int n;
```

```
                            cout<<"Enter the number by which you wish to
increase the vertices: ";
                            cin>>n;
                            v+=n;
                            g.update_v(v);
                        }
                        break;
                case 3: {
                            g.display();
                        }
                        break;
                case 4: {
                            edgelist mst;
                            g.krushkal(mst);
                        }
                        break;
                default:cout<<"Error 404.....page not found\n";
            }
            cout<<"Do you wish to continue(y/n): ";
            cin>>op;
        }while(op=='y' || op=='Y');
        cout<<"Test pass(y/n): ";
        cin>>r;
    }while(r=='n' || r=='N');
    cout<<"*****************\n";
    cout<<"*   Thank You!   *\n";
    cout<<"*****************\n";
    return 0;
}
```

Output Screenshot:-



Conclusion:-

We Have Solved The Above Given Problem Using Appropriate Algorithm i.e.Kruskal's Minimum Spanning Tree Algorithm.