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 until 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;
#define MAX 30

typedef struct edge
{
    int u,v,w;
}edge;

typedef struct edgelist
{
    edge data[MAX];
    int count;
}edgelist;
```

```
edgelist elist;


int G[MAX][MAX],n;
edgelist spanlist;


void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();


int main()
{
   int i,j;
   cout<<"\nEnter number of city's:";


   cin>>n;
cout<<"\nEnter the adjacency matrix of city ID's:\n";


for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         cin>>G[i][j];


   kruskal();
   print();
```

```
}

void kruskal()
{
    int belongs[MAX],i,j,cno1,cno2;
    elist.count=0;

    for(i=1;i<n;i++)
        for(j=0;j<i;j++)
        {
            if(G[i][j]!=0)
            {
                elist.data[elist.count].u=i;
                elist.data[elist.count].v=j;
                elist.data[elist.count].w=G[i][j];
                elist.count++;
            }
        }

    sort();

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

    spanlist.count=0;
```

```
    for(i=0;i<elist.count;i++)
    {
        cno1=find(belongs,elist.data[i].u);
        cno2=find(belongs,elist.data[i].v);

        if(cno1!=cno2)
        {
            spanlist.data[spanlist.count]=elist.data[i];
            spanlist.count=spanlist.count+1;
            union1(belongs,cno1,cno2);
        }
    }
}

int find(int belongs[],int vertexno)
{
    return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
    int i;

    for(i=0;i<n;i++)
        if(belongs[i]==c2)
            belongs[i]=c1;
```

```
}

void sort()
{
    int i,j;
    edge temp;

    for(i=1;i<elist.count;i++)
        for(j=0;j<elist.count-1;j++)
            if(elist.data[j].w>elist.data[j+1].w)
            {
                temp=elist.data[j];
                elist.data[j]=elist.data[j+1];
                elist.data[j+1]=temp;
            }
}

void print()
{
    int i,cost=0;
    for(i=0;i<spanlist.count;i++)
    {
        cout<<"\n"<<spanlist.data[i].u<<"  "<<spanlist.data[i].v<<"  "<<spanlist.data[i].w;
        cost=cost+spanlist.data[i].w;
    }
    cout<<"\n\nMinimum cost of the telephone lines between the cities:"<<cost<<"\n";
```

}

Output:

Enter number of city's:6

Enter the adjacency matrix of city ID's:

0 3 1 6 0 0

3 0 5 0 3 0

1 5 0 5 6 4

6 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

2 0 1

5 3 2

1 0 3

4 1 3

5 2 4

Minimum cost of the telephone lines between the cities:13

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.