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: Understand the problem statement, determine and implement the data structure suitable for solving above real time example.

Theory:

The above statement can be implemented by **Kruskal's Minimum Spanning Tree.**

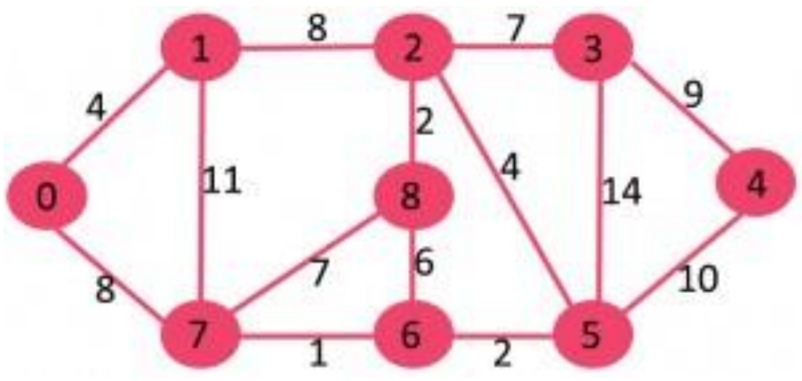
Kruskal's Minimum Spanning Tree:

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. ... 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.

Algorithm:

- KRUSKAL(G):
- A = ∅ For each vertex v ∈ G.V:
- MAKE-SET(v)
- For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
- if FIND-SET(u) ≠ FIND-SET(v):
- A = A ∪ {(u, v)}
- UNION(u, v)
- return A

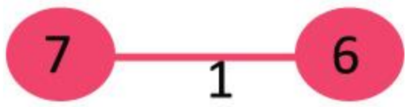
Example:



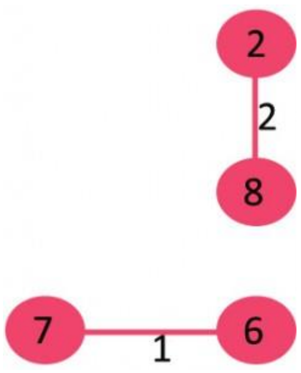
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

Now pick all edges one by one from sorted list of edges

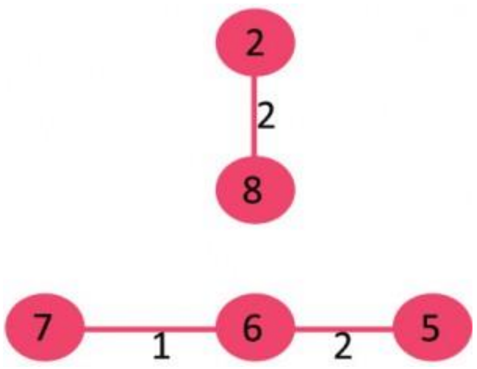
1. *Pick edge 7-6:* No cycle is formed, include it.



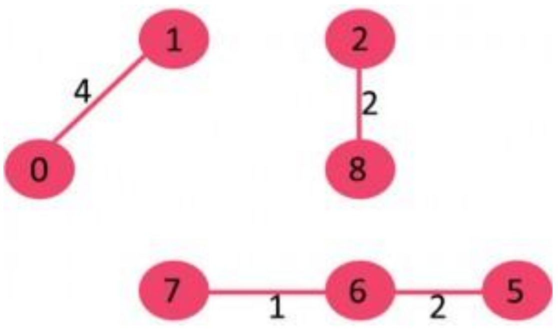
2. *Pick edge 8-2:* No cycle is formed, include it.



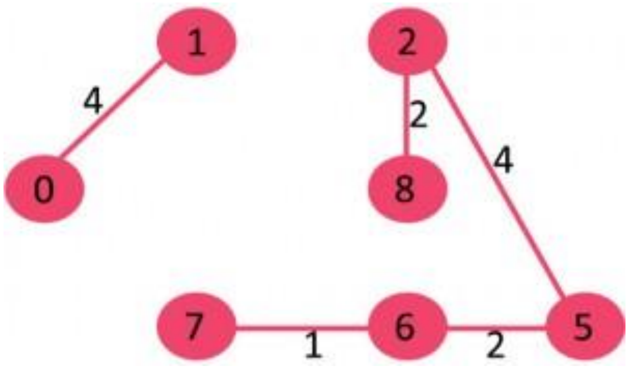
3. *Pick edge 6-5:* No cycle is formed, include it.



4. *Pick edge 0-1:* No cycle is formed, include it.

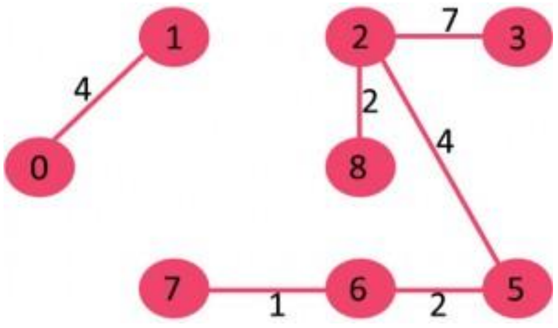


5. *Pick edge 2-5:* No cycle is formed, include it.



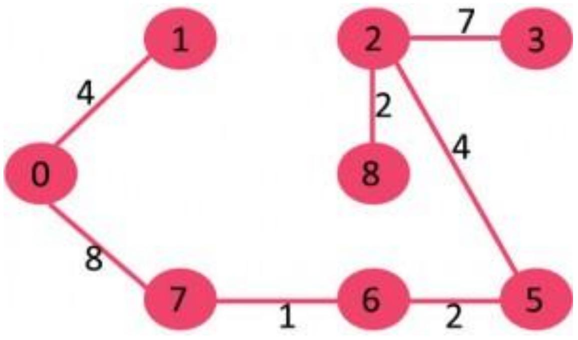
6. *Pick edge 8-6:* Since including this edge results in cycle, discard it.

7. *Pick edge 2-3:* No cycle is formed, include it.



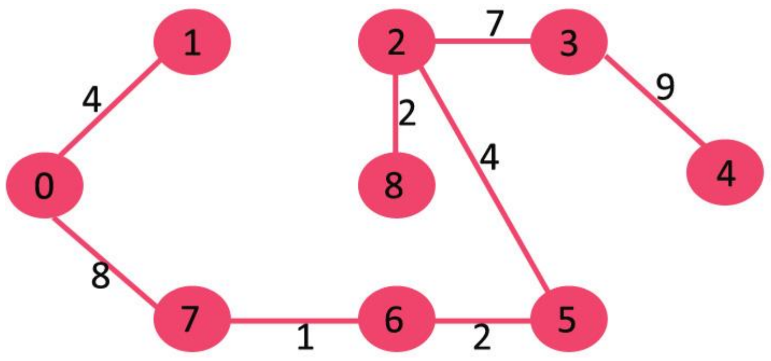
8. *Pick edge 7-8:* Since including this edge results in cycle, discard it.

9. *Pick edge 0-7:* No cycle is formed, include it.



10. *Pick edge 1-2:* Since including this edge results in cycle, discard it.

11. *Pick edge 3-4:* No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

Program:

```
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:

Thus, we learnt about **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. And its application for traversing each node with the shortest distance to reach it.