# LAB-4

#### Find minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm. Time analysis to be done by varying the input graph size.

### Code:

#include <bits/stdc++.h>

#include<iostream>

using namespace std;

int c = 0;

class Edge{

public:

    int src;

    int dest;

    int wt;

};

bool compare(Edge e1,Edge e2){

    return e1.wt<e2.wt;

}

int getParent(int v,int parent[]){

      c++;

    if(parent[v]==v){

        return v;

    }

    return getParent(parent[v],parent);

}

 int main(){

    int n,e,w=0,k=0;

    cout<<"Enter number vertices\n";

    cin>>n;

    e=n\*(n-1);

    Edge edges[e];

    // cout<<"Enter source destination and weight for each edge\n";

    // for(int i=0;i<E;i++){

    //     cin>>edges[i].src>>edges[i].dest>>edges[i].wt;

    // }

    for(int i=0;i<n;i++)

    {

        for(int j=0;j<n-1;j++)

        {

            edges[k].src=i;

            edges[k].dest=j;

            edges[k].wt=1+rand()%100;

            k++;

        }

    }

    //Kruskal algo

    //sorting the edges array in increasing order

    sort(edges,edges+e,compare);

    Edge output[n-1];

    //Union find algorithm to detect cycle

    int parent[n];

    for(int i=0;i<n;i++){

        parent[i]=i;

    }

    int x=0;

    int i=0;

    while(x<n-1){

        c++;

        Edge currentEdge=edges[i];

        int p1=getParent(currentEdge.src,parent);

        int p2=getParent(currentEdge.dest,parent);

        if(p1!=p2){

            output[x]=currentEdge;

            x++;

            parent[p1]=p2;

        }

        i++;

    }

    cout<<"\n"<<endl;

    cout<<"Minimal Spanning Tree"<<endl;

    for(int i=0;i<n-1;i++){

        if(output[i].src<output[i].dest){

            cout<<output[i].src<<" "<<output[i].dest<<" "<<output[i].wt<<endl;

            w = w + output[i].wt;

        }

        else

        { cout<<output[i].dest<<" "<<output[i].src<<" "<<output[i].wt<<endl;

        w = w + output[i].wt;

        }

    }

  cout << "cost of minimal spanning tree " << w << endl;

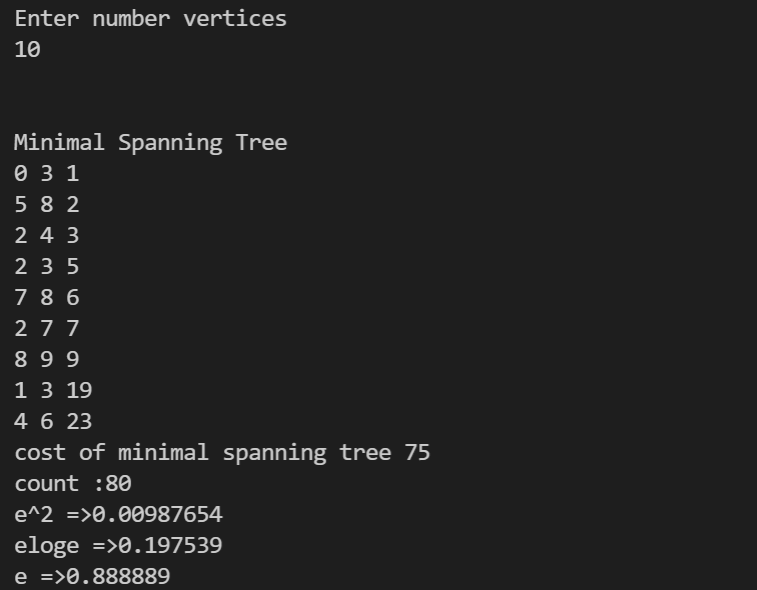
  cout<<"count :"<<c;

  cout<<"\ne^2 =>"<<(float)c/(e\*e)<<"\neloge =>"<<(float)c/(e\*log(e))<<"\ne =>"<<(float)c/(e)<<endl;

    return 0;

}

#### Output:



#### Analysis:

