krushals\_07.cpp

**Compile:** g++ krushals\_07.cpp -o krushals\_07

**Run:** ./krushals\_07

**Program:**

#include <iostream>

#include <fstream>

using namespace std;

classkruskal

{

private:

int n;

intnoe;

intgraph\_edge[100][4];

int tree[10][10];

int sets[100][10];

int top[100];

public:

intread\_graph();

voidinitialize\_span\_t();

voidsort\_edges();

void algorithm();

intfind\_node(int );

voidprint\_min\_span\_t();

};

intkruskal::read\_graph()

{

cout<<"This program implements the kruskal algorithm\n";

cout<<"Enter the no. of nodes in the undirected weighted graph";

cin>>n;

noe=0;

cout<<"Enter the weights for the following edges ::\n";

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

{

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

{

cout<<i<<" , "<<j<<": ";

int w;

cin>>w;

if(w!=0)

{

noe++;

graph\_edge[noe][1]=i;

graph\_edge[noe][2]=j;

graph\_edge[noe][3]=w;

}

}

}

cout<<"\n\nThe edges in the given graph are::\n";

for(inti=1;i<=noe;i++)

{

cout<<" < "<<graph\_edge[i][1]

<<" , "<<graph\_edge[i][2]

<<" > "<<graph\_edge[i][3]<<endl;

}

}

voidkruskal::sort\_edges()

{

for(inti=1;i<=noe-1;i++)

{

for(int j=1;j<=noe-i;j++)

{

if(graph\_edge[j][3]>graph\_edge[j+1][3])

{

int t=graph\_edge[j][1];

graph\_edge[j][1]=graph\_edge[j+1][1];

graph\_edge[j+1][1]=t;

t=graph\_edge[j][2];

graph\_edge[j][2]=graph\_edge[j+1][2];

graph\_edge[j+1][2]=t;

t=graph\_edge[j][3];

graph\_edge[j][3]=graph\_edge[j+1][3];

graph\_edge[j+1][3]=t;

}

}

}

cout<<"\n\nAfter sorting the edges in the given graph are::\n";

for(inti=1;i<=noe;i++)

cout<<""<<graph\_edge[i][1]

<<" , "<<graph\_edge[i][2]

<<" > ::"<<graph\_edge[i][3]<<endl;

}

voidkruskal::algorithm()

{

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

{

sets[i][1]=i;

top[i]=1;

}

cout<<"\nThe algorithm starts ::\n\n";

for(inti=1;i<=noe;i++)

{

int p1=find\_node(graph\_edge[i][1]);

int p2=find\_node(graph\_edge[i][2]);

if(p1!=p2)

{

cout<<"The edge included in the tree is ::"

<<" < "<<graph\_edge[i][1]<<" , "

<<graph\_edge[i][2]<<" > "<<endl<<endl;

tree[graph\_edge[i][1]][graph\_edge[i][2]]=graph\_edge[i][3];

tree[graph\_edge[i][2]][graph\_edge[i][1]]=graph\_edge[i][3];

// Mix the two sets

for(int j=1;j<=top[p2];j++)

{

top[p1]++;

sets[p1][top[p1]]=sets[p2][j];

}

top[p2]=0;

}

else

{

cout<<"Inclusion of the edge "

<<" < "<<graph\_edge[i][1]<<" , "

<<graph\_edge[i][2]<<" > "<<"forms a cycle so it is removed\n\n";

}

}

}

intkruskal::find\_node(int n)

{

for(inti=1;i<=noe;i++)

{

for(int j=1;j<=top[i];j++)

{

if(n==sets[i][j])

returni;

}

}

return -1;

}

voidkruskal::print\_min\_span\_t()

{

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

{

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

cout<<tree[i][j]<<"\t";

cout<<endl;

}

}

int main()

{

kruskalobj;

obj.read\_graph();

obj.sort\_edges();

obj.algorithm();

obj.print\_min\_span\_t();

return 0;

}

**Output:**



