**Assignment No:-**

**Assignment Name**:- **Write a program to find Minimum-Cost Spanning Trees (Kruskal’s Algorithm).**

**Name:- WARKE PURVA DILIP.**

**Roll No:- 136.**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

int COST[7][7]={

{0,0 ,0 ,0,0,0,0},

{0,9999,10,9999,30,45,9999},

{0,10,9999,50,9999,40,25},

{0,9999,50,9999,9999,35,15},

{0,30,9999,9999,9999,9999,20},

{0,45,40,35,9999,9999,55},

{0,9999,25,15,20,55,9999}

};

class GRAPH

{

private:

//int COST[10][10];

int n,NEAR[10],T[10][3],mincost,P[10],EDGE[15][4],noe;

//int n,A[10][10];

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void KRUSKAL();

void CREATE\_ED\_LIST();

void SORT\_ED\_LIST();

void U(int,int);

int F(int);

};

void GRAPH::U(int i,int j)

{

P[i]=j;

}

int GRAPH::F(int i)

{

int j=i;

while(P[j]>0)

{

j=P[j];

}

return j;

}

GRAPH::GRAPH(int par)

{

n=par;

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

P[i]=0;

}

void GRAPH::READ\_GRAPH()

{

cout<<"\nEnter cost matrix : ";

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

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

cin>>COST[i][j];

cout<<endl;

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

{

cout<<endl;

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

cout<<COST[i][j]<<" ";

}

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl;

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

{

cout<<endl;

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

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

}

}

int MIN (int a,int b)

{

if(a<b) return a; else return b;

}

void GRAPH::CREATE\_ED\_LIST()

{

noe=0;

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

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

if(COST[i][j] < 9999)

{

noe=noe+1;

EDGE[noe][1]=i;

EDGE[noe][2]=j;

EDGE[noe][3]=COST[i][j];

}

//cout<<noe;

}

void GRAPH::SORT\_ED\_LIST()

{

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

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

{

if(EDGE[j][3] > EDGE[j+1][3])

{

int temp;

temp=EDGE[j][1];

EDGE[j][1]= EDGE[j+1][1];

EDGE[j+1][1]=temp;

//---------------------------

temp=EDGE[j][2];

EDGE[j][2]= EDGE[j+1][2];

EDGE[j+1][2]=temp;

//---------------------------

temp=EDGE[j][3];

EDGE[j][3]= EDGE[j+1][3];

EDGE[j+1][3]=temp;

}

}

/\* for(i=1;i<=noe;i++)

cout<<EDGE[i][1]<<" "<<EDGE[i][2]<<" "<<EDGE[i][3]<<endl;

\*/

}

void GRAPH::KRUSKAL()

{

int u,v,j,k;

//crate edge list

CREATE\_ED\_LIST();

SORT\_ED\_LIST();

int i=0;

mincost=0;

int ptr=1;

while(i<=n-1 && i<=noe)

{

u= EDGE[ptr][1];

v= EDGE[ptr][2];

j=F(u);

k=F(v);

if(j!=k)

{

i=i+1;

T[i][1]=u;

T[i][2]=v;

mincost=mincost+COST[u][v];

U(j,k);

}

ptr=ptr+1;

}

if(i < n-1)

cout<<"\nNo spanning Tree";

else

{

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

cout<<T[k][1]<<" "<<T[k][2]<<endl;

cout<<"\nCost of spanning Tree = "<<mincost;

}

}

void main()

{

int n;

clrscr();

cout<<"\nEnter no of nodes : ";

cin>>n;

GRAPH obj(n);

//obj.READ\_GRAPH();

obj.KRUSKAL();

obj.SHOW\_GRAPH();

getch();

}