**Md. Asad Chowdhury Dipu**

**ID: 2019-1-60-093**

**Section: 01**

**Ans#1**

#include<stdio.h>

void main()

{

int n,u,v,j,i,d[10],from[10];

int Graph[10][10], span[10][10], c[10][10];

int edges, IsVisited[10], MinDistance, MinCost;

printf("Enter No. of Vertices:");

scanf("%d",&n);

printf("\nEnter The Adjacency Matrix:\n");

for(i=0; i<n; i++)

{

for(j=0; j<n; j++)

{

scanf("%d",&Graph[i][j]);

}

}

for(i=0; i<n; i++)

for(j=0; j<n; j++)

{

if(Graph[i][j]==0)

{

c[i][j]=99999;

}

else

{

c[i][j]=Graph[i][j];

}

span[i][j]=0;

}

d[0]=0;

IsVisited[0]=1;

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

{

d[i]=c[0][i];

from[i]=0;

IsVisited[i]=0;

}

MinCost=0;

edges=n-1;

while(edges>0)

{

MinDistance=99999;

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

if(IsVisited[i] == 0&& d[i]<MinDistance)

{

v=i;

MinDistance=d[i];

}

u=from[v];

span[u][v]=d[v];

span[v][u]=d[v];

edges--;

IsVisited[v]=1;

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

if(IsVisited[i]==0&&c[i][v]<d[i])

{

d[i]=c[i][v];

from[i]=v;

}

MinCost+=c[u][v];

}

printf("\nSpanning tree:\n");

for(i=0; i<n; i++)

{

printf("\n");

for(j=0; j<n; j++)

printf("%d\t",span[i][j]);

}

printf("\n\nTotal Cost : %d",MinCost);

}

**Ans#2**

#include<stdio.h>

#define MAX 20

typedef struct Edge

{

int u,v,w;

}edge;

typedef struct EdgeList

{

edge data[MAX];

int n;

}EdgeList;

EdgeList nlist;

int Graph[MAX][MAX],n;

EdgeList s;

int look(int b[],int v)

{

return(b[v]);

}

void union1(int b[],int c,int d)

{

int i;

for(i=0;i<n;i++)

if(b[i]==d)

b[i]=c;

}

void sort1()

{

int i,j;

edge temp;

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

for(j=0;j<nlist.n-1;j++)

if(nlist.data[j].w>nlist.data[j+1].w)

{

temp=nlist.data[j];

nlist.data[j]=nlist.data[j+1];

nlist.data[j+1]=temp;

}

}

void kruskal()

{

int b[MAX],i,j,p,q;

nlist.n=0;

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

for(j=0;j<i;j++)

{

if(Graph[i][j]!=0)

{

nlist.data[nlist.n].u=i;

nlist.data[nlist.n].v=j;

nlist.data[nlist.n].w=Graph[i][j];

nlist.n++;

}

}

sort1();

for(i=0;i<n;i++)

b[i]=i;

s.n=0;

for(i=0;i<nlist.n;i++)

{

p=look(b,nlist.data[i].u);

q=look(b,nlist.data[i].v);

if(p!=q)

{

s.data[s.n]=nlist.data[i];

s.n=s.n+1;

union1(b,p,q);

}

}

}

int main()

{

int i,j,cost=0;

printf("Enter the number of vertices:");

scanf("%d",&n);

printf("\nEnter The Adjacency Matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&Graph[i][j]);

kruskal();

printf("\nThe Spanning Tree is : ");

for(i=0;i<s.n;i++)

{

printf("\n%d\t%d\t%d",s.data[i].u,s.data[i].v,s.data[i].w);

cost=cost+s.data[i].w;

}

printf("\n\nCost of the panning tree is : %d units",cost);

}

**Ans#3**

#include<bits/stdc++.h>

using namespace std;

struct node

{

int nd,Weight;

}temp;

struct Line

{

int Name,Key;

bool flag;

} A[100];

vector<node> g[100];

int n, dis[100],par[100];

int pop()

{

int min=1000000;

int index;

for(int i=0; i<n; i++)

{

if(A[i].Key < min && A[i].flag==true)

{

min=A[i].Key;

index=i;

}

}

A[index].flag=false;

return index;

}

bool isEmpty()

{

for(int i=0; i<n; i++)

{

if(A[i].flag)

return false;

}

return true;

}

int Dijkstra(int n,int u, int v, int w)

{

while(!isEmpty())

{

int u=pop();

for(int i=0; i<g[u].size(); i++)

{

node t=g[u][i];

int v=t.nd;

int w=t.Weight;

if(dis[v]>dis[u]+w)

{

dis[v]=dis[u]+w;

par[v]=u;

A[v].Key=dis[v];

}

}

}

}

void PrintPath(int parent[],int node,int length)

{

if(node==-1) return;

PrintPath(parent,parent[node],length+1);

cout<<(char)(65+node);

if(length!=0){

cout<<", ";

}

}

int main()

{

int u,v,w,i,edge;

cout<<"Enter number of vertex: ";

cin>>n;

cout<<"Enter number of edges: ";

cin>>edge;

cout<<"Enter edges along with Weight:\n";

for(i=0; i<edge; i++)

{

cin>>u>>v>>w;

temp.nd=v;

temp.Weight=w;

g[u].push\_back(temp);

temp.nd=u;

g[v].push\_back(temp);

}

for(int k=0;k<1;k++){

for(i=0; i<n; i++)

{

dis[i]=99999;

par[i]=-1;

}

dis[k]=0;

for(i=0; i<n; i++)

{

A[i].Name=i;

A[i].Key=dis[i];

A[i].flag=true;

}

cout<<"Source node: "<<(char)(65+k);

cout<<"\nDestination Node Min Cost Min path\n"<<endl;

Dijkstra(n,u,v,w);

for(i=0; i<n; i++)

{

cout<<(char)(65+i)<<" ";

cout<<dis[i]<<" ";

PrintPath(par,i,0);

cout<<endl;

}

}

return 0;

}