



Program Code:-

#include <iostream>

#include <iomanip>

using namespace std;

const int MAX = 10;

class EdgeList; // forward declaration

class Edge // USED IN KRUSKAL

{

int u, v, w;

public:

Edge() {} // Empty Constructor

Edge(int a, int b, int weight)

{

u = a;

v = b;

w = weight;

}

friend class EdgeList;

friend class PhoneGraph;

};

//---- EdgeList Class ----------

class EdgeList

{

Edge data[MAX];

int n;

public:

friend class PhoneGraph;

EdgeList()

{

n = 0;

}

void sort();

void print();

};

//----Bubble Sort for sorting edges in increasing weights' order ---//

void EdgeList::sort()

{

Edge temp;

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

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

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

{

temp = data[j];

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

data[j + 1] = temp;

}

}

void EdgeList::print()

{

int cost = 0;

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

{

cout << "\n"

<< i + 1 << " " << data[i].u << "--" << data[i].v << " = " << data[i].w;

cost = cost + data[i].w;

}

cout << "\nMinimum cost of Telephone Graph = " << cost;

}

//------------ Phone Graph Class---------------

class PhoneGraph

{

int data[MAX][MAX];

int n;

public:

PhoneGraph(int num)

{

n = num;

}

void readgraph();

void printGraph();

int mincost(int cost[], bool visited[]);

int prim();

void kruskal(EdgeList &spanlist);

int find(int belongs[], int vertexno);

void unionComp(int belongs[], int c1, int c2);

};

void PhoneGraph::readgraph()

{

cout << "Enter Adjacency(Cost) Matrix : \n";

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

{

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

cin >> data[i][j];

}

}

void PhoneGraph::printGraph()

{

cout << "\nAdjacency (COST) Matrix : \n";

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

{

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

{

cout << setw(3) << data[i][j];

}

cout << endl;

}

}

int PhoneGraph::mincost(int cost[], bool visited[]) // finding vertex with minimum cost

{

int min = 9999, min\_index; // initialize min to MAX value(ANY) as temporary

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

{

if (visited[i] == 0 && cost[i] < min)

{

min = cost[i];

min\_index = i;

}

}

return min\_index; // return index of vertex which is not visited and having minimum cost

}

int PhoneGraph::prim()

{

bool visited[MAX];

int parents[MAX];

int cost[MAX]; // saving minimum cost

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

{

cost[i] = 9999; // set cost as infinity/MAX\_VALUE

visited[i] = 0; // initialize visited array to false

}

cost[0] = 0; // starting vertex cost

parents[0] = -1; // make first vertex as a root

for (int i = 0; i < n - 1; i++)

{

int k = mincost(cost, visited);

visited[k] = 1;

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

{

if (data[k][j] && visited[j] == 0 && data[k][j] < cost[j])

{

parents[j] = k;

cost[j] = data[k][j];

}

}

}

cout << "Minimum Cost Telephone Map : \n";

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

{

cout << i << " -- " << parents[i] << " = " << cost[i] << endl;

}

int mincost = 0;

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

mincost += cost[i]; // data[i][parents[i]];

return mincost;

}

//------- Kruskal's Algorithm

void PhoneGraph::kruskal(EdgeList &spanlist)

{

int belongs[MAX]; // Separate Components at start (No Edges, Only vertices)

int cno1, cno2; // Component 1 & 2

EdgeList elist;

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

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

{

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

{

elist.data[elist.n] = Edge(i, j, data[i][j]); // constructor for initializing edge

elist.n++;

}

}

elist.sort(); // sorting in increasing weight order

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

belongs[i] = i;

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

{

cno1 = find(belongs, elist.data[i].u); // find set of u

cno2 = find(belongs, elist.data[i].v); ////find set of v

if (cno1 != cno2) // if u & v belongs to different sets

{

spanlist.data[spanlist.n] = elist.data[i]; // ADD Edge to spanlist

spanlist.n = spanlist.n + 1;

unionComp(belongs, cno1, cno2); // ADD both components to same set

}

}

}

void PhoneGraph::unionComp(int belongs[], int c1, int c2)

{

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

{

if (belongs[i] == c2)

belongs[i] = c1;

}

}

int PhoneGraph::find(int belongs[], int vertexno)

{

return belongs[vertexno];

}

//--------- MAIN PROGRAM-----------------------------------

int main()

{

int vertices, choice;

EdgeList spantree;

cout << "Enter Number of cities : ";

cin >> vertices;

PhoneGraph p1(vertices);

p1.readgraph();

do

{

cout << "\n1.Find Minimum Total Cost(By Prim's Algorithm)"

<< "\n2.Find Minimum Total Cost(by Kruskal's Algorithms)"

<< "\n3.Re-Read Graph(INPUT)"

<< "\n4.Print Graph"

<< "\n0. Exit"

<< "\nEnter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << " Minimum cost of Phone Line to cities is : " << p1.prim();

break;

case 2:

p1.kruskal(spantree);

spantree.print();

break;

case 3:

p1.readgraph();

break;

case 4:

p1.printGraph();

break;

default:

cout << "\nWrong Choice!!!";

}

} while (choice != 0);

return 0;

}

Program Output :

Text

Description automatically generated

Text

Description automatically generated