**PUNE INSTITUTE OF COMPUTER TECHNOLOGY DHANKAWADI, PUNE**

**Data Structures And Algorithms(DSA)**

**Assignment No. 08**

**Title : Graph(Shortest Path Algorithm)**

**SE-IT-10**  **ACADEMIC YEAR :- 2020-2021**

**Name :- Diptesh Ravindra Varule Roll No :- 23277**

**Source Code :**

//============================================================================

// Name : dsa\_Assignment8.cpp

// Author : Diptesh Varule

// Version : Updating…..

// Copyright : The Cartel

// Description : Hello World in C++, Ansi-style

//============================================================================

**Source Code :**

**SPA\_Header.h :**

#ifndef SPA\_HEADER\_H\_

#define SPA\_HEADER\_H\_

#include <cstring>

using namespace std;

//Structure AL.

struct AL{

int vertex; //Vertex Number.

int weight; //Weight.

AL\* link; //Link To Another Vertex.

};

//Structure of Edge.

struct Edge

{

int U; //Starting Vertex.

int V; //Ending Vertex.

int wt; //Weight Of The Edge.

};

//Class Graph.

class Graph{

private:

int G[10][10] = {0}; //Adjacent Matrix.

int vertices; //Number of Vertices.

string vname[10]; //Vertices Names.

int edges; //Number of Edges.

AL \*list[10]; //Adajent List.

bool isDirected; //To check for Directed Graph.

Edge edge[50]; //Edge Array.

public:

Graph(); //Constructor.

~Graph(); //Destructor.

void createGraphMat(); //create Graph(Matrix).

void displayGraphMat(); //display Grapg(Matrix).

void createGraphList(); //create Graph List.

void displayGraphList(); //display Graph List.

void dijktrasAlgo(); //Dijktras's Algorithm.

};

AL\* newNode(int i, int wt);

#endif /\* SPA\_HEADER\_H\_ \*/

**SPA\_Impl.cpp :**

#include <iostream>

#include "SPA\_Header.h"

#include <bits/stdc++.h>

using namespace std;

//Constructor

Graph::Graph(){

vertices = 0;

edges = 0;

G[10][10] = {0};

isDirected = false;

}

//Destructor

Graph::~Graph(){}

//Create Graph Using Adjacent Matrix.

void Graph::createGraphMat(){

vertices = 0;

edges = 0;

cout<<"Is The Graph Directed"<<endl;

cout<<"1.True\n2.False"<<endl;

int in = -1;

while(in < 1 || in > 2){

cout<<"Directed : ";

cin>>in;

}

if(in == 1){

isDirected = true;

}

else{

isDirected = false;

}

cout<<"\nEnter The Number Of Land Marks In the Graph : "; //Asking For Vertices.

cin>>vertices;

cout<<endl;

cin.ignore();

for(int i = 0; i < vertices; i++){

cout<<"Enter The Name Of The Land Mark "<<i+1<<" : ";

getline(cin, vname[i]);

}

cout<<endl;

if(isDirected){

int k = 0;

for(int i = 0; i < vertices; i++){

for(int j = 0; j < vertices; j++){ //Traversing Matrix.

if(i == j){

G[i][j] = 0;

continue;

}

cout<<"Enter The Distance Between "<<vname[i]<<" --> "<<vname[j]<<" : "; //Asking For Weight.

int weight;

cin>>weight;

G[i][j] = weight; //Assigning Weight.

if(weight != 0){ //Entering The Edges in Edge Array.

edge[k].U = i;

edge[k].V = j;

edge[k].wt = weight;

k++;

edges++;

}

}

cout<<endl;

}

cout<<"\nAdjacent Matrix Created Successfully";

return;

}

int k = 0;

for(int i = 0; i < vertices; i++){

for(int j = i+1; j < vertices; j++){ //Traversing For Upper Half Matrix.

cout<<"Enter The Distance Between "<<vname[i]<<" And "<<vname[j]<<" : "; //Asking For Weight.

int weight;

cin>>weight;

G[i][j] = G[j][i] = weight; //Assigning Weight.

if(weight != 0){ //Entering The Edges in Edge Array.

edge[k].U = i;

edge[k].V = j;

edge[k].wt = weight;

k++;

edges++;

edge[k].U = j;

edge[k].V = i;

edge[k].wt = weight;

k++;

edges++;

}

}

cout<<endl;

}

cout<<"\nAdjacent Matrix Created Successfully";

}

//Display Graph Using Adjacent Matrix.

void Graph::displayGraphMat(){

if(isDirected){

cout<<endl;

for(int i = 0; i < vertices; i++){

for(int j = 0; j < vertices; j++){//Traversing Matrix.

//Displaying The Edges With it's Weight.

if(G[i][j] != 0){

cout<<"The Distance Between "<<vname[i]<<" And "<<vname[j]<<" Is : "<<G[i][j]<<endl;

}

}

cout<<endl;

}

}

else{

cout<<endl;

for(int i = 0; i < vertices; i++){

for(int j = i+1; j < vertices; j++){//Traversing For Upper Half Matrix.

//Displaying The Edges With it's Weight.

if(G[i][j] != 0){

cout<<"The Distance Between "<<vname[i]<<" And "<<vname[j]<<" Is : "<<G[i][j]<<endl;

}

}

cout<<endl;

}

}

for(int i = 0; i < edges; i++){

cout<<"|"<<vname[edge[i].U]<<"|-->|"<<vname[edge[i].V];

cout<<"|"<<", Weight = "<<edge[i].wt<<endl;

}

cout<<endl;

cout<<"\nDisplaying Adjacency List"<<endl;

createGraphList();

displayGraphList();

}

//Create Graph Using Adjacent List.

void Graph::createGraphList(){

for(int i = 0; i < vertices; i++){

AL\* nev = newNode(i, 0);

list[i] = nev;

}

for(int i = 0; i < edges; i++){

AL\* trav = list[edge[i].U];

while(trav->link != NULL){

trav = trav->link;

}

AL\* nev = newNode(edge[i].V, edge[i].wt);

trav->link = nev;

}

}

//Display Graph Using Adjacent List.

void Graph::displayGraphList(){

cout<<endl;

for(int i = 0; i < vertices; i++){

cout<<"|"<<vname[list[i]->vertex]<<"|";

AL\* trav = list[i]; //Creating trav for Traversal.

while(trav != NULL){

trav = trav->link;

if(trav == NULL){ //If Trav is Null Then Break.

break;

}

cout<<"->|"<<vname[trav->vertex]<<", "<<trav->weight<<"|"; //Display vertex.

}

cout<<endl; //Space.

}

}

//Dijktras's Algorithm.

void Graph::dijktrasAlgo()

{

int v = vertices;

int d[30], visited[30], p[30], weight[10][10];

for(int i = 0; i < 30; i++){ //Initializing distance, path and visited array.

d[i] = 7777;

p[i] = visited[i] = 0;

}

for(int m = 0; m < vertices; m++){ //Creating weight Matrix;

for(int n = 0; n < vertices; n++){

weight[m][n] = G[m][n];

}

}

int i, j, k, min, current; //Traversing Variables.

cout<<endl;

for(int i = 0; i < vertices; i++){

cout<<i<<". "<<vname[i]<<endl;

}

int s;

cout<<endl<<"Enter The source Serial Number : "; //Asking User For Source.

cin>>s;

for (i=0;i<v;i++) //Initializing Visited, DIstance And Path Arrays.

{

visited[i] =0;

if(weight[s][i]!=0)

d[i] = weight[s][i];

p[i]=s;

}

current = s; //Setting Current to source.

visited[current] = 1; //Now current is visited.

d[s] = 0; //Distance to itself is 0.

cout<<"\nVisited, Distance and path Status : "<<endl; //Displaying Distance and Path Status.

for(i = 0; i < v; i++){

cout<<endl<<"i = "<<i<<", Visited["<<i<<"] = "<<visited[i]<<", p["<<i<<"] = "<<p[i]<<", d["<<i<<"] = "<<d[i];

}

for(i = 0; i < v-2; i++) //Outer loop for Traversing Till v-2 times.

{

min = 7777; //Assigning Minimum to 7777(Infinity).

for(j = 0; j < v; j++) //Inner Loop For Finding Minimum Non Visited Edge.

{

if (min > d[j] && visited[j] == 0) //If Found then Assigning Current to j.

{

min = d[j];

current = j;

}

}

cout<<endl<<"Selected Vertex is : "<<current; //Displaying Selected Vertex.

cout<<endl<<"Minimum is : "<<min<<endl; //Dispaying Minimum Edge.

visited[current] = 1; //Now Current Is Visited.

for(k=0;k<v;k++) //Loop For Finding minimum Path.

{

if(visited[k] == 0 && (d[current] + weight[current][k]) < d[k]) //Checking For New Unvisited Edge To create path.

{

if(weight[current][k] != 0) //Checking If Edge Is Not Present.

{

d[k] = d[current] + weight[current][k]; //Updating The Total Distance.

p[k] = current; //Assigning The New Path.

}

}

}

cout<<"\nVisited, Distance and path Status : "<<endl; //Displaying Distance and Path Status.

for(int m = 0; m < v; m++){

cout<<endl<<"i = "<<m<<", Visited["<<m<<"] = "<<visited[m]<<", p["<<m<<"] = "<<p[m]<<", d["<<m<<"] = "<<d[m];

}

}//End Of Outer Loop.

cout<<endl<<"------------------------------------------------------";

cout<<endl<<"\nShortest Path from Source to all Destinations";

cout<<endl<<"Source is -->"<<vname[s];

for(i=0;i<v;i++) //Displaying All the Shortest Paths.

{

if(i != s) //If Not Source.

{

cout<<endl<<"Vertex = "<<vname[i]<<", Distance = "<<d[i];

cout<<"\nPath = "<<vname[i];

j = i;

do //Traversing Till Source is Reachead.

{

j = p[j];

cout<<" <- "<<vname[j];

}while(j != s); //If Not Source.

}

cout<<endl;

}

cout<<endl<<"------------------------------------------------------";

}

//Function to create new AL Node.

AL\* newNode(int i, int wt){

AL \*temp = new AL;

temp->vertex = i;

temp->weight = wt;

temp->link = NULL;

return temp;

}

**SPA.cpp :**

#include <iostream>

#include "SPA\_Header.h"

using namespace std;

int main() {

Graph g; //Creating Graph.

int ch; //Backend-Choice.

do{

//User Guide.

cout<<"\n=============================================\n";

cout<<"\n1.Input Graph \n2.Display Graph \n3.Dijkstras's Algorithm";

cout<<"\n0.Exit \n";

cout<<"\nEnter Choice : ";

cin>>ch;

cout<<"\n=============================================\n";

switch(ch){

case 1: //Creating Graph.

g.createGraphMat();

break;

case 2: //Displaying Graph.

g.displayGraphMat();

break;

case 3: //Displaying MST Using Prim's Algorithm.

g.dijktrasAlgo();

break;

case 0: //Ending The Program.

break;

default://Handling Invalid Input.

cout<<"\n\tINVALID OPTION !!!";

break;

}

}while(ch != 0);

cout << "\nProgram End" << endl; //Prints Program End

return 0;

}

**Output :**

=============================================

1.Input Graph

2.Display Graph

3.Dijkstras's Algorithm

0.Exit

Enter Choice : 1

=============================================

Is The Graph Directed

1.True

2.False

Directed : 1

Enter The Number Of Land Marks In the Graph : 5

Enter The Name Of The Land Mark 1 : Kothrud

Enter The Name Of The Land Mark 2 : Shivaji Nagar

Enter The Name Of The Land Mark 3 : Pimpri Chinchwad

Enter The Name Of The Land Mark 4 : Swar Gate

Enter The Name Of The Land Mark 5 : Katraj

Enter The Distance Between Kothrud --> Shivaji Nagar : 10

Enter The Distance Between Kothrud --> Pimpri Chinchwad : 5

Enter The Distance Between Kothrud --> Swar Gate : 0

Enter The Distance Between Kothrud --> Katraj : 0

Enter The Distance Between Shivaji Nagar --> Kothrud : 0

Enter The Distance Between Shivaji Nagar --> Pimpri Chinchwad : 2

Enter The Distance Between Shivaji Nagar --> Swar Gate : 1

Enter The Distance Between Shivaji Nagar --> Katraj : 0

Enter The Distance Between Pimpri Chinchwad --> Kothrud : 0

Enter The Distance Between Pimpri Chinchwad --> Shivaji Nagar : 3

Enter The Distance Between Pimpri Chinchwad --> Swar Gate : 9

Enter The Distance Between Pimpri Chinchwad --> Katraj : 2

Enter The Distance Between Swar Gate --> Kothrud : 0

Enter The Distance Between Swar Gate --> Shivaji Nagar : 0

Enter The Distance Between Swar Gate --> Pimpri Chinchwad : 0

Enter The Distance Between Swar Gate --> Katraj : 4

Enter The Distance Between Katraj --> Kothrud : 7

Enter The Distance Between Katraj --> Shivaji Nagar : 0

Enter The Distance Between Katraj --> Pimpri Chinchwad : 0

Enter The Distance Between Katraj --> Swar Gate : 6

Adjacent Matrix Created Successfully

=============================================

1.Input Graph

2.Display Graph

3.Dijkstras's Algorithm

0.Exit

Enter Choice : 2

=============================================

The Distance Between Kothrud And Shivaji Nagar Is : 10

The Distance Between Kothrud And Pimpri Chinchwad Is : 5

The Distance Between Shivaji Nagar And Pimpri Chinchwad Is : 2

The Distance Between Shivaji Nagar And Swar Gate Is : 1

The Distance Between Pimpri Chinchwad And Shivaji Nagar Is : 3

The Distance Between Pimpri Chinchwad And Swar Gate Is : 9

The Distance Between Pimpri Chinchwad And Katraj Is : 2

The Distance Between Swar Gate And Katraj Is : 4

The Distance Between Katraj And Kothrud Is : 7

The Distance Between Katraj And Swar Gate Is : 6

|Kothrud|-->|Shivaji Nagar|, Weight = 10

|Kothrud|-->|Pimpri Chinchwad|, Weight = 5

|Shivaji Nagar|-->|Pimpri Chinchwad|, Weight = 2

|Shivaji Nagar|-->|Swar Gate|, Weight = 1

|Pimpri Chinchwad|-->|Shivaji Nagar|, Weight = 3

|Pimpri Chinchwad|-->|Swar Gate|, Weight = 9

|Pimpri Chinchwad|-->|Katraj|, Weight = 2

|Swar Gate|-->|Katraj|, Weight = 4

|Katraj|-->|Kothrud|, Weight = 7

|Katraj|-->|Swar Gate|, Weight = 6

Displaying Adjacency List

|Kothrud|->|Shivaji Nagar, 10|->|Pimpri Chinchwad, 5|

|Shivaji Nagar|->|Pimpri Chinchwad, 2|->|Swar Gate, 1|

|Pimpri Chinchwad|->|Shivaji Nagar, 3|->|Swar Gate, 9|->|Katraj, 2|

|Swar Gate|->|Katraj, 4|

|Katraj|->|Kothrud, 7|->|Swar Gate, 6|

=============================================

1.Input Graph

2.Display Graph

3.Dijkstras's Algorithm

0.Exit

Enter Choice : 3

=============================================

0. Kothrud

1. Shivaji Nagar

2. Pimpri Chinchwad

3. Swar Gate

4. Katraj

Enter The source Serial Number : 0

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 0, p[1] = 0, d[1] = 10

i = 2, Visited[2] = 0, p[2] = 0, d[2] = 5

i = 3, Visited[3] = 0, p[3] = 0, d[3] = 7777

i = 4, Visited[4] = 0, p[4] = 0, d[4] = 7777

Selected Vertex is : 2

Minimum is : 5

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 0, p[1] = 2, d[1] = 8

i = 2, Visited[2] = 1, p[2] = 0, d[2] = 5

i = 3, Visited[3] = 0, p[3] = 2, d[3] = 14

i = 4, Visited[4] = 0, p[4] = 2, d[4] = 7

Selected Vertex is : 4

Minimum is : 7

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 0, p[1] = 2, d[1] = 8

i = 2, Visited[2] = 1, p[2] = 0, d[2] = 5

i = 3, Visited[3] = 0, p[3] = 4, d[3] = 13

i = 4, Visited[4] = 1, p[4] = 2, d[4] = 7

Selected Vertex is : 1

Minimum is : 8

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 1, p[1] = 2, d[1] = 8

i = 2, Visited[2] = 1, p[2] = 0, d[2] = 5

i = 3, Visited[3] = 0, p[3] = 1, d[3] = 9

i = 4, Visited[4] = 1, p[4] = 2, d[4] = 7

------------------------------------------------------

Shortest Path from Source to all Destinations

Source is -->Kothrud

Vertex = Shivaji Nagar, Distance = 8

Path = Shivaji Nagar <- Pimpri Chinchwad <- Kothrud

Vertex = Pimpri Chinchwad, Distance = 5

Path = Pimpri Chinchwad <- Kothrud

Vertex = Swar Gate, Distance = 9

Path = Swar Gate <- Shivaji Nagar <- Pimpri Chinchwad <- Kothrud

Vertex = Katraj, Distance = 7

Path = Katraj <- Pimpri Chinchwad <- Kothrud

------------------------------------------------------

=============================================

1.Input Graph

2.Display Graph

3.Dijkstras's Algorithm

0.Exit

Enter Choice : 00

=============================================

Program End