**IT300 – Design and Analysis of Algorithms**

Lab Assignment – 5

**Name:** Niraj Nandish

**Roll No:** 191IT234

1. Program 1 – Johnson's Algorithm
   1. Code

#include <iostream>

#include <limits.h>

#include <math.h>

#include <stdio.h>

#include <vector>

using namespace std;

int minDistance(vector<int> *dist*, vector<bool> *vis*)

{

int min = INT\_MAX;

int min\_ind;

for (int i = 0; i < 4; i++)

{

if (*vis*[i] == false && *dist*[i] <= min)

{

min = *dist*[i];

min\_ind = i;

}

}

return min\_ind;

}

void dijkstra(vector<vector<int>> *mod\_adj\_mat*, int *u*, int *v*)

{

vector<bool> vis(4, false);

vector<int> dist(4, INT\_MAX);

vector<int> path(4);

dist[*u*] = 0;

for (int i = 0; i < 3; i++)

{

int min\_ind = minDistance(dist, vis);

vis[min\_ind] = true;

for (int j = 0; j < 4; j++)

{

if (!vis[j] && *mod\_adj\_mat*[min\_ind][j] != -1 && dist[min\_ind] + *mod\_adj\_mat*[min\_ind][j] < dist[j] && dist[min\_ind] != INT\_MAX)

{

dist[j] = dist[min\_ind] + *mod\_adj\_mat*[min\_ind][j];

path[j] = min\_ind;

}

}

}

if (dist[*v*] != INT\_MAX)

{

int temp\_arr[4];

int x = 0;

int a = *v*;

temp\_arr[0] = *v*;

x++;

do

{

temp\_arr[x] = path[a];

a = path[a];

x++;

} while (a != *u*);

cout << "Shortest path from vertex " << *u* << " to " << *v* << " is: " << *u*;

for (int z = x - 2; z >= 0; z--)

{

cout << " -> ";

cout << temp\_arr[z];

}

cout << endl;

}

else

{

cout << "Shortest path from vertex " << *u* << " to " << *v* << " is not possible to be found." << endl;

}

}

void dijkstraToAll(vector<vector<int>> *mod\_adj\_mat*, int *u*)

{

for (int i = 0; i < 4; i++)

{

if (i != *u*)

{

dijkstra(*mod\_adj\_mat*, *u*, i);

}

}

}

vector<int> bellmanFord(vector<vector<int>> &*edge\_mat*, int *adj\_mat*[][4])

{

vector<int> dist(5, INT\_MAX);

vector<int> temp;

dist[4] = 0;

for (int i = 0; i < 4; i++)

{

temp.push\_back(4);

temp.push\_back(i);

temp.push\_back(0);

*edge\_mat*.push\_back(temp);

temp.clear();

}

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < *edge\_mat*.size(); j++)

{

if ((dist[*edge\_mat*[j][0]] != INT\_MAX) && (dist[*edge\_mat*[j][0]] + *edge\_mat*[j][2] < dist[*edge\_mat*[j][1]]))

{

dist[*edge\_mat*[j][1]] = dist[*edge\_mat*[j][0]] + *edge\_mat*[j][2];

}

}

}

return dist;

}

void johnson(int *adj\_mat*[][4])

{

vector<vector<int>> edge\_mat;

vector<int> temp(3);

vector<vector<int>> mod\_adj\_mat(4, vector<int>(4, 0));

vector<int> dist;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (*adj\_mat*[i][j] != 0)

{

temp[0] = i;

temp[1] = j;

temp[2] = *adj\_mat*[i][j];

edge\_mat.push\_back(temp);

}

}

}

dist = bellmanFord(edge\_mat, *adj\_mat*);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (*adj\_mat*[i][j] != 0)

{

mod\_adj\_mat[i][j] = *adj\_mat*[i][j] + dist[i] - dist[j];

}

else

{

mod\_adj\_mat[i][j] = -1;

}

}

}

for (int i = 0; i < 4; i++)

{

cout << "SOURCE VERTEX " << i << endl;

dijkstraToAll(mod\_adj\_mat, i);

cout << endl;

}

}

int main()

{

int adj\_mat[][4] = {{0, -5, 2, 3},

{0, 0, 4, 0},

{0, 0, 0, 1},

{0, 0, 0, 0}};

johnson(adj\_mat);

return 0;

}

* 1. Screenshots

Graphical user interface, text

Description automatically generated

1. Program 2 – Floyd-Warshall Algorithm
   1. Code and Screenshots

#include <iostream>

#include <limits.h>

#include <math.h>

#include <stdio.h>

#include <vector>

using namespace std;

void floydWarshall(int *no\_V*, vector<vector<int>> &*dist*, vector<vector<int>> &*next*)

{

for (int k = 0; k < *no\_V*; k++)

{

if (k == 1 || k == 3 || k == 5)

{

cout << "\n\nValues of dist(i,j) at k = " << k << ": " << endl;

for (int m = 0; m < *no\_V*; m++)

{

for (int n = 0; n < *no\_V*; n++)

{

cout << *dist*[m][n] << " ";

}

cout << endl;

}

}

for (int i = 0; i < *no\_V*; i++)

{

for (int j = 0; j < *no\_V*; j++)

{

if (*dist*[i][k] == INT\_MAX || *dist*[k][j] == INT\_MAX)

{

continue;

}

if (*dist*[i][j] > *dist*[i][k] + *dist*[k][j])

{

*dist*[i][j] = *dist*[i][k] + *dist*[k][j];

*next*[i][j] = *next*[i][k];

}

}

}

}

}

vector<int> constrPath(int *u*, int *v*, vector<vector<int>> *next*)

{

vector<int> result;

if (*next*[*u*][*v*] == -1)

{

return {};

}

result.push\_back(*u*);

while (*u* != *v*)

{

*u* = *next*[*u*][*v*];

result.push\_back(*u*);

}

return result;

}

int main()

{

int no\_V, no\_E, source, dest, wt, n;

vector<int> path;

cout << "Enter number of vertices: ";

cin >> no\_V;

cout << "Enter number of edges: ";

cin >> no\_E;

vector<vector<int>> adj\_mat(no\_V, vector<int>(no\_V, INT\_MAX));

vector<vector<int>> dist(no\_V, vector<int>(no\_V));

vector<vector<int>> next(no\_V, vector<int>(no\_V));

for (int i = 1; i <= no\_E; i++)

{

cout << "\nEdge number " << i;

cout << "\nEnter source vertex: ";

cin >> source;

cout << "Enter destination vertex: ";

cin >> dest;

cout << "Enter weight: ";

cin >> wt;

adj\_mat[source][dest] = wt;

adj\_mat[dest][source] = wt;

}

for (int i = 0; i < no\_V; i++)

{

for (int j = 0; j < no\_V; j++)

{

if (i == j)

adj\_mat[i][j] = 0;

}

}

cout << "\n\nInitial values of dist(i,j): " << endl;

for (int i = 0; i < no\_V; i++)

{

for (int j = 0; j < no\_V; j++)

{

dist[i][j] = adj\_mat[i][j];

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

if (adj\_mat[i][j] == INT\_MAX)

{

next[i][j] = -1;

}

else

{

next[i][j] = j;

}

}

cout << endl;

}

floydWarshall(no\_V, dist, next);

cout << "\n\nFinal value of dist(i,j): " << endl;

for (int m = 0; m < no\_V; m++)

{

for (int n = 0; n < no\_V; n++)

{

cout << dist[m][n] << " ";

}

cout << endl;

}

cout << "\n\nEnter source vertex: ";

cin >> source;

cout << "Enter destination vertex: ";

cin >> dest;

path = constrPath(source, dest, next);

cout << "Shortest path from vertex " << source << " to " << dest << " is: " << path[0];

for (int i = 1; i < path.size(); i++)

{

cout << " -> " << path[i];

}

cout << endl;

return 0;

}

Text

Description automatically generatedText

Description automatically generated with low confidence

* 1. How can the output of the Floyd-Warshall algorithm be used to detect the presence of a negativeweight cycle? Explain your answer.

Ans. The output of Floyd-Warshall algorithm can be used to check if a negative-weight cycle is present or not. All we have to do is check the diagonal entries of dist(i,j) matrix. If the value of the diagonal entries is negative, then there exists a negative weight cycle. This means that there is a path from the vertex to itself, hence it has negative weight.