**DAA PRACTICAL**

**Name: Virendra Kashinath Bagul Roll No. 05**

**DIJIKSTRAS ALGORITHM PROGRAM**

#include <iostream>

using namespace std;

#define MAX 100

#define TEMP 0

#define PERM 1

#define infinity 9999

#define NIL -1

void findPath(int s, int v); // function declaration to find the shortest path

void Dijkstra(int s); // function declaration for Dijkstra's algorithm

int min\_temp();

void create\_graph();

int n; // number of vertices

int adj[MAX][MAX]; // adjacency matrix to represent the graph

int pred[MAX]; // array to store predecessors in the shortest path

int pathlength[MAX];

int status[MAX];

int main() {

int s, v; // variables for source and destination vertices

create\_graph();

cout << "Enter the source vertex: ";

cin >> s;

Dijkstra(s); // calling Dijkstra's algorithm to find shortest paths

// start of while loop

while (1) {

cout << "Enter destination vertex (-1 to quit): ";

cin >> v; // taking input for destination vertex

if (v == -1)

break;

if (v < 0 || v >= n) // if the input vertex is out of range, display error message

cout << "This vertex does not exist\n";

else if (v == s)

cout << "Source and destination vertices are the same\n";

else if (pathlength[v] == infinity)

cout << "There is no path from source to destination vertex\n";

else // if a valid destination vertex is entered, find and display the shortest path

findPath(s, v);

}

// end of while loop

return 0;

} // end of main section

void Dijkstra(int s) {

int i, current;

// make all vertices temporary

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

pred[i] = NIL;

pathlength[i] = infinity;

status[i] = TEMP;

}

// make pathlength of source vertex = 0

pathlength[s] = 0;

// start of while loop

while (1) {

// search for temporary vertex for minimum pathlength and make it 'current' vertex

current = min\_temp();

if (current == NIL)

return;

status[current] = PERM;

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

// check for adjacent temporary vertices

if ((adj[current][i] != 0) && (status[i] == TEMP))

if (pathlength[current] + adj[current][i] < pathlength[i]) {

pred[i] = current; // relabel it

pathlength[i] = pathlength[current] + adj[current][i];

}

}

}

} // end of Dijkstra function

/\* returns temporary vertex with minimum value of pathlength, returns NIL if no temporary vertex left or all temporary vertices left have pathlength infinity \*/

int min\_temp() {

int i;

int min = infinity;

int k = NIL;

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

if (status[i] == TEMP && pathlength[i] < min) {

min = pathlength[i];

k = i;

}

}

return k;

} // end of min\_temp

void findPath(int s, int v) {

int i, u;

int path[MAX]; // stores the shortest path

int shortDist = 0; // length of shortest path

int count = 0; // number of vertices in the shortest path

// store the full path in the array path

while (v != s) {

count++;

path[count] = v;

u = pred[v];

shortDist += adj[u][v];

v = u;

}

count++;

path[count] = s;

cout << "Shortest path is: ";

for (i = count; i >= 1; i--)

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

cout << "\nThe shortest distance is: " << shortDist << endl;

} // end of findPath

void create\_graph() {

int i, max\_edges, origin, destin, wt;

cout << "Enter the number of vertices: ";

cin >> n;

max\_edges = n \* (n - 1);

for (i = 1; i <= max\_edges; i++) {

cout << "Enter edge " << i << " (enter -1 -1 to finish): ";

cin >> origin >> destin;

if (origin == -1 && destin == -1)

break;

cout << "Enter weight of this edge: ";

cin >> wt;

if (origin > n || destin > n || origin < 0 || destin < 0) {

cout << "Invalid edge! Please enter again." << endl;

i--;

} else

adj[origin][destin] = wt; // assigning weight to the edge in the adjacency matrix

}

}

// OUTPUT

