**DAA PRACTICAL**

**Name:Parth Borse Roll No.: 07**

**DIJIKSTRAS ALGORITHM PROGRAM**

#include <iostream>

using namespace std;

#define MAX 100

#define TEMP 0

#define PERM 1

#define infinity 9999

#define NIL -1

// Function prototypes

void findPath(int s, int v);

void Dijkstra(int s);

int min\_temp();

void create\_graph();

// Global variables

int n; // number of vertices

int adj[MAX][MAX];

int pred[MAX];

int pathlength[MAX];

int status[MAX];

int main() {

int s, v;

create\_graph(); // Function call to create the graph

cout << "Enter the source vertex: ";

cin >> s;

Dijkstra(s); // Call Dijkstra's algorithm to find shortest paths from source vertex

// Loop to repeatedly ask for destination vertices until user enters -1

while (1) {

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

cin >> v;

if (v == -1)

break;

if (v < 0 || v >= n)

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

findPath(s, v); // Function call to find and display shortest path

}

return 0;

}

// Function to implement Dijkstra's algorithm to find shortest paths

void Dijkstra(int s) {

int i, current;

// Initialization: make all vertices temporary

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

pred[i] = NIL;

pathlength[i] = infinity;

status[i] = TEMP;

}

pathlength[s] = 0; // Set path length of source vertex to 0

// Main loop of Dijkstra's algorithm

while (1) {

// Search for temporary vertex with minimum path length and make it 'current' vertex

current = min\_temp();

if (current == NIL)

return;

status[current] = PERM;

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

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

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

pred[i] = current; // Set predecessor of vertex i to current vertex

pathlength[i] = pathlength[current] + adj[current][i]; // Update path length

}

}

}

}

}

// Function to find temporary vertex with minimum path length

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;

}

// Function to find and display shortest path from source vertex to destination vertex

void findPath(int s, int v) {

int i, u;

int path[MAX];

int shortDist = 0;

int count = 0;

// 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;

// Display the shortest path and its length

cout << "Shortest path is: ";

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

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

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

}

// Function to create the graph by taking user input for edges and their weights

void create\_graph() {

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

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

cin >> n;

max\_edges = n \* (n - 1); // Maximum number of edges in a directed graph

// Input edges and weights until user enters -1 -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;

// Validate input vertices

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

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

i--; // Decrement i to allow re-entry of invalid edge

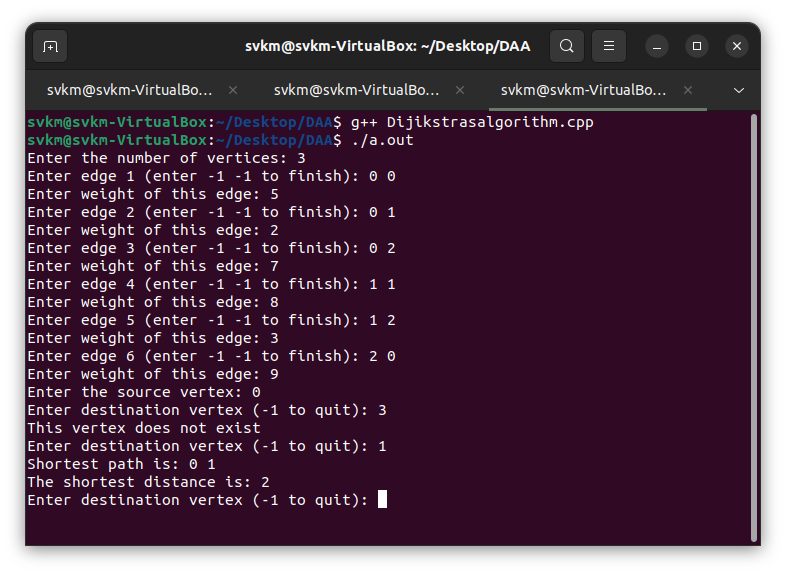
} else

adj[origin][destin] = wt;

}

}

// OUTPUT

****