Vishwakarma Institute of Technology, Pune

**Name:** Aditya Udhav Suryawanshi

**Div:** TY CS D **Roll No.:** 49 **Batch:** 2

**PRN No.:** 12111453 **Assignment No.** 4 **Subject:** CN LAB

**Problem Statement**: Write a program to find the shortest path using Dijkstra Equation for Link State Routing Protocol which is used by Open Shortest Path First Protocol (OSPF) in the Internet for the network flow provided by instructor.

**ShortestPath Code:**

#include <stdio.h>

#include <limits.h>

#define V 10 *// Maximum number of vertices (adjust as needed)*

*// Function to find the vertex with the minimum distance value*

int minDistance(int *dist*[], int *sptSet*[], int *n*) {

int min = INT\_MAX, min\_index;

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

if (!sptSet[v] && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

return min\_index;

}

*// Function to print the rough structure of the graph*

void printGraph(int graph[V][V], int *n*) {

printf("Rough structure of the graph:\n");

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

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

printf("%d ", graph[i][j]);

}

printf("\n");

}

}

*// Function to print the shortest path from source to destination*

void printPath(int *parent*[], int *j*) {

if (parent[j] == -1)

return;

printPath(parent, parent[j]);

printf(" -> %d", j);

}

*// Function to print the solution*

void printSolution(int *dist*[], int *parent*[], int *src*, int *dest*) {

printf("Shortest path from %d to %d: %d\n", src, dest, dist[dest]);

printf("Path: %d", src);

printPath(parent, dest);

printf("\n");

}

*// Function that implements Dijkstra's algorithm to find the shortest path*

void dijkstra(int graph[V][V], int *src*, int *dest*, int *n*) {

int dist[V]; *// The output array to store the shortest distance from src to i*

int parent[V]; *// Array to store the parent of each vertex in the shortest path tree*

int sptSet[V]; *// sptSet[i] will be true if vertex i is included in the shortest path tree or the shortest distance from src to i is finalized*

*// Initialize all distances as INFINITE and sptSet[] as false*

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

dist[i] = INT\_MAX;

sptSet[i] = 0;

parent[i] = -1;

}

*// Distance of source vertex from itself is always 0*

dist[src] = 0;

*// Find shortest path for all vertices*

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

int u = minDistance(dist, sptSet, n);

sptSet[u] = 1;

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

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

parent[v] = u;

}

}

}

*// Print the shortest path*

printSolution(dist, parent, src, dest);

}

int main() {

int graph[V][V] = {0};

int n;

printf("Enter the number of vertices (maximum %d): ", V);

scanf("%d", &n);

if (n < 1 || n > V) {

printf("Invalid number of vertices. Please enter a value between 1 and %d.\n", V);

return 1;

}

*// Input the graph edges and weights*

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

printf("Enter neighbors and weights for node %d (-1 to stop):\n", i);

int neighbor, weight;

while (1) {

scanf("%d", &neighbor);

if (neighbor == -1) {

break;

}

if (neighbor < 0 || neighbor >= n) {

printf("Invalid neighbor node. Enter a valid node between 0 and %d.\n", n - 1);

continue;

}

printf("Enter weight for edge to node %d: ", neighbor);

scanf("%d", &weight);

graph[i][neighbor] = weight;

graph[neighbor][i] = weight; *// Assuming an undirected graph*

}

}

int src, dest;

printf("Enter source node: ");

scanf("%d", &src);

printf("Enter destination node: ");

scanf("%d", &dest);

if (src < 0 || src >= n || dest < 0 || dest >= n) {

printf("Invalid source or destination node.\n");

return 1;

}

dijkstra(graph, src, dest, n);

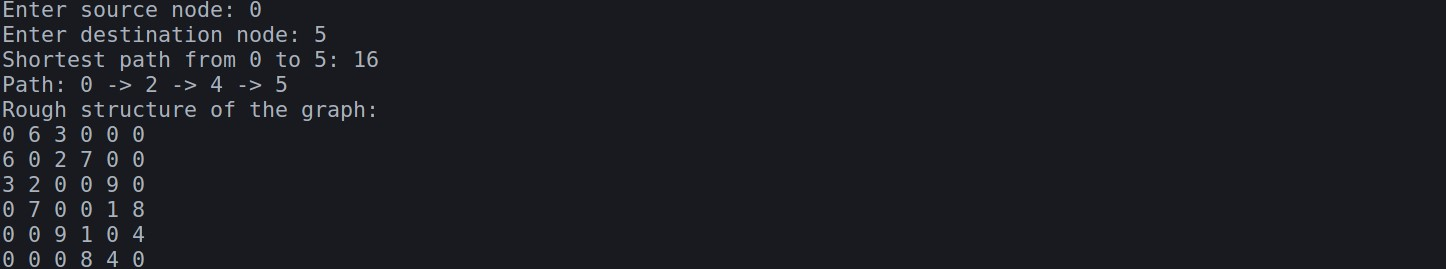
*// Print the rough structure of the graph*

printGraph(graph, n);

return 0;

}

**OUTPUT:**



**RoutingTable Code :**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

#define MAX\_NODES 100

*// Structure to represent a node in the network graph*

struct Node {

int id;

int cost[MAX\_NODES]; *// Array to store link costs to other nodes*

};

*// Function to find the node with the minimum distance*

int minDistance(int *dist*[], bool *sptSet*[], int *numNodes*) {

int min = INT\_MAX;

int minIndex;

int v;

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

if (!sptSet[v] && dist[v] < min) {

min = dist[v];

minIndex = v;

}

}

return minIndex;

}

*// Function to perform Dijkstra's algorithm*

void dijkstra(struct Node graph[MAX\_NODES], int *numNodes*, int *src*) {

int dist[MAX\_NODES];

bool sptSet[MAX\_NODES];

int i,count,v;

*// Initialize distance and sptSet*

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

dist[i] = INT\_MAX;

sptSet[i] = false;

}

*// Distance of source node from itself is always 0*

dist[src] = 0;

*// Find shortest path for all nodes*

for ( count = 0; count < numNodes - 1; count++) {

int u = minDistance(dist, sptSet, numNodes);

sptSet[u] = true;

*// Update dist[] value of adjacent nodes*

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

if (!sptSet[v] && graph[u].cost[v] && dist[u] != INT\_MAX

&& dist[u] + graph[u].cost[v] < dist[v]) {

dist[v] = dist[u] + graph[u].cost[v];

}

}

}

*// Print the shortest path distances*

printf("Shortest Path Distances from Node %d:\n", src);

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

printf("Node %d: %d\n", i, dist[i]);

}

}

int main() {

int numNodes,i,j;

printf("Enter the number of nodes: ");

scanf("%d", &numNodes);

struct Node graph[MAX\_NODES];

*// Input link costs between nodes*

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

graph[i].id = i;

printf("Enter link costs for Node %d to other nodes (0 for no connection):\n", i);

for ( j = 0; j < numNodes; j++) {

scanf("%d", &graph[i].cost[j]);

}

}

int sourceNode;

printf("Enter the source node: ");

scanf("%d", &sourceNode);

dijkstra(graph, numNodes, sourceNode);

return 0;

}

**OUTPUT :**

