**23. Implementation of Shortest Path Algorithms using Dijkstra’s Algorithm**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_NODES 100

#define INF INT\_MAX

// Function to find the vertex with minimum distance value

int minDistance(int dist[], int visited[], int n) {

int min = INF, min\_index;

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

if (visited[v] == 0 && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

return min\_index;

}

// 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 n, int src) {

printf("\nVertex\tDistance\tPath\n");

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

if (dist[i] == INF) {

printf("%d -> %d\tINF\t\tNo path\n", src, i);

} else {

printf("%d -> %d\t%d\t\t%d", src, i, dist[i], src);

printPath(parent, i);

printf("\n");

}

}

}

// Dijkstra's algorithm implementation

void dijkstra(int graph[MAX\_NODES][MAX\_NODES], int n, int src) {

int dist[MAX\_NODES]; // Shortest distance from src to i

int visited[MAX\_NODES]; // Visited vertices

int parent[MAX\_NODES]; // Parent array to store shortest path tree

// Initialize all distances as INFINITE and visited[] as false

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

dist[i] = INF;

visited[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++) {

// Pick the minimum distance vertex from unvisited vertices

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

// Mark the picked vertex as visited

visited[u] = 1;

// Update dist value of adjacent vertices of the picked vertex

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

// Update dist[v] only if:

// 1. It's not visited

// 2. There's an edge from u to v

// 3. Total weight of path from src to v through u is smaller than current dist[v]

if (!visited[v] && graph[u][v] && dist[u] != INF &&

dist[u] + graph[u][v] < dist[v]) {

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

parent[v] = u;

}

}

}

// Print the shortest distances and paths

printSolution(dist, parent, n, src);

}

// Function to display the graph matrix

void displayGraph(int graph[MAX\_NODES][MAX\_NODES], int n) {

printf("\nGraph Matrix (0 means no direct edge):\n");

printf(" ");

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

printf("%4d", i);

}

printf("\n");

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

printf("%4d:", i);

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

if (graph[i][j] == INF) {

printf(" -");

} else {

printf("%4d", graph[i][j]);

}

}

printf("\n");

}

}

int main() {

int n, edges, src;

int graph[MAX\_NODES][MAX\_NODES];

printf("=== Dijkstra's Shortest Path Algorithm ===\n");

// Initialize graph with no edges

for (int i = 0; i < MAX\_NODES; i++) {

for (int j = 0; j < MAX\_NODES; j++) {

graph[i][j] = 0;

}

}

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

scanf("%d", &n);

if (n <= 0 || n > MAX\_NODES) {

printf("Invalid number of nodes! (1-%d)\n", MAX\_NODES);

return 1;

}

printf("Enter number of edges: ");

scanf("%d", &edges);

printf("Enter edges (source destination weight):\n");

printf("Note: Enter weight 0 if no direct connection\n");

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

int u, v, w;

printf("Edge %d: ", i + 1);

scanf("%d %d %d", &u, &v, &w);

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

printf("Invalid node! Nodes must be between 0 and %d\n", n - 1);

i--;

continue;

}

if (w < 0) {

printf("Weight cannot be negative! Dijkstra's doesn't work with negative weights.\n");

i--;

continue;

}

graph[u][v] = w;

graph[v][u] = w; // For undirected graph

}

int choice;

do {

printf("\n=== Menu ===\n");

printf("1. Find shortest paths from a source node\n");

printf("2. Display graph matrix\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter source node (0 to %d): ", n - 1);

scanf("%d", &src);

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

printf("Invalid source node!\n");

} else {

dijkstra(graph, n, src);

}

break;

case 2:

displayGraph(graph, n);

break;

case 3:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (choice != 3);

return 0;

}