Roll NO:27

NAME : Mali Anjali Prakash

DIV: FYMCA-A

Assignment no -8

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_VERTICES 10

#define INF INT\_MAX

// Define a structure for an adjacency list node

typedef struct Node {

    int vertex;

    int weight;

    struct Node\* next;

} Node;

// Define a structure for the graph

typedef struct Graph {

    int numVertices;

    Node\* adjList[MAX\_VERTICES];

} Graph;

// Function to create a new graph

Graph\* createGraph(int vertices) {

    Graph\* graph = (Graph\*)malloc(sizeof(Graph));

    graph->numVertices = vertices;

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

        graph->adjList[i] = NULL;

    }

    return graph;

}

// Function to add an edge to the graph

void addEdge(Graph\* graph, int src, int dest, int weight) {

    // Add edge from src to dest

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    newNode->vertex = dest;

    newNode->weight = weight;

    newNode->next = graph->adjList[src];

    graph->adjList[src] = newNode;

    // Add edge from dest to src (since the graph is undirected)

    newNode = (Node\*)malloc(sizeof(Node));

    newNode->vertex = src;

    newNode->weight = weight;

    newNode->next = graph->adjList[dest];

    graph->adjList[dest] = newNode;

}

// Function to find the vertex with the minimum distance that is not yet visited

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

    int min = INF, minIndex;

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

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

            min = dist[v];

            minIndex = v;

        }

    }

    return minIndex;

}

// Dijkstra's algorithm to find the shortest path from the source vertex

void dijkstra(Graph\* graph, int startVertex, int dist[], int prev[]) {

    int visited[MAX\_VERTICES] = {0};

    for (int i = 0; i < graph->numVertices; i++) {

        dist[i] = INF;

        prev[i] = -1;

    }

    dist[startVertex] = 0;

    for (int count = 0; count < graph->numVertices - 1; count++) {

        int u = minDistance(dist, visited, graph->numVertices);

        visited[u] = 1;

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

        Node\* adjList = graph->adjList[u];

        while (adjList != NULL) {

            int v = adjList->vertex;

            if (!visited[v] && dist[u] != INF && dist[u] + adjList->weight < dist[v]) {

                dist[v] = dist[u] + adjList->weight;

                prev[v] = u;

            }

            adjList = adjList->next;

        }

    }

}

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

void printPath(int prev[], int j) {

    if (prev[j] == -1) {

        printf("%d ", j);

        return;

    }

    printPath(prev, prev[j]);

    printf("-> %d ", j);

}

// Function to print the shortest distance and path

void printSolution(int dist[], int prev[], int vertices, int startVertex, int endVertex) {

    printf("Shortest distance from %d to %d is: %d\n", startVertex, endVertex, dist[endVertex]);

    printf("Shortest path: ");

    printPath(prev, endVertex);

    printf("\n");

}

int main() {

    Graph\* graph = createGraph(6); // Let's assume 6 areas in the city

    // Add roads (edges) with distances

    addEdge(graph, 0, 1, 10); // Home -> Area 1

    addEdge(graph, 0, 2, 15); // Home -> Area 2

    addEdge(graph, 1, 2, 5);  // Area 1 -> Area 2

    addEdge(graph, 1, 3, 20); // Area 1 -> College

    addEdge(graph, 2, 3, 10); // Area 2 -> College

    addEdge(graph, 3, 4, 30); // College -> Area 4

    addEdge(graph, 4, 5, 5);  // Area 4 -> Area 5

    addEdge(graph, 3, 5, 10); // College -> Area 5

    int dist[MAX\_VERTICES];

    int prev[MAX\_VERTICES];

    // Perform Dijkstra's algorithm from Home (0) to College (3)

    dijkstra(graph, 0, dist, prev);

    // Print the shortest path and distance

    printSolution(dist, prev, graph->numVertices, 0, 3);

    return 0;

}

