Roll NO:27

NAME : Mali Anjali Prakash

DIV: FYMCA-A

Assignment no -7

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 10

// Define the structure for an adjacency list node

typedef struct Node {

    int vertex;

    struct Node\* next;

} Node;

// Define the 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 (undirected)

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

    // Add edge from src to dest

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

    newNode->vertex = dest;

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

    graph->adjList[src] = newNode;

    // Add edge from dest to src (since it's undirected)

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

    newNode->vertex = src;

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

    graph->adjList[dest] = newNode;

}

// Function to perform DFS

void DFS(Graph\* graph, int startVertex, int visited[]) {

    printf("Visiting vertex %d\n", startVertex);

    visited[startVertex] = 1;

    // Traverse the adjacency list of the vertex

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

    while (adjList != NULL) {

        int connectedVertex = adjList->vertex;

        if (!visited[connectedVertex]) {

            DFS(graph, connectedVertex, visited);

        }

        adjList = adjList->next;

    }

}

// Function to perform BFS

void BFS(Graph\* graph, int startVertex) {

    int visited[MAX\_VERTICES] = {0};

    int queue[MAX\_VERTICES];

    int front = 0, rear = 0;

    // Mark the start vertex as visited and enqueue it

    visited[startVertex] = 1;

    queue[rear++] = startVertex;

    while (front < rear) {

        int currentVertex = queue[front++];

        printf("Visiting vertex %d\n", currentVertex);

        // Traverse the adjacency list of the current vertex

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

        while (adjList != NULL) {

            int connectedVertex = adjList->vertex;

            if (!visited[connectedVertex]) {

                visited[connectedVertex] = 1;

                queue[rear++] = connectedVertex;

            }

            adjList = adjList->next;

        }

    }

}

// Function to print the adjacency list representation of the graph

void printGraph(Graph\* graph) {

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

        Node\* temp = graph->adjList[i];

        printf("\nVertex %d: ", i);

        while (temp) {

            printf("%d -> ", temp->vertex);

            temp = temp->next;

        }

        printf("NULL");

    }

    printf("\n");

}

int main() {

    // Create a graph (example city map with 6 locations)

    Graph\* graph = createGraph(6);

    // Add edges (undirected roads between locations)

    addEdge(graph, 0, 1); // Road between location 0 and 1

    addEdge(graph, 0, 2); // Road between location 0 and 2

    addEdge(graph, 1, 3); // Road between location 1 and 3

    addEdge(graph, 1, 4); // Road between location 1 and 4

    addEdge(graph, 2, 4); // Road between location 2 and 4

    addEdge(graph, 3, 5); // Road between location 3 and 5

    addEdge(graph, 4, 5); // Road between location 4 and 5

    // Print the graph

    printf("City Map (Graph Representation):\n");

    printGraph(graph);

    // Perform DFS from vertex 0

    printf("\nDepth First Search (DFS) starting from vertex 0:\n");

    int visited[MAX\_VERTICES] = {0};

    DFS(graph, 0, visited);

    // Perform BFS from vertex 0

    printf("\nBreadth First Search (BFS) starting from vertex 0:\n");

    BFS(graph, 0);

    return 0;

}

