Practical-10

Implement graph data structures using adjacency matrix and

adjacency list representation. Perform the graph traversal such as breadth-first-search (BFS) and depth-first-search (DFS).

//graph

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Define the maximum number of vertices in the graph

#define MAX\_VERTICES 10

// Adjacency Matrix Representation of Graph

typedef struct GraphMatrix {

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

int numVertices;

} GraphMatrix;

// Adjacency List Representation of Graph

typedef struct GraphList {

int numVertices;

struct Node\* adjLists[MAX\_VERTICES];

} GraphList;

// Node structure for adjacency list

struct Node {

int vertex;

struct Node\* next;

};

// Function to initialize an adjacency matrix graph

void initGraphMatrix(GraphMatrix\* g, int vertices) {

g->numVertices = vertices;

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

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

g->adjMatrix[i][j] = 0; // No edges initially

}

}

}

// Function to add an edge to the adjacency matrix

void addEdgeMatrix(GraphMatrix\* g, int src, int dest) {

g->adjMatrix[src][dest] = 1;

g->adjMatrix[dest][src] = 1; // For undirected graph

}

// Function to print the adjacency matrix

void printGraphMatrix(GraphMatrix\* g) {

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

for (int j = 0; j < g->numVertices; j++) {

printf("%d ", g->adjMatrix[i][j]);

}

printf("\n");

}

}

// Function to initialize an adjacency list graph

void initGraphList(GraphList\* g, int vertices) {

g->numVertices = vertices;

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

g->adjLists[i] = NULL; // No edges initially

}

}

// Function to create a new node for adjacency list

struct Node\* createNode(int v) {

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Function to add an edge to the adjacency list

void addEdgeList(GraphList\* g, int src, int dest) {

struct Node\* newNode = createNode(dest);

newNode->next = g->adjLists[src];

g->adjLists[src] = newNode;

// For undirected graph, add an edge from dest to src as well

newNode = createNode(src);

newNode->next = g->adjLists[dest];

g->adjLists[dest] = newNode;

}

// Function to print the adjacency list

void printGraphList(GraphList\* g) {

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

struct Node\* temp = g->adjLists[i];

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

while (temp) {

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

temp = temp->next;

}

printf("NULL\n");

}

}

// Breadth-First Search (BFS) using adjacency matrix

void BFSMatrix(GraphMatrix\* g, int startVertex) {

bool visited[MAX\_VERTICES] = {false};

int queue[MAX\_VERTICES];

int front = -1, rear = -1;

visited[startVertex] = true;

queue[++rear] = startVertex;

printf("BFS (Matrix): ");

while (front != rear) {

int current = queue[++front];

printf("%d ", current);

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

if (g->adjMatrix[current][i] == 1 && !visited[i]) {

queue[++rear] = i;

visited[i] = true;

}

}

}

printf("\n");

}

// Depth-First Search (DFS) using adjacency matrix

void DFSMatrix(GraphMatrix\* g, int vertex, bool visited[]) {

visited[vertex] = true;

printf("%d ", vertex);

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

if (g->adjMatrix[vertex][i] == 1 && !visited[i]) {

DFSMatrix(g, i, visited);

}

}

}

void DFSMatrixWrapper(GraphMatrix\* g, int startVertex) {

bool visited[MAX\_VERTICES] = {false};

printf("DFS (Matrix): ");

DFSMatrix(g, startVertex, visited);

printf("\n");

}

// Breadth-First Search (BFS) using adjacency list

void BFSList(GraphList\* g, int startVertex) {

bool visited[MAX\_VERTICES] = {false};

int queue[MAX\_VERTICES];

int front = -1, rear = -1;

visited[startVertex] = true;

queue[++rear] = startVertex;

printf("BFS (List): ");

while (front != rear) {

int current = queue[++front];

printf("%d ", current);

struct Node\* temp = g->adjLists[current];

while (temp) {

if (!visited[temp->vertex]) {

queue[++rear] = temp->vertex;

visited[temp->vertex] = true;

}

temp = temp->next;

}

}

printf("\n");

}

// Depth-First Search (DFS) using adjacency list

void DFSList(GraphList\* g, int vertex, bool visited[]) {

visited[vertex] = true;

printf("%d ", vertex);

struct Node\* temp = g->adjLists[vertex];

while (temp) {

if (!visited[temp->vertex]) {

DFSList(g, temp->vertex, visited);

}

temp = temp->next;

}

}

void DFSListWrapper(GraphList\* g, int startVertex) {

bool visited[MAX\_VERTICES] = {false};

printf("DFS (List): ");

DFSList(g, startVertex, visited);

printf("\n");

}

int main() {

// Graph using Adjacency Matrix

GraphMatrix gMatrix;

initGraphMatrix(&gMatrix, 5);

addEdgeMatrix(&gMatrix, 0, 1);

addEdgeMatrix(&gMatrix, 0, 4);

addEdgeMatrix(&gMatrix, 1, 2);

addEdgeMatrix(&gMatrix, 1, 3);

addEdgeMatrix(&gMatrix, 1, 4);

addEdgeMatrix(&gMatrix, 3, 4);

printf("Adjacency Matrix Representation:\n");

printGraphMatrix(&gMatrix);

// BFS and DFS using Adjacency Matrix

BFSMatrix(&gMatrix, 0);

DFSMatrixWrapper(&gMatrix, 0);

// Graph using Adjacency List

GraphList gList;

initGraphList(&gList, 5);

addEdgeList(&gList, 0, 1);

addEdgeList(&gList, 0, 4);

addEdgeList(&gList, 1, 2);

addEdgeList(&gList, 1, 3);

addEdgeList(&gList, 1, 4);

addEdgeList(&gList, 3, 4);

printf("\nAdjacency List Representation:\n");

printGraphList(&gList);

// BFS and DFS using Adjacency List

BFSList(&gList, 0);

DFSListWrapper(&gList, 0);

return 0;

}

Output: