**22. Write a C program to Graph traversal using Depth First Search**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODES 100

// Stack implementation for DFS (iterative version)

typedef struct Stack {

int items[MAX\_NODES];

int top;

} Stack;

// Graph representation using adjacency matrix

typedef struct Graph {

int numNodes;

int adjMatrix[MAX\_NODES][MAX\_NODES];

} Graph;

// Stack operations

Stack\* createStack() {

Stack\* s = (Stack\*)malloc(sizeof(Stack));

s->top = -1;

return s;

}

int isStackEmpty(Stack\* s) {

return s->top == -1;

}

void push(Stack\* s, int value) {

if (s->top == MAX\_NODES - 1) {

printf("Stack is full!\n");

} else {

s->top++;

s->items[s->top] = value;

}

}

int pop(Stack\* s) {

if (isStackEmpty(s)) {

printf("Stack is empty!\n");

return -1;

} else {

return s->items[s->top--];

}

}

int peek(Stack\* s) {

if (isStackEmpty(s)) {

return -1;

}

return s->items[s->top];

}

// Graph operations

Graph\* createGraph(int nodes) {

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

graph->numNodes = nodes;

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

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

graph->adjMatrix[i][j] = 0;

}

}

return graph;

}

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

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

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

}

// DFS traversal using recursion

void DFSRecursive(Graph\* graph, int node, int visited[]) {

visited[node] = 1;

printf("%d ", node);

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

if (graph->adjMatrix[node][i] == 1 && !visited[i]) {

DFSRecursive(graph, i, visited);

}

}

}

// DFS traversal using iterative approach (stack)

void DFSIterative(Graph\* graph, int startNode) {

int visited[MAX\_NODES] = {0};

Stack\* s = createStack();

push(s, startNode);

visited[startNode] = 1;

printf("DFS Traversal (Iterative): ");

while (!isStackEmpty(s)) {

int currentNode = pop(s);

printf("%d ", currentNode);

// Push all unvisited adjacent nodes in reverse order

// to maintain the same traversal order as recursive

for (int i = graph->numNodes - 1; i >= 0; i--) {

if (graph->adjMatrix[currentNode][i] == 1 && !visited[i]) {

visited[i] = 1;

push(s, i);

}

}

}

printf("\n");

free(s);

}

// Wrapper function for recursive DFS

void DFSRecursiveWrapper(Graph\* graph, int startNode) {

int visited[MAX\_NODES] = {0};

printf("DFS Traversal (Recursive): ");

DFSRecursive(graph, startNode, visited);

printf("\n");

}

// Display adjacency matrix

void displayGraph(Graph\* graph) {

printf("\nAdjacency Matrix:\n");

printf(" ");

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

printf("%2d ", i);

}

printf("\n");

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

printf("%2d: ", i);

for (int j = 0; j < graph->numNodes; j++) {

printf("%2d ", graph->adjMatrix[i][j]);

}

printf("\n");

}

}

int main() {

int nodes, edges, choice, startNode;

int src, dest;

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

scanf("%d", &nodes);

Graph\* graph = createGraph(nodes);

printf("Enter number of edges: ");

scanf("%d", &edges);

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

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

scanf("%d %d", &src, &dest);

if (src >= nodes || dest >= nodes) {

printf("Invalid edge! Node numbers should be between 0 and %d\n", nodes-1);

i--;

continue;

}

addEdge(graph, src, dest);

}

while (1) {

printf("\nGraph Operations Menu:\n");

printf("1. Perform DFS Traversal (Recursive)\n");

printf("2. Perform DFS Traversal (Iterative)\n");

printf("3. Display Adjacency Matrix\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter starting node for DFS (0 to %d): ", nodes-1);

scanf("%d", &startNode);

if (startNode < 0 || startNode >= nodes) {

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

} else {

DFSRecursiveWrapper(graph, startNode);

}

break;

case 2:

printf("Enter starting node for DFS (0 to %d): ", nodes-1);

scanf("%d", &startNode);

if (startNode < 0 || startNode >= nodes) {

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

} else {

DFSIterative(graph, startNode);

}

break;

case 3:

displayGraph(graph);

break;

case 4:

free(graph);

printf("Exiting program.\n");

exit(0);

default:

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

}

}

return 0;

}