**AI LAB ASSIGNMENT 2**

**Name: Atharva Salitri PRN: 12310120**

**Div : CSAI B Batch: 2 Roll No: 37**

**Implement Uninformed Search**

**1)BREADTH FIRST SEARCH (BFS)**

**CODE:**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

int queue[MAX\_VERTICES];

int front = -1, rear = -1;

void enqueue(int vertex) {

if (rear == MAX\_VERTICES - 1) {

printf("Queue is full.\n");

} else {

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = vertex;

}

}

int dequeue() {

int item;

if (front == -1 || front > rear) {

printf("Queue is empty.\n");

item = -1;

} else {

item = queue[front];

front++;

}

return item;

}

bool isEmpty() {

return front == -1 || front > rear;

}

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

Node\* graph[MAX\_VERTICES];

void addEdge(int from, int to) {

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

newNode->vertex = to;

newNode->next = graph[from];

graph[from] = newNode;

}

void BFS(int startVertex, int vertices) {

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

printf("BFS traversal from right: \n");

visited[startVertex] = true;

enqueue(startVertex);

while (!isEmpty()) {

int currentVertex = dequeue();

printf("%d ", currentVertex);

Node\* temp = graph[currentVertex];

while (temp != NULL) {

int adjacentVertex = temp->vertex;

if (!visited[adjacentVertex]) {

visited[adjacentVertex] = true;

enqueue(adjacentVertex);

}

temp = temp->next;

}

}

}

int main() {

int vertices, edges;

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

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

graph[i] = NULL;

}

printf("Enter the number of edges: ");

scanf("%d", &edges);

printf("Enter the edges (from to): \n");

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

int from, to;

scanf("%d %d", &from, &to);

addEdge(from, to);

//addEdge(to, from);

}

int startVertex;

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

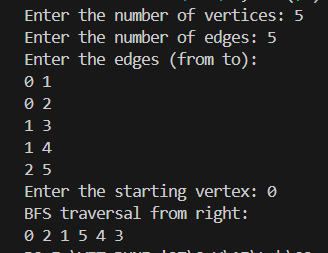
BFS(startVertex, vertices);

printf("\n");

return 0;

}

**OUTPUT:**

****

**2)DEPTH FIRST SEARCH (DFS)**

**CODE:**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

Node\* graph[MAX\_VERTICES];

void addEdge(int from, int to) {

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

newNode->vertex = to;

newNode->next = graph[from];

graph[from] = newNode;

}

void DFSUtil(int vertex, bool visited[]) {

// Mark the current node as visited and print it

visited[vertex] = true;

printf("%d ", vertex);

// Recur for all the vertices adjacent to this vertex

Node\* temp = graph[vertex];

while (temp != NULL) {

int adjacentVertex = temp->vertex;

if (!visited[adjacentVertex]) {

DFSUtil(adjacentVertex, visited);

}

temp = temp->next;

}

}

void DFS(int startVertex, int vertices) {

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

printf("DFS traversal from vertex %d: \n", startVertex);

DFSUtil(startVertex, visited);

}

int main() {

int vertices, edges;

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

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

graph[i] = NULL;

}

printf("Enter the number of edges: ");

scanf("%d", &edges);

printf("Enter the edges (from to): \n");

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

int from, to;

scanf("%d %d", &from, &to);

addEdge(from, to);

// Uncomment the next line for an undirected graph

// addEdge(to, from);

}

int startVertex;

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

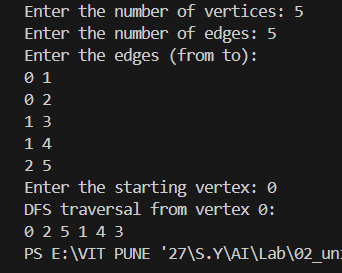
DFS(startVertex, vertices);

printf("\n");

return 0;

}

**OUTPUT:**



**3) DEPTH LIMITED SEARCH (DLFS)**

**CODE:**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

Node\* graph[MAX\_VERTICES];

void addEdge(int from, int to) {

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

newNode->vertex = to;

newNode->next = graph[from];

graph[from] = newNode;

}

void DLSUtil(int vertex, bool visited[], int depth, int limit) {

// If the depth limit is reached, return

if (depth > limit) {

return;

}

// Mark the current node as visited and print it

visited[vertex] = true;

printf("%d ", vertex);

// Recur for all the vertices adjacent to this vertex

Node\* temp = graph[vertex];

while (temp != NULL) {

int adjacentVertex = temp->vertex;

if (!visited[adjacentVertex]) {

DLSUtil(adjacentVertex, visited, depth + 1, limit);

}

temp = temp->next;

}

}

void DLS(int startVertex, int vertices, int limit) {

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

printf("Depth-Limited Search traversal from vertex %d with limit %d: \n", startVertex, limit);

DLSUtil(startVertex, visited, 0, limit);

}

int main() {

int vertices, edges;

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

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

graph[i] = NULL;

}

printf("Enter the number of edges: ");

scanf("%d", &edges);

printf("Enter the edges (from to): \n");

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

int from, to;

scanf("%d %d", &from, &to);

addEdge(from, to);

// Uncomment the next line for an undirected graph

// addEdge(to, from);

}

int startVertex, limit;

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

printf("Enter the depth limit: ");

scanf("%d", &limit);

DLS(startVertex, vertices, limit);

printf("\n");

return 0;

}

**OUTPUT:**

