**Assignment 3**

**Consider the scenario of a maze. The maze is represented as a grid of cells, where each cell can be either open or blocked. Each cell in the maze represents a vertex. The goal is to find a path from the starting point to the goal within a given maze using DFS and BFS.**

**CODE:**

#include <iostream>

using namespace std;

struct Node

{

int vertex;

Node\* next;

};

struct List

{

Node\* head;

};

struct StackNode

{

int data;

StackNode\* next;

};

class Stack

{

private:

StackNode\* top;

public:

Stack()

{

top = NULL;

}

void push(int val)

{

StackNode\* newNode = new StackNode;

newNode->data = val;

newNode->next = top;

top = newNode;

}

int pop()

{

if (top == NULL) {

cout << "Stack Underflow\n";

return -1;

}

int val=top->data;

StackNode\* temp=top;

top=top->next;

delete temp;

return val;

}

bool isEmpty()

{

return top == NULL;

}

};

struct QueueNode

{

int data;

QueueNode\* next;

};

class Queue

{

private:

QueueNode\* front;

QueueNode\* rear;

public:

Queue()

{

front=rear=NULL;

}

void enqueue(int val)

{

QueueNode\* newNode = new QueueNode;

newNode->data = val;

newNode->next = NULL;

if (rear == NULL)

{

front = rear = newNode;

return;

}

rear->next = newNode;

rear = newNode;

}

int dequeue()

{

if (front == NULL)

{

cout << "Queue Underflow\n";

return -1;

}

int val = front->data;

QueueNode\* temp = front;

front = front->next;

if (front == NULL)

{

rear = NULL;

}

delete temp;

return val;

}

bool isEmpty()

{

return front == NULL;

}

};

class Graph

{

private:

int vertices;

List\* adjacencyList;

int\* visited;

public:

Graph(int numVertices)

{

vertices=numVertices;

adjacencyList=new List[vertices];

visited = new int[vertices];

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

{

adjacencyList[i].head = NULL;

visited[i] = 0;

}

}

void addEdge(int src, int dest)

{

Node\* newNode = new Node;

newNode->vertex = dest;

newNode->next = adjacencyList[src].head;

adjacencyList[src].head = newNode;

newNode = new Node;

newNode->vertex = src;

newNode->next = adjacencyList[dest].head;

adjacencyList[dest].head = newNode;

}

void bfs(int start)

{

Queue Q;

int visited[vertices] = {0};

Q.enqueue(start);

visited[start]=1;

while (!Q.isEmpty())

{

int current = Q.dequeue();

cout << current << " ";

Node\* temp = adjacencyList[current].head;

while (temp != NULL)

{

if (visited[temp->vertex] == 0)

{

Q.enqueue(temp->vertex);

visited[temp->vertex] = 1;

}

temp = temp->next;

}

}

cout << endl;

}

void dfs(int start)

{

Stack S;

int visited[vertices]={0};

S.push(start);

while (!S.isEmpty())

{

int v = S.pop();

if (visited[v]==0)

{

cout << v << " ";

visited[v] = 1;

Node\* temp = adjacencyList[v].head;

while (temp != NULL)

{

if (visited[temp->vertex] == 0)

{

S.push(temp->vertex);

}

temp = temp->next;

}

}

}

cout << endl;

}

};

int main()

{

int numVertices, numEdges;

cout<<"Enter number of vertices: ";

cin>>numVertices;

Graph g(numVertices);

cout <<"Enter number of edges: ";

cin>>numEdges;

cout << "Enter edges (source destination):\n";

for (int i = 0; i < numEdges; i++)

{

int src, dest;

cin>>src>>dest;

g.addEdge(src, dest);

}

int startNode;

cout<<"Enter starting node for BFS: ";

cin >> startNode;

cout<<"BFS Traversal: ";

g.bfs(startNode);

cout<<"Enter starting node for DFS: ";

cin>>startNode;

cout<<"DFS Traversal: ";

g.dfs(startNode);

cout<<"\nDone\n";

return 0;

}