**DAA PRACTICAL**

**Name:Parth Borse Roll No.: 07**

**Breadth First search(BFS)**

#include <iostream>

#include <cstdlib>

using namespace std;

#define MAX 100

#define infinity 999

#define NIL -1

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

int Distance[MAX];

int pred[MAX];

void create\_graph();

void BF\_traversal();

void BFS(int v);

int queue[MAX];

int front = -1, rear = -1;

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main() {

int u, v, i, count, path[MAX];

create\_graph();

BF\_traversal();

while (true) {

cout << "Enter destination vertex (-1 to quit): ";

cin >> v;

if (v == -1)

break;

if (v < 0 || v >= n) {

cout << "Destination vertex does not exist\n";

continue;

}

count = 0;

if (Distance[v] == infinity) {

cout << "No path from start vertex to destination vertex\n";

continue;

} else

cout << "Shortest distance is " << Distance[v] << endl;

// Store the full path in the array path

while (v != NIL) {

count++;

path[count] = v;

u = pred[v];

v = u;

}

cout << "Shortest path is: ";

for (i = count; i >= 1; i--) {

cout << path[i] << " ";

}

cout << endl;

}

return 0;

} //end of main

void create\_graph() {

int i, max\_edges, origin, destin;

cout << "Enter the number of vertices: ";

cin >> n;

max\_edges = n \* (n - 1);

for (i = 1; i <= max\_edges; i++) {

cout << "Enter edge " << i << " (enter -1 -1 to finish): ";

cin >> origin >> destin;

if (origin == -1 && destin == -1)

break;

if (origin >= n || destin >= n || origin < 0 || destin < 0) {

cout << "Invalid edge! Please enter again." << endl;

i--;

} else {

adj[origin][destin] = 1; // Marking the existence of edge from origin to destin

}

}

}

void BF\_traversal() {

int v;

for (v = 0; v < n; v++) {

state[v] = initial;

pred[v] = NIL;

Distance[v] = infinity;

}

cout << "Enter the starting vertex for BFS: ";

cin >> v;

BFS(v);

cout << endl;

}

void BFS(int v) {

int i;

insert\_queue(v);

state[v] = waiting;

Distance[v] = 0;

pred[v] = NIL;

while (!isEmpty\_queue()) {

v = delete\_queue();

state[v] = visited;

for (i = 0; i < n; i++) {

//check for adjacent unvisited vertices

if (adj[v][i] != 0 && state[i] == initial) {

insert\_queue(i);

state[i] = waiting;

pred[i] = v;

Distance[i] = Distance[v] + 1;

}

}

}

}

void insert\_queue(int vertex) {

if (rear == MAX - 1)

cout << "Queue Overflow\n";

else {

if (front == -1)

front = 0;

rear = rear + 1;

queue[rear] = vertex;

}

}

int delete\_queue() {

int delete\_item;

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

cout << "Queue Underflow\n";

exit(1);

}

delete\_item = queue[front];

front = front + 1;

return delete\_item;

}

int isEmpty\_queue() {

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

return 1;

else

return 0;

}

**//Output**

