Practical 12

Aim:- To Implement BFS and DFS Graph Traversal techniques.

Theory:-

Breadth First Search Graph Traversal

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node

that meets a set of criteria. It starts at the root of the graph and visits all nodes at the

current depth level before moving on to the nodes at the next depth level.

How does BFS work?

Starting from the root, all the nodes at a particular level are visited first and then the nodes

of the next level are traversed till all the nodes are visited.

To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed

into the queue and the nodes of the current level are marked visited and popped from the

queue.

Depth First Search Graph Traversal

Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The

only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited

twice). To avoid processing a node more than once, use a boolean visited array. A graph can

have more than one DFS traversal.

How does DFS work?

Depth-first search is an algorithm for traversing or searching tree or graph data structures.

The algorithm starts at the root node (selecting some arbitrary node as the root node in the

case of a graph) and explores as far as possible along each branch before backtracking.

1. BFS Program

Code:-

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 50

// This struct represents a directed graph using

// adjacency list representation

typedef struct Graph\_t {

// No. of vertices

int V;

bool adj[MAX\_VERTICES][MAX\_VERTICES];

} Graph;

// Constructor

Graph\* Graph\_create(int V)

{

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

g->V = V;

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

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

g->adj[i][j] = false;

}

}

return g;

}

// Destructor

void Graph\_destroy(Graph\* g) { free(g); }

// Function to add an edge to graph

void Graph\_addEdge(Graph\* g, int v, int w)

{

// Add w to v’s list.

g->adj[v][w] = true;

}

// Prints BFS traversal from a given source s

void Graph\_BFS(Graph\* g, int s)

{

// Mark all the vertices as not visited

bool visited[MAX\_VERTICES];

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

visited[i] = false;

}

// Create a queue for BFS

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue[rear++] = s;

while (front != rear) {

// Dequeue a vertex from queue and print it

s = queue[front++];

printf("%d ", s);

// Get all adjacent vertices of the dequeued

// vertex s.

// If an adjacent has not been visited,

// then mark it visited and enqueue it

for (int adjacent = 0; adjacent < g->V;

adjacent++) {

if (g->adj[s][adjacent] && !visited[adjacent]) {

visited[adjacent] = true;

queue[rear++] = adjacent;

}

}

}

}

// Driver code

int main()

{

// Create a graph

Graph\* g = Graph\_create(4);

Graph\_addEdge(g, 0, 1);

Graph\_addEdge(g, 0, 2);

Graph\_addEdge(g, 1, 2);

Graph\_addEdge(g, 2, 0);

Graph\_addEdge(g, 2, 3);

Graph\_addEdge(g, 3, 3);

printf("Following is Breadth First Traversal "

"(starting from vertex 2) \n");

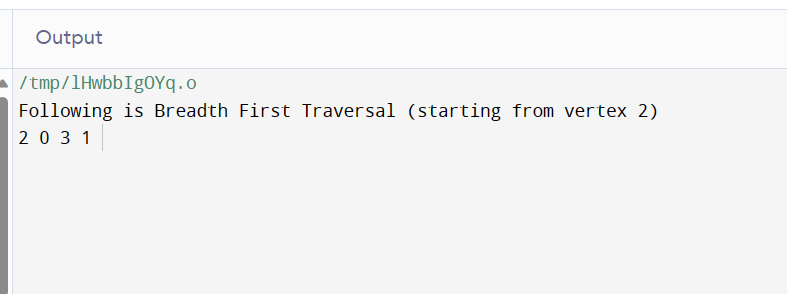
Graph\_BFS(g, 2);

Graph\_destroy(g);

return 0;

}

Output:-



1. DFS Program

Code:-

// C++ program to print DFS traversal from

// a given vertex in a given graph

#include <bits/stdc++.h>

using namespace std;

// Graph class represents a directed graph

// using adjacency list representation

class Graph {

public:

map<int, bool> visited;

map<int, list<int> > adj;

// Function to add an edge to graph

void addEdge(int v, int w);

// DFS traversal of the vertices

// reachable from v

void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

// Add w to v’s list.

adj[v].push\_back(w);

}

void Graph::DFS(int v)

{

// Mark the current node as visited and

// print it

visited[v] = true;

cout << v << " ";

// Recur for all the vertices adjacent

// to this vertex

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFS(\*i);

}

// Driver code

int main()

{

// Create a graph given in the above diagram

Graph g;

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

cout << "Following is Depth First Traversal"

" (starting from vertex 2) \n";

// Function call

g.DFS(2);

return 0;

}

Output:-

