**Experiment 3.1**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 08/11/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

1. **Aim/Overview of the practical:**

Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as (i) to find the topological sort of a directed acyclic graph, OR (ii) to find a path from source to goal in a maze.

1. **Task to be done/which logistics used:**

In this problem we will implement DFS and its application using C++.

1. **Algorithm/Flowchart:**

For DFS of an undirected graph-:

* + Create a recursive function that takes the index of the node and a visited array.
  + Mark the current node as visited and print the node.
  + Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.
  + Run a loop from 0 to the number of vertices and check if the node is unvisited in the previous DFS, then call the recursive function with the current node.

For topological sorting-:

* + Create a stack to store the nodes.
  + Initialize visited array of size N to keep the record of visited nodes.
  + Run a loop from 0 till N.
  + If the node is not marked True in visited array.
  + Call the recursive function for topological sort and perform the following steps.
  + Mark the current node as True in the visited array.
  + Run a loop on all the nodes which has a directed edge to the current node.
  + If the node is not marked True in the visited array.
  + Recursively call the topological sort function on the node.
  + Push the current node in the stack.
  + Print all the elements in the stack.

For maze problem-:

* + Start from the given source cell in the matrix and explore all four possible paths.
  + Check if the destination is reached or not.
  + Explore all the paths and backtrack if destination is not reached.
  + And also keep track of visited cells using an array.

1. **Steps for experiment/practical/Code:**

DFS of an undirected graph:

#include <bits/stdc++.h>

using namespace std;

class Graph

{

void DFSUtil(int v);

public:

map < int, bool > visited;

map < int, list < int >> adj;

void addEdge(int v, int w);

void DFS();

};

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

{

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

}

void Graph::DFSUtil(int v)

{

visited[v] = true;

cout << v << " ";

list < int > ::iterator i;

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

if (!visited[ \* i])

DFSUtil( \* i);

}

void Graph::DFS()

{

for (auto i: adj)

if (visited[i.first] == false)

DFSUtil(i.first);

}

int main()

{

cout << "Sahul Kumar Parida" << endl;

cout << "20BCS4919" << endl;

Graph g;

g.addEdge(0, 2);

g.addEdge(0, 8);

g.addEdge(1, 1);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(9, 7);

cout << "Depth First Traversal of an Undirected Graph\n";

g.DFS();

return 0;

}

Topological sorting:

#include <bits/stdc++.h>

using namespace std;

class Graph

{

int V;

list < int > \* adj;

void topologicalSortUtil(int v, bool visited[], stack < int > & Stack);

public:

Graph(int V);

void addEdge(int v, int w);

void topologicalSort();

};

Graph::Graph(int V)

{

this -> V = V;

adj = new list < int > [V];

}

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

{

// Add w to v’s list.

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

}

void Graph::topologicalSortUtil(int v, bool visited[], stack < int > & Stack)

{

visited[v] = true;

list < int > ::iterator i;

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

if (!visited[ \* i])

topologicalSortUtil( \* i, visited, Stack);

Stack.push(v);

}

void Graph::topologicalSort()

{

stack < int > Stack;

bool \* visited = new bool[V];

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

visited[i] = false;

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

if (visited[i] == false)

topologicalSortUtil(i, visited, Stack);

while (Stack.empty() == false) {

cout << Stack.top() << " ";

Stack.pop();

}

}

int main()

{

cout << "Sahul Kumar Parida" << endl;

cout << "20BCS4919" << endl;

Graph g(6);

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

g.addEdge(3, 1);

cout << "Following is a Topological Sort of the given "

"graph \n";

g.topologicalSort();

return 0;

}

Maze problem:

#include <climits>

#include <cstring>

#include <iostream>

#include <vector>

using namespace std;

bool isSafe(vector<vector<int>> &mat, vector<vector<bool>> &visited, int x,

int y)

{

return (x >= 0 && x < mat.size() && y >= 0 && y < mat[0].size()) &&

mat[x][y] == 1 && !visited[x][y];

}

void findShortestPath(vector<vector<int>> &mat, vector<vector<bool>> &visited,

int i, int j, int x, int y, int &min\_dist, int dist)

{

if (i == x && j == y)

{

min\_dist = min(dist, min\_dist);

return;

}

visited[i][j] = true;

if (isSafe(mat, visited, i + 1, j)) {

findShortestPath(mat, visited, i + 1, j, x, y, min\_dist, dist + 1);

}

if (isSafe(mat, visited, i, j + 1)) {

findShortestPath(mat, visited, i, j + 1, x, y, min\_dist, dist + 1);

}

if (isSafe(mat, visited, i - 1, j)) {

findShortestPath(mat, visited, i - 1, j, x, y, min\_dist, dist + 1);

}

// go to the left cell

if (isSafe(mat, visited, i, j - 1)) {

findShortestPath(mat, visited, i, j - 1, x, y, min\_dist, dist + 1);

}

visited[i][j] = false;

}

int findShortestPathLength(vector<vector<int>> &mat, pair<int, int> &src,

pair<int, int> &dest)

{

if (mat.size() == 0 || mat[src.first][src.second] == 0 ||

mat[dest.first][dest.second] == 0)

return -1;

int row = mat.size();

int col = mat[0].size();

vector<vector<bool>> visited;

visited.resize(row, vector<bool>(col));

int dist = INT\_MAX;

findShortestPath(mat, visited, src.first, src.second, dest.first, dest.second,

dist, 0);

if (dist != INT\_MAX)

return dist;

return -1;

}

int main()

{

cout << "Sahul Kumar Parida" << endl;

cout << "20BCS4919" << endl;

vector<vector<int>> mat = {

{1, 0, 1, 1, 1, 1, 0, 1, 1, 1}, {1, 0, 1, 0, 1, 1, 1, 0, 1, 1},

{1, 1, 1, 0, 1, 1, 0, 1, 0, 1}, {0, 0, 0, 0, 1, 0, 0, 0, 0, 1},

{1, 1, 1, 0, 1, 1, 1, 0, 1, 0}, {1, 0, 1, 1, 1, 1, 0, 1, 0, 0},

{1, 0, 0, 0, 0, 0, 0, 0, 0, 1}, {1, 0, 1, 1, 1, 1, 0, 1, 1, 1},

{1, 1, 0, 0, 0, 0, 1, 0, 0, 1}};

pair<int, int> src = make\_pair(0, 0);

pair<int, int> dest = make\_pair(3, 4);

int dist = findShortestPathLength(mat, src, dest);

if (dist != -1)

cout << "Shortest Path is " << dist;

else

cout << "Shortest Path doesn't exist";

return 0;

}

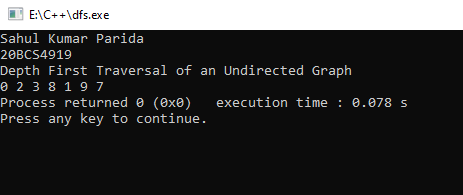
1. **Observations/Discussions/ Complexity Analysis:**

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

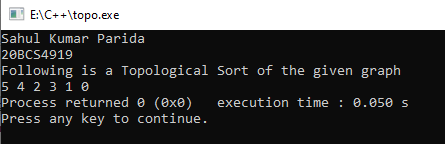
The space complexity of the algorithm is O(V).

The time complexity of topological sort is O(V+E), where V = Vertices, E = Edges.

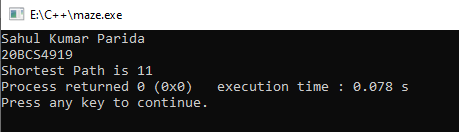
1. **Output:**

DFS of an undirected graph:

Topological sorting:



Maze problem:



**Learning outcomes (What I have learnt):**

1. DFS (Depth First Search).

2. Topological sorting.

3. Applications of DFS.