**Experiment 3.1**

**Aim:** *Develop a program and analyze complexity 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.*

**Objectives:** *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.*

**Input/Apparatus Used:** *VS CODE*

# Procedure/Algorithm:

# *Create a class or data structure to represent a graph.*

# *Initialize the graph with the number of vertices (V) and an adjacency list to represent the edges.*

# *Create a method within the graph class for adding an edge between two vertices.*

# *Create a private helper method within the graph class for the DFS traversal:*

# *Mark the current vertex as visited.*

# *Print the current vertex.*

# *For each unvisited neighbor of the current vertex, recursively call the DFS function on that neighbor.*

# *Create a public method in the graph class to start the DFS traversal:*

# *Initialize a boolean array to keep track of visited vertices.*

# *Call the private DFS helper method for the starting vertex.*

# *In the main function:*

# *Create an instance of the graph with the desired number of vertices.*

# *Add edges between vertices to represent the graph's structure.*

# *Call the DFS method with the starting vertex to begin the traversal.*

# *Print the visited vertices as they are traversed.*

# *Compile and run the program to observe the depth-first traversal of the graph starting from a specified vertex.*

# Code:

# *#include <iostream>*

# *#include <list>*

# *using namespace std;*

# *class Graph*

# *{*

# *int V;*

# *list<int> \*adj;*

# *void DFSUtil(int v, bool visited[]);*

# *public:*

# *Graph(int V);*

# *void addEdge(int v, int w);*

# *void DFS(int v);*

# *};*

# *Graph::Graph(int V)*

# *{*

# *this->V = V;*

# *adj = new list<int>[V];*

# *}*

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

# *{*

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

# *}*

# *void Graph::DFSUtil(int v, bool visited[])*

# *{*

# *visited[v] = true;*

# *cout << v << " ";*

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

# *{*

# *if (!visited[\*i])*

# *{*

# *DFSUtil(\*i, visited);*

# *}*

# *}*

# *}*

# *void Graph::DFS(int v)*

# *{*

# *bool \*visited = new bool[V];*

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

# *{*

# *visited[i] = false;*

# *}*

# *DFSUtil(v, visited);*

# *}*

# *int main()*

# *{*

# *Graph g(4);*

# *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";*

# *g.DFS(2);*

# *return 0;*

# *}*

# Observations/Outcome :

# 

# Time Complexity:

* *Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.*