Practical-3.1

Student Name: Rajdeep Jaiswal **UID:** 20BCS2761
Branch: CSE **Section/Group: 902 WM B**
Semester: 05
Subject Name: Design & Analysis Algorithm **Subject Code:** 20CSP-312

1. Aim:

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.

2. Task to be done:

To implement DFS.

3. Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)
Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their
STATUS = 2 (waiting state)[END OF LOOP]
Step 6: EXIT

Code:

```
 #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)
{
   adj[v].push_back(w); // Add w to v's list.
}

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's 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);
```

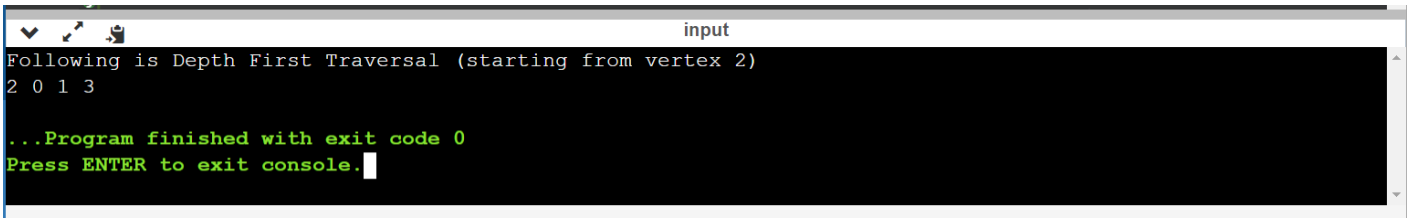
```
    return 0;
}
```

5. **Complexity Analysis:**

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

   Auxiliary Space: O(V), since an extra visited array of size V is required.

6. **Result:**



Learning outcomes (What I have learnt):

1. Learn about searching technique.
2. Learn about time complexity of program.
3. Learnt to implement Depth First Search.