Experiment: 3.1

Student Name: Aniket Kumar
UID: 20BCS5306
Branch: BE-CSE
Section/Group: WM_703-B
Semester: 5th
Subject Code: 20CSP-312
Subject Name: DAA Lab
Date of Performance: 31/10/2022

1. Aim:

Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as

1. To find the topological sort of a directed acyclic graph,
   OR
2. To find a path from source to goal in a maze.

2. Task:

To do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as

3. To find the topological sort of a directed acyclic graph,
   OR
4. To find a path from source to goal in a maze.

3. Software Used:

1. Visual Studio Code
2. MinGW
3. C++ compiler

4. Code:

```
#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)
{ 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()
{
        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;
```

}

5. Output:

```
Following is a Topological Sort of the given graph
5 4 2 3 1 0
```

6. Time Complexity:-

The time complexity of this algorithm will be O(V+E)

Learning outcomes:

1. Learned about Dynamic programming
2. Learned about optimization techniques
3. Learned about the knapsack problem
4. Learned about different ways of solving knapsack problem