Documentation Report: Depth First Search (DFS) with Stack

# Objective

The purpose of this task is to implement Depth First Search (DFS) using an explicit stack (instead of recursion). The algorithm searches for a goal node in a tree structure and returns the sequence of visited nodes.

# Code

tree = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': [],  
 'F': ['G'],  
 'G': []  
}  
def dfs\_stack(start, goal):  
 visited = []  
 stack = [start]  
 while stack:  
 node = stack.pop()  
 if node not in visited:  
 visited.append(node)  
 if node == goal:  
 break  
 stack.extend(reversed(tree[node]))  
 return visited  
print(dfs\_stack('A', 'F'))

# Line by Line Explanation

## Tree Representation

Defines the tree as a dictionary, where each key is a node and its value is a list of child nodes. For example, 'A': ['B', 'C'] means A has two children B and C.

## DFS Function Definition

The function dfs\_stack(start, goal) takes the starting node and the goal node to search for.

## Initialization

visited = [] initializes an empty list to keep track of visited nodes. stack = [start] initializes the stack with the starting node.

## Loop Until Stack is Empty

while stack: continues until the stack becomes empty.

## Pop from Stack

node = stack.pop() removes the last element from the stack (LIFO).

## Check if Node is Already Visited

if node not in visited: ensures nodes are only processed once. visited.append(node) adds the current node to visited.

## Goal Check

if node == goal: break stops the search once the goal is found.

## Add Children to Stack

stack.extend(reversed(tree[node])) adds children of the current node to the stack in reversed order so DFS explores left to right.

## Return Visited Nodes

return visited outputs the traversal path of visited nodes.

## Function Call

print(dfs\_stack('A', 'F')) runs DFS from A to search for F and prints the result.

# Sample Output

['A', 'B', 'D', 'E', 'C', 'F']

Explanation of traversal order:  
- Start at A  
- Explore B → D → E  
- Backtrack and explore C → F (goal found)