# Lab Report: Depth First Search (DFS) using Stack (Iterative)

## Introduction

Depth First Search (DFS) is a fundamental graph traversal algorithm. In this lab, we implemented DFS using an iterative approach with a stack. DFS explores as far as possible along each branch before backtracking, which makes it different from Breadth First Search (BFS).

## Purpose of the Code

The purpose of this code is to demonstrate how DFS can be implemented without recursion. Instead of using recursive function calls, we use a stack data structure to manage the nodes that need to be visited. This helps us understand how DFS works internally.

## Explanation of the Code

1. def dfs\_stack(graph, start):

- This is the function definition for the DFS algorithm. It takes two arguments: the graph and the starting node.

2. visited = set()

- A set is used to store nodes that have already been visited. We use a set because it does not allow duplicates and membership checks are fast.

3. stack = [start]

- A stack (list) is initialized with the starting node. DFS uses stack behavior (Last In First Out) to go deep into the graph.

4. while stack:

- This loop runs until the stack becomes empty, meaning there are no more nodes to visit.

5. ver = stack.pop()

- The last element is popped from the stack. This simulates the depth-first behavior.

6. if ver not in visited:

- If the node has not been visited before, we process it.

7. print(ver, end=' ')

- The current node is printed as part of the traversal order.

8. visited.add(ver)

- Mark the current node as visited to avoid revisiting it later.

9. stack.extend(graph[ver] - visited)

- All unvisited neighbors of the current node are added to the stack. This ensures DFS will continue deeper into the graph.

## Graph Structure

The graph is represented as a dictionary of sets. Each key is a node, and its value is the set of adjacent (neighbor) nodes.

graph = {  
 '0': set(['1','2']),  
 '1': set(['0','3','4']),  
 '2': set(['0']),  
 '3': set(['1']),  
 '4': set(['2','3'])  
}

## Output

The traversal starts from node '0'. The DFS explores deeper into the graph before backtracking. The expected output of the traversal is:  
  
Depth First Traversal (Stack Iterative):  
0 2 1 4 3

## Conclusion

In this lab, we learned how to implement DFS using an iterative method with a stack. We understood the role of each part of the code, why a stack and a visited set are used, and how DFS traversal works step by step. This method avoids recursion and gives us clear control over the traversal process.