# HAMDA HUSSAIN(59)

# DFS Solutions Documentation

This document contains solutions of two questions related to Depth First Search (DFS):  
1. DFS using Stack (without nodes)  
2. DFS Traversals (Inorder, Preorder, Postorder) using Stack (without nodes)  
  
Each question includes code, explanation, and final outputs.

## Question 1: DFS with Stack (Without Nodes)

This implementation performs DFS using a stack and adjacency list representation of a graph.

### Code:

def dfs\_with\_stack(graph, start):  
 visited = []  
 stack = [start]  
  
 while stack:  
 vertex = stack.pop()  
 if vertex not in visited:  
 visited.append(vertex)  
 for neighbor in reversed(graph[vertex]):  
 if neighbor not in visited:  
 stack.append(neighbor)  
 return visited  
  
# Example graph (dictionary form)  
graph = {  
 1: [2, 3],  
 2: [4, 5],  
 3: [6],  
 4: [],  
 5: [],  
 6: []  
}  
  
print(dfs\_with\_stack(graph, 1))

### Final Output:

[1, 2, 4, 5, 3, 6]

A number on a black background

AI-generated content may be incorrect.

## Question 2: DFS Traversals (Inorder, Preorder, Postorder)

This question demonstrates DFS traversals using stack without nodes, where tree is stored in array form.

### 1. Inorder Traversal (LNR)

Visits nodes in order: Left → Node → Right.

def inorder\_iterative(tree):  
 stack = []  
 index = 0  
 result = []  
 while stack or index < len(tree):  
 if index < len(tree):  
 stack.append(index)  
 index = 2 \* index + 1 # Move to left child  
 else:  
 index = stack.pop()  
 result.append(tree[index]) # Visit Node  
 index = 2 \* index + 2 # Move to right child  
 return result

Final Result: 4 2 5 1 6 3

### 2. Preorder Traversal (NLR)

Visits nodes in order: Node → Left → Right.

def preorder\_iterative(tree):  
 if not tree:  
 return []  
 stack = [0]  
 result = []  
 while stack:  
 index = stack.pop()  
 result.append(tree[index]) # Visit Node  
 right = 2 \* index + 2  
 left = 2 \* index + 1  
 if right < len(tree):  
 stack.append(right)  
 if left < len(tree):  
 stack.append(left)  
 return result

Final Result: 1 2 4 5 3 6

### 3. Postorder Traversal (LRN)

Visits nodes in order: Left → Right → Node.

def postorder\_iterative(tree):  
 if not tree:  
 return []  
 stack1 = [0]  
 stack2 = []  
 while stack1:  
 index = stack1.pop()  
 stack2.append(index)  
 left = 2 \* index + 1  
 right = 2 \* index + 2  
 if left < len(tree):  
 stack1.append(left)  
 if right < len(tree):  
 stack1.append(right)  
 return [tree[i] for i in reversed(stack2)]

Final Result: 4 5 2 6 3 1

