**AI LAB – ASSIGNMENT**

**NAME: ABDULLAH**

**ROLL NO: Su92-BSAIM-F24-002**

**SECTION: BSAI-3A**

# Lab 5 Tasks - DFS

## Task 1: DFS with Stack (Without Node)

In this task, we implement Depth First Search (DFS) using a stack instead of recursion. We do not use a Node class, rather we represent the graph using an adjacency list (dictionary).

### Python Code:

def dfs\_stack(graph, start):  
 visited = set()  
 stack = [start]  
  
 while stack:  
 vertex = stack.pop()  
 if vertex not in visited:  
 print(vertex, end=" ")  
 visited.add(vertex)  
 for neighbor in reversed(graph[vertex]):  
 if neighbor not in visited:  
 stack.append(neighbor)  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("DFS Traversal using Stack:")  
dfs\_stack(graph, 'A')

### Explanation:

1. We use a set 'visited' to keep track of visited nodes.  
2. A stack is initialized with the start node.  
3. While the stack is not empty:  
 - Pop the top element.  
 - If it is not visited, print it and mark it visited.  
 - Push all unvisited neighbors into the stack.  
4. The process repeats until the stack is empty.

Example Output: A B D E F C

## Task 2: Inorder, Preorder, Postorder Traversals

In this task, we research and implement the three DFS-based traversals of a binary tree: Inorder, Preorder, and Postorder. The binary tree is represented as a dictionary instead of a Node class.

### Python Code:

tree = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': [None, 'F'],  
 'D': [None, None],  
 'E': [None, None],  
 'F': [None, None]  
}  
  
def preorder(tree, root):  
 if root is None:  
 return  
 print(root, end=" ")  
 preorder(tree, tree[root][0])  
 preorder(tree, tree[root][1])  
  
def inorder(tree, root):  
 if root is None:  
 return  
 inorder(tree, tree[root][0])  
 print(root, end=" ")  
 inorder(tree, tree[root][1])  
  
def postorder(tree, root):  
 if root is None:  
 return  
 postorder(tree, tree[root][0])  
 postorder(tree, tree[root][1])  
 print(root, end=" ")  
  
print("Preorder Traversal:")  
preorder(tree, 'A')  
print("\nInorder Traversal:")  
inorder(tree, 'A')  
print("\nPostorder Traversal:")  
postorder(tree, 'A')

### Explanation of Traversals:

1. Preorder (Root → Left → Right): Visit root first, then left subtree, then right subtree.  
 Example Result: A B D E C F  
  
2. Inorder (Left → Root → Right): Visit left subtree first, then root, then right subtree.  
 Example Result: D B E A C F  
  
3. Postorder (Left → Right → Root): Visit left subtree first, then right subtree, then root.  
 Example Result: D E B F C A