NAME:

M NAUMAN ANWAR

# 1.DFS with Stack and Node

This document explains how Depth First Search (DFS) works using a stack and a Node-based graph structure in Python. DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

## 1. Node Class

We first define a simple Node class. Each node has a value and a list of neighbors that represent connections.

class Node:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.neighbors = [ ]

## 2. DFS Using Stack

We implement DFS using an explicit stack instead of recursion. The algorithm keeps track of visited nodes to avoid repetition.

def dfs\_stack(start):  
 visited = set()   
 stack = [start]   
  
 while stack:  
 node = stack.pop() # Take the last node from the stack  
 if node not in visited:  
 print(node.val, end=" ")   
 visited.add(node)  
  
 for n in reversed(node.neighbors): # Add neighbors to stack  
 if n not in visited:  
 stack.append(n)

## 3. Example Graph

We create a graph with 5 nodes (A, B, C, D, E) and connect them as follows:

a = Node("A")  
b = Node("B")  
c = Node("C")  
d = Node("D")  
e = Node("E")  
  
a.neighbors = [b, c]  
b.neighbors = [d, e]  
c.neighbors = [e]  
d.neighbors = []  
e.neighbors = []

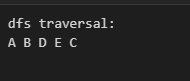
Graph visualization:  
 A  
 / \  
 B C  
 / \ \  
D E E

## 4. Running DFS

Finally, we call the DFS function starting from node A:

print("DFS traversal:")  
dfs\_stack(a)

Expected Output: A B D E C



2.Tree Traversals in Python

## 1. Introduction

This program demonstrates three common types of **Depth First Search (DFS)** traversals on a **binary tree**. Each node in the tree has a value, a left child, and a right child.

The three traversals covered are:

* **Preorder (Root → Left → Right)**
* **Inorder (Left → Root → Right)**
* **Postorder (Left → Right → Root)**

## 2. Node Class

A simple Node class is created to represent each node of the binary tree.

* Each node stores a value (val).
* It has two attributes: left and right (children).
* If a node has no child, the attribute remains None.

## 3. Traversal Functions

### a) Preorder Traversal (Root → Left → Right)

* Visit and print the root node.
* Traverse the left subtree.
* Traverse the right subtree.

**Use case:** Useful for copying a tree or evaluating prefix expressions.

### b) Inorder Traversal (Left → Root → Right)

* Traverse the left subtree.
* Visit and print the root node.
* Traverse the right subtree.

**Use case:** In Binary Search Trees (BSTs), inorder traversal gives **nodes in ascending sorted order**.

### c) Postorder Traversal (Left → Right → Root)

* Traverse the left subtree.
* Traverse the right subtree.
* Visit and print the root node.

**Use case:** Useful for deleting a tree or evaluating postfix expressions.

## 5. Program Output

* **Preorder Traversal:**  
  A B D E C F
* **Inorder Traversal:**  
  D B E A C F
* **Postorder Traversal:**  
  D E B F C A

