# DFS Traversal

## 1. DFS Using Stack (Graph Representation)

This code performs Depth First Search (DFS) using a stack for a graph. It starts from a given node and explores as far as possible along each branch before backtracking. The graph is represented using a dictionary, where each node stores its connected nodes. A stack is used to keep track of nodes to be visited next. When a node is visited, it is printed and marked as explored. Unvisited neighboring nodes are added to the stack in reverse sorted order to maintain proper traversal sequence.

**Main Steps:**

1. Start with the given starting node and push it into the stack.

2. Pop the top node from the stack and visit it.

3. Add all its unvisited neighbors to the stack.

4. Repeat this process until the stack becomes empty.

## 2. DFS Using Stack Inside Node Class (Tree Representation)

This code performs DFS traversal on a tree using a Node class that contains its own internal stack. Each Node object stores data (element) and links to its left and right child nodes. The DFS traversal is done using an explicit stack implemented within the Node class using class methods (push, pop, and is\_empty). The traversal starts from the root node and continues until all nodes are explored.

**Key Working Steps:**

1. Push the starting node (root) into the stack.

2. Pop the top node and print its value.

3. Push its right and left child nodes (right first, so left is processed first).

4. Continue this process until the stack is empty.

This method simulates the DFS traversal process manually without recursion and shows how stack-based logic works inside a class.

## 3. DFS Traversal (Preorder, Inorder, and Postorder)

This code demonstrates three types of Depth First Search traversals on a binary tree using recursion: Preorder, Inorder, and Postorder. Each traversal method uses a recursive approach to visit nodes in a specific order.

Traversal Types and Their Orders:

1. Preorder: Visit Root → Left → Right

2. Inorder: Visit Left → Root → Right

3. Postorder: Visit Left → Right → Root

These three traversals help in understanding different ways of exploring or printing tree nodes. Recursion is used here instead of a stack, but logically, recursion internally uses a call stack.

The tree used in this code has nodes labeled A to X, showing a clear hierarchical structure with left and right subtrees.

## Conclusion

All three codes explain different ways to perform Depth First Search (DFS). The first code works on a graph using a manual stack, the second uses a stack inside a class for a tree, and the third shows recursive DFS traversals (Preorder, Inorder, and Postorder). These approaches help understand how DFS works in both graph and tree structures using iterative and recursive methods.