Binary Tree: DFS (Iterative) — Coding Interview Notes (Light Theme)

General Pattern Template

```
def dfs(root):
    stack = [root]
    ans = 0

    while stack:
        node = stack.pop()
        # do logic
        if node.left:
            stack.append(node.left)
        if node.right:
            stack.append(node.right)

    return ans
```

Concept:
The **Iterative Depth-First Search (DFS)** pattern uses an explicit stack to traverse binary trees without recursion. It simulates the call stack manually, allowing control over traversal order and avoiding recursion depth limits.

Common Use Cases: Preorder, Inorder, Postorder traversals, computing sums, and iterative processing.
Time Complexity: O(n) **Space Complexity:** O(h) — proportional to tree height.

Key Ideas

1. Use a stack to simulate recursive DFS behavior.
2. Pop nodes from stack; push children in reverse order (depends on traversal type).
3. Traversal order is determined by push order (pre/in/post).
4. Iterative DFS avoids recursion depth limits and can be more memory efficient on large trees.

Example 1: Preorder Traversal (Root → Left → Right)

Goal: Perform DFS visiting nodes in Root → Left → Right order.
Approach: Push right child first so left is processed next.

```
def preorder_iterative(root):
    if not root:
        return []
    stack = [root]
    result = []
    while stack:
        node = stack.pop()
        result.append(node.val)
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)

    return result
```

Example 2: Inorder Traversal (Left → Root → Right)

Goal: Perform Inorder traversal without recursion.
Approach: Use a pointer and stack to traverse leftmost nodes first, then process current, then right.

```
def inorder_iterative(root):
    stack = []
    result = []
    curr = root

    while curr or stack:
        while curr:
            stack.append(curr)
            curr = curr.left
        curr = stack.pop()
        result.append(curr.val)
        curr = curr.right

    return result
```

Example 3: Postorder Traversal (Left → Right → Root)

Goal: Visit nodes in Left → Right → Root order using iteration.
Approach: Push (node, visited_flag) tuples to simulate recursion.

```
def postorder_iterative(root):
    if not root:
        return []
    stack = [(root, False)]
    result = []

    while stack:
        node, visited = stack.pop()
        if node:
            if visited:
                result.append(node.val)
            else:
                stack.append((node, True))
                stack.append((node.right, False))
                stack.append((node.left, False))
    return result
```

Example 4: Compute Tree Sum (Iterative)

Goal: Return the sum of all node values using an explicit stack.
Approach: Similar to preorder, accumulate value while traversing nodes.

```
def tree_sum_iterative(root):
    if not root:
        return 0
    stack = [root]
    total = 0
    while stack:
        node = stack.pop()
        total += node.val
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)
    return total
```

Summary Table

Traversal TypeOrderPush Order / LogicComplexity PreorderRoot → Left → RightPush right, then leftO(n) InorderLeft → Root → RightGo left until null, then popO(n) PostorderLeft → Right → RootUse (node, visited) tupleO(n) Custom DFSFlexibleDepends on logic placementO(n)