168, 205. Binary Tree & Binary Search Tree

168- Pre order traversal of a tree both using recursion and iteration.

205- Convert Binary tree into Binary Search tree.

Tree - Tree is a hierarchical data structure consisting of nodes.

Types :- Binary tree - Each node has at most two children.

Binary Search tree - A binary tree where the left child is smaller and the right child is greater than the parent.

Tree Traversal

Visiting all nodes of a tree systematically.

The primary tree traversal methods are:-

* DFS - Depthfirst Traversal.

∴ It explores as far as possible along each brach before backtracking.

→ Preorder (Root → left → Right)

→ Inorder (Left → Root → Right)

→ postorder (Left → Right → Root).

* Breadth - first Traversal

BFS explores nodes level by level using a queue.

Recursion in trees.

- Recursion is a natural for tree problems because trees are inherently recursive structures.
- The recursive function calls itself to traverse subtrees.
- In pre order traversal, recursion process the root, then the left subtree, then the right subtree.

Iterative Tree Traversal using Stacks and Queue

Stack: Used for iterative DFS (Preorder, Inorder, Postorder).

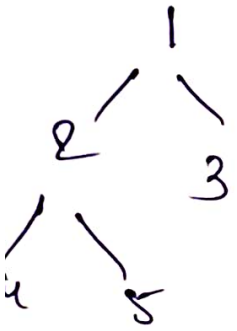
Queue - Used for BFS (level order Traversal).

Recursive preorder Traversal

```
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) {
        this.val = val;
        this.left = this.right = null;
    }
}

Public class preorderTraversal {
    Public static void preorderRecursive(TreeNode root) {
        if (root == null) return;
        System.out.print(root.val + " ");
        PreorderRecursive(root.left);
        PreorderRecursive(root.right);
    }
    psvm(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        System.out.println("Recursive Preorder Traversal:");
        PreorderRecursive(root);
```

if the root is null the function returns
(Stopping condition for recursion).

it the Root node
- print the value of the current node

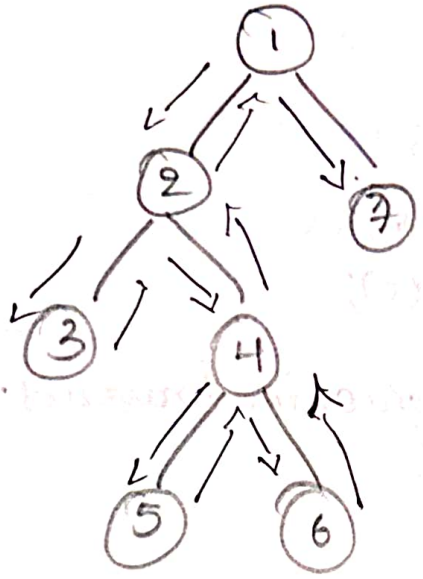


— , 1 2 4 5 3.

O(h) , @function call stack.

Iterative pre order.

Root - left - Right

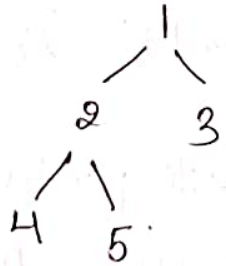


→ 1, 2, 3, 4, 5, 6, 7

```
import java.util.Stack;

Public class IterativePreorderTraversal {

  Public static void preorderIterative (TreeNode root) {

    if (root == = null) return;

    Stack <TreeNode> stack = new Stack <> ();
    stack.push(root); // push the root node into stack. ①
    While (! stack.isEmpty()) {
    TreeNode node = stack.pop(); Pop the top node.
    System.out.println ("node.val + " "); Process the node
      push right child first, then left child
      if (node.right! = null) stack.push(node.right);
  } } if (node.left != null) stack.push (node.left);
```

```
Public static void main (String[] args){
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.left = new TreeNode(4);
    root.left.right = new TreeNode(5);
System.out.println("Iterative preorder Traversal:");
  PreorderIterative(root);
}
```

1) Initialize stack.
   -> create a stack and push the root node
2) Loop until stack is Empty
   -> pop the top node from the stack.
   -> print its value (process the node).
   -> Push the right child first (so that the left child is processed first).
   -> Push the left child next.



-> 1 2 4 5 3.

O(h) h is height of tree.

Find the middle element of a Stack.

1. Stack Data Structure:

- A stack is a linear data structure that follows the LIFO (Last In First Out) principle. It has two main operations.

→ PUSH. - Adds an element to the top of the stack

→ POP - Removes the top element from the stack

→ Peek – Retrieves the top element without removing it.

To find middle element

1) Stack size and Middle Element.

→ The size of the stack determines where the middle element is located.

→ If the size is N, the middle index is

midIndex = N/2.

→ If N = 5 → middle element is at index 2 (0 based Indexl)

→ If N = 6 → middle element is at index 3

Using an Extra stack for storage.

Since a stack only allows access to the top element, we must pop elements until we reach the middle. We store popped elements in a temporary stack and then push them back.

Eg:- Finding Middle in [1, 2, 3, 4, 5]

1. Pop elements until index 2 (Store them in another stack).
2. Peek to find the middle (3).
3. Push elements back to restore the original stack.

Recursion for finding Middle Element.

Recursion is a technique where a function calls itself to solve a smaller problem. The recursive approach:

1. Base case: If the middle index is reached, return the element.
2. Recursive case: Pop an element, find the middle in the smaller stack & push the popped element back.

```
import java.util.Stack;
public class Middle {
    Public static int findMiddle (Stack<Integer> stack)
{
    int size = stack.size();
[   if (size == 0){
        throw new IllegalStateException ("Stack is empty");
    } ] checking for an empty stack.
int midIndex = size/2; // finding the middle index.
Stack<Integer> tempStack = new Stack<>();
                                    // Using a temp Stack.
for (int i=0; i<midIndex; i++){
    tempstack.push(stack.pop());
}                                   moving Element to the temp stack.
int middleElement = stack.peek();   Retrieving the mid element
while (!tempStack.isEmpty()){
    Stack.push(tempStack.pop());    Restoring the original stack.
}
return middleElement;
}
Public static void main (String[] args){   Taking part.
    Stack<Integer> stack = new Stack<>();
     Stack.push(1);
     stack.push(2);
      stack.push(3);
      Stack.push(4);
```

Initialize an empty stack

```
Stack.push(5);
System.out.println("Middle Element: " + findMiddle
                                        (stack));
}
}
```

Time and space complexity Analysis.

finding Middle Element

- O(N) - time complexity
- O(N) - Space complexity

Total Complexity - O(N).