I will show you all how to tackle various tree questions using iterative inorder traversal. First one is the standard iterative inorder traversal using stack. Hope everyone agrees with this solution.

Question : [Binary Tree Inorder Traversal](https://leetcode.com/problems/binary-tree-inorder-traversal/)

public List<Integer> inorderTraversal(TreeNode root) {

List<Integer> list = new ArrayList<>();

if(root == null) return list;

Stack<TreeNode> stack = new Stack<>();

while(root != null || !stack.empty()){

while(root != null){

stack.push(root);

root = root.left;

}

root = stack.pop();

list.add(root.val);

root = root.right;

}

return list;

}

Now, we can use this structure to find the Kth smallest element in BST.

Question : [Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

public int kthSmallest(TreeNode root, int k) {

Stack<TreeNode> stack = new Stack<>();

while(root != null || !stack.isEmpty()) {

while(root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

if(--k == 0) break;

root = root.right;

}

return root.val;

}

We can also use this structure to solve BST validation question.

Question : [Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/)

public boolean isValidBST(TreeNode root) {

if (root == null) return true;

Stack<TreeNode> stack = new Stack<>();

TreeNode pre = null;

while (root != null || !stack.isEmpty()) {

while (root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

if(pre != null && root.val <= pre.val) return false;

pre = root;

root = root.right;

}

return true;

}