# #LeetCode98

https://leetcode.com/problems/validate-binary-search-tree/description/

Yifeng Zeng

## #题目描述

Validate Binary Search Tree

## #思路报告

The first thing is to clarify the defination of the BST input. The BST has a very important property which is that the in-order traversal of a BST is a non-decreasing sequence or a increasing sequence (discuss this with the interviewer, LC uses increasing sequence). So the primitive idea is to do an in-order traversal of this TreeNode, and find if the current element is smaller than the previous element. If it is, then it is not a BST, otherwise it is.

The non-recursive code that I previous recite is this:

代码如下:

```
public boolean isValidBST(TreeNode root) {
    if (root == null) {
        return true;
    }
    Deque<TreeNode> stack = new ArrayDeque<>();
    TreeNode cur = root;
    TreeNode prev = null;
    Boolean isFirst = true;

    while (!stack.isEmpty() || cur != null) {
        while (cur != null) {
            stack.push(cur);
            cur = cur.left;
        }
        cur = stack.pop();
        if (isFirst) {
            prev = cur;
            isFirst = false;
        } else {
            if (cur.val <= prev.val) {
                return false;
            }
            prev = cur;
        }
        cur = cur.right;
    }

    return true;
}
```

From one of Qinyuan's lecture, the non-recursive code can be something like this, and the order can be any of pre/in/post-order. More details will be explained in the report of LC145 Binary Tree Postorder Traversal.

Code:

```
class Pair {
    TreeNode node;
    boolean print;
    public Pair(TreeNode node, boolean print) {
        this.node = node;
        this.print = print;
    }
}

public boolean isValidBST(TreeNode root) {
    if (root == null) {
        return true;
    }

    Deque<Pair> stack = new ArrayDeque<>();
    stack.push(new Pair(root, false));
    boolean isFirst = true;
    TreeNode prev = null;

    while (!stack.isEmpty()) {
        Pair cur = stack.pop();
        if (cur.node == null) {
            continue;
        }

        if (cur.print) {
            if (isFirst) {
                isFirst = false;
            } else {
                if (cur.node.val <= prev.val) {
                    return false;
                }
            }
            prev = cur.node;
        } else {
            stack.push(new Pair(cur.node.right, false));
            stack.push(new Pair(cur.node, true));
            stack.push(new Pair(cur.node.left, false));
        }
    }

    return true;
}
```

We can also solve it without traversal. The defination is that all the left children has smaller values than the root value, all the right children has larger values than the root value and both left and right child is also a BST. Then for the current root, I can just see if the largest number in all the left children is smaller than the root value and see if the smallest number in all the right children is larger than the root value. So we can use divid and conquer. We need to see the smallest and largest number in the root's children so we might need a wrapper for return.

Code:

```
class ReturnType {
    long max;
    long min;
    boolean isValid;
    public ReturnType(long max, long min, boolean isValid) {
        this.max = max;
        this.min = min;
        this.isValid = isValid;
    }
}

public boolean isValidBST(TreeNode root) {
    return helper(root).isValid;
}

private ReturnType helper(TreeNode root) {
    ReturnType res = new ReturnType(Long.MIN_VALUE, Long.MAX_VALUE, true);
    if (root == null) {
        return res;
    }

    res.min = root.val;
```

```
        res.max = root.val;

        ReturnType left = helper(root.left);
        ReturnType right = helper(root.right);

        res.isValid = left.isValid && right.isValid && left.max < root.val && root.val < r

        res.min = Math.min(res.min, left.min);
        res.min = Math.min(res.min, right.min);
        res.max = Math.max(res.max, left.max);
        res.max = Math.max(res.max, right.max);

        return res;
    }
```

## #套路总结

- When starting to code, ask if root == null is true or false.
- The BST has a very important property which is that the in-order traversal of a BST is a non-decreasing sequence or a increasing sequence (LC uses increasing sequence).