Validate Binary Search Tree

Given a binary tree, determine if it is a valid binary search tree (BST).

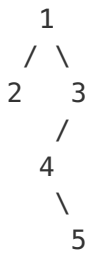
Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys**less than** the node's key.
- The right subtree of a node contains only nodes with keys**greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:



The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

Solution 1

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        TreeNode* prev = NULL;
        return validate(root, prev);
    }
    bool validate(TreeNode* node, TreeNode* &prev) {
        if (node == NULL) return true;
        if (!validate(node->left, prev)) return false;
        if (prev != NULL && prev->val >= node->val) return false;
        prev = node;
        return validate(node->right, prev);
    }
};
```

> Update:
>
> If we use in-order traversal to serialize a binary search tree, we can get a list of values in ascending order. It can be proved with the definition of BST. And here I use the reference of TreeNode pointer `prev` as a global variable to mark the address of previous node in the list.
>
> "In-order Traversal": https://en.wikipedia.org/wiki/Tree_traversal#In-order

If you know what `INT_MAX` or `INT_MIN` is, then it is no excuse for your carelessness.

written by jakwings original link here

Solution 2

```
public class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    public boolean isValidBST(TreeNode root, long minVal, long maxVal) {
        if (root == null) return true;
        if (root.val >= maxVal || root.val <= minVal) return false;
        return isValidBST(root.left, minVal, root.val) && isValidBST(root.right,
root.val, maxVal);
    }
}
```

Basically what I am doing is recursively iterating over the tree while defining interval `<minVal, maxVal>` for each node which value must fit in.

written by sruzic original link here

Solution 3

```
bool isValidBST(TreeNode* root) {
    return isValidBST(root, NULL, NULL);
}

bool isValidBST(TreeNode* root, TreeNode* minNode, TreeNode* maxNode) {
    if(!root) return true;
    if(minNode && root->val <= minNode->val || maxNode && root->val >= maxNode->va
l)
        return false;
    return isValidBST(root->left, minNode, root) && isValidBST(root->right, root,
maxNode);
}
```

written by jaewoo original link here