Documentation for Validating a Binary Search Tree (BST)

Problem Statement

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

A valid 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.

Example 1:

Input: root = [2,1,3]

Output: true

Example 2:

Input: root = [5,1,4,null,null,3,6]

Output: false

Explanation: The root node's value is 5 but its right child's value is 4.

Constraints:

- The number of nodes in the tree is in the range [1, 10^4].
- -2^31 <= Node.val <= 2^31 1

Solution

To determine if a given binary tree is a valid BST, we need to ensure that every node in the tree adheres to the BST properties. This can be achieved using a recursive approach where each node is validated based on a permissible range of values. Initially, the entire range of possible values is considered. As we traverse the tree, we narrow this range based on the properties of the BST.

Explanation

1. **TreeNode Class:**
   - This class defines the structure of a node in the binary tree. Each node has a value (val), a left child (left), and a right child (right).

2. **Solution Class:**
   - The isValidBST method is the main function that initiates the validation process.
   - The validate function is a helper function that performs the recursive validation.
     - *It takes three parameters:* the current node (node), the lower bound (low), and the upper bound (high).

- If the current node is None, it returns True, as an empty subtree is a valid BST.
- It checks if the current node's value falls within the valid range (low < node.val < high). If not, it returns False.
- It recursively checks the left subtree with the updated upper bound (current node's value) and the right subtree with the updated lower bound (current node's value).

Usage

To use the solution:

1. Create the binary tree by instantiating TreeNode objects and linking them accordingly.
2. Instantiate the Solution class.
3. Call the isValidBST method with the root of the binary tree.

Example 1

```
root = TreeNode(2, TreeNode(1), TreeNode(3))

sol = Solution()

print(sol.isValidBST(root))  # Output: True
```

Example 2

```
root = TreeNode(5, TreeNode(1), TreeNode(4, TreeNode(3), TreeNode(6)))

sol = Solution()

print(sol.isValidBST(root))  # Output: False
```

Conclusion

The provided solution efficiently validates whether a binary tree is a BST by recursively ensuring that each node's value adheres to the constraints defined by the BST properties. The approach leverages the permissible value range for each node, ensuring that the tree maintains the correct structure throughout its subtrees.