LC 98. Validate Binary Search Tree

Question

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.

Example 1:

```
    2
   / \
  1   3

Input: [2,1,3]
Output: true
```

Example 2:

```
    5
   / \
  1   4
     / \
    3   6

Input: [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.
```

Solution

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isValidBST(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
```

```
        #Solution 2
        #One solution would be to check every parent as we work down the tree but
it creates a lot of complicated logic.
        #So why don't we check every child on the way up! All we have to do is
keep track of the mininum and maximum valid values on the way down.
        #    2
        #   / \
        #  1   3

        #1 // -inf < 1 < 2, so it's still valid
        #3 //  2 < 3 < inf, so it's still a valid tree
        #2 // -inf < 2 < inf, so it's a valid tree!

        #    5
        #   / \
        #  1   4

        #1 // -inf < 1 < 5, so it's still a valid tree
        #4 // 5 > 4 < inf, this tree is not a valid binary tree!

To prevent some extra checks let's start with -infinity and infinity.
        return self.check_bst(root, float("-inf"), float("inf"))

    def check_bst(self, node, left, right):
        if not node:
            return True

        if not left < node.val < right:
            return False

        return (self.check_bst(node.left, left, node.val)
                and self.check_bst(node.right, node.val, right))


        #Solution
        if not root:
            return True
        stack, prev = [], -float('inf')
        while stack or root:
            if root:
                stack.append(root)
                root = root.left
            else:
                node = stack.pop()
                root = node.right
                if node:
```

```
                if node.val <= prev:
                    return False
                prev = node.val
        return True
```