Daily Coding Problem #89

Problem

This problem was asked by LinkedIn.

Determine whether a tree is a valid binary search tree.

A binary search tree is a tree with two children, `left` and `right`, and satisfies the constraint that the key in the `left` child must be less than or equal to the root and the key in the `right` child must be greater than or equal to the root.

Solution

To solve this problem, we need to recall the definition of a binary search tree. Each node of a BST has the following properties:

- A node's left subtree contains only nodes with keys less than the nodes' key.
- A node's right subtree contains only nodes with keys greater than the nodes' key.
- Both the left and right subtrees must be valid BSTs.

From the properties above, we can construct a recursive solution. It's tempting to write a solution which checks whether the left node's key is less than the current node's key and the right node's key is greater than the current node's key. However, we have to make sure that the property holds for the entire subtree, not just the children. For example, the following binary tree would be considered valid if we only checked the children:

```
  2
 / \
1   3
 \
  4
```

We can iterate through the entire left and right subtrees to determine whether the keys are valid. However, the work would be doing can be simplified into one recursive method.

Let's call our recursive method `is_bst()`. At each call in our recursive method, we can maintain a range of valid values for the node's keys -- we'll call the lower bound `min_key` and upper bound `max_key`. If the current node's key is *outside* the range of `min_key` to `max_key`, then return `false`. Otherwise, we call the method on the left and right child nodes, returning `true` if both calls return `true`. If a node is `null`, we should return `true`.

When we call `is_bst()` on the children, we limit the range of valid keys based on our current key. If we call `is_bst()` on the *left* node, then `min_key` should remain the same, while `max_key` should be updated to the current node's key. Similarly, if we call `is_bst()` on the *right* node, then `max_key` should remain the same, while `min_key` should be updated to the current node's key.

```
class TreeNode:
    def __init__(self, key):
```

```
        self.left = None
        self.right = None
        self.key = key

def is_bst(root):
    def is_bst_helper(root, min_key, max_key):
        if root is None:
            return True
        if root.key <= min_key or root.key >= max_key:
            return False
        return is_bst_helper(root.left, min_key, root.key) and \
               is_bst_helper(root.right, root.key, max_key)

    return is_bst_helper(root, float('-inf'), float('inf'))
```

The time complexity of this solution is `O(N)`, as it requires visiting every node in the tree.