Binary Search Tree Check - SOLUTION

Problem Statement

Given a binary tree, check whether it’s a binary search tree or not.

Again, no solution cell, just worry about your code making sense logically. Hint: Think about tree traversals.

Solution

Here is a simple solution- If a tree is a binary search tree, then traversing the tree inorder should lead to sorted order of the values in the tree. So, we can perform an inorder traversal and check whether the node values are sorted or not.

In [3]:
```
tree_vals = []

def inorder(tree):
    if tree != None:
        inorder(tree.getLeftChild())
        tree_vals.append(tree.getRootVal())
        inorder(tree.getRightChild())

def sort_check(tree_vals):
    return tree_vals == sorted(tree_vals)

inorder(tree)
sort_check(tree_vals)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-a22266754f2e> in <module>
     10     return tree_vals == sorted(tree_vals)
     11 
---> 12 inorder(tree)
     13 sort_check(tree_vals)

NameError: name 'tree' is not defined
```

Another classic solution is to keep track of the minimum and maximum values a node can take. And at each node we will check whether its value is between the min and max values it’s allowed to take. The root can take any value between negative infinity and positive infinity. At any node, its left child should be smaller than or equal than its own value, and similarly the right child should be larger than or equal to. So during recursion, we send the current value as the new max to our left child and send the min as it is without changing. And to the right child, we send the current value as the new min and send the max without changing.

In [2]:
```
class Node:
    def __init__(self, k, val):
        self.key = k
        self.value = val
```

```python
        self.left = None
        self.right = None

def tree_max(node):
    if not node:
        return float("-inf")
    maxleft  = tree_max(node.left)
    maxright = tree_max(node.right)
    return max(node.key, maxleft, maxright)

def tree_min(node):
    if not node:
        return float("inf")
    minleft  = tree_min(node.left)
    minright = tree_min(node.right)
    return min(node.key, minleft, minright)

def verify(node):
    if not node:
        return True
    if (tree_max(node.left) <= node.key <= tree_min(node.right) and
        verify(node.left) and verify(node.right)):
        return True
    else:
        return False

root= Node(10, "Hello")
root.left = Node(5, "Five")
root.right= Node(30, "Thirty")

print(verify(root)) # prints True, since this tree is valid

root = Node(10, "Ten")
root.right = Node(20, "Twenty")
root.left = Node(5, "Five")
root.left.right = Node(15, "Fifteen")

print(verify(root)) # prints False, since 15 is to the left of 10
```

```
True
False
```

This is a classic interview problem, so feel free to just Google search "Validate BST" for more information on this problem!

Good Job!