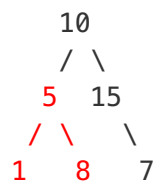
Largest BST Subtree

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.

Note:
A subtree must include all of its descendants.
Here's an example:



The Largest BST Subtree in this case is the highlighted one.
The return value is the subtree's size, which is 3.

1. You can recursively use algorithm similar to 98. Validate Binary Search Tree at each node of the tree, which will result in O(nlogn) time complexity.

Follow up:
Can you figure out ways to solve it with O(n) time complexity?

Solution 1

> edited code: thanks @hyj143 and @petrichory

```
public class Solution {

    class Result {  // (size, rangeLower, rangeUpper) -- size of current tree, ra
nge of current tree [rangeLower, rangeUpper]
        int size;
        int lower;
        int upper;

        Result(int size, int lower, int upper) {
            this.size = size;
            this.lower = lower;
            this.upper = upper;
        }
    }

    int max = 0;

    public int largestBSTSubtree(TreeNode root) {
        if (root == null) { return 0; }
        traverse(root);
        return max;
    }

    private Result traverse(TreeNode root) {
        if (root == null) { return new Result(0, Integer.MAX_VALUE, Integer.MIN_V
ALUE); }
        Result left = traverse(root.left);
        Result right = traverse(root.right);
        if (left.size == -1 || right.size == -1 || root.val <= left.upper || root
.val >= right.lower) {
            return new Result(-1, 0, 0);
        }
        int size = left.size + 1 + right.size;
        max = Math.max(size, max);
        return new Result(size, Math.min(left.lower, root.val), Math.max(right.up
per, root.val));
    }
}
```

```java
/*
    in brute-force solution, we get information in a top-down manner.
    for O(n) solution, we do it in bottom-up manner, meaning we collect informati
on during backtracking.
*/
public class Solution {

    class Result {  // (size, rangeLower, rangeUpper) -- size of current tree, ra
nge of current tree [rangeLower, rangeUpper]
        int size;
        int lower;
        int upper;

        Result(int size, int lower, int upper) {
            this.size = size;
            this.lower = lower;
            this.upper = upper;
        }
    }

    int max = 0;

    public int largestBSTSubtree(TreeNode root) {
        if (root == null) { return 0; }
        traverse(root, null);
        return max;
    }

    private Result traverse(TreeNode root, TreeNode parent) {
        if (root == null) { return new Result(0, parent.val, parent.val); }
        Result left = traverse(root.left, root);
        Result right = traverse(root.right, root);
        if (left.size==-1 || right.size==-1 || root.val<left.upper || root.val>ri
ght.lower) {
            return new Result(-1, 0, 0);
        }
        int size = left.size + 1 + right.size;
        max = Math.max(size, max);
        return new Result(size, left.lower, right.upper);
    }
}
```

written by mach7 original link here

Solution 2

```
def largestBSTSubtree(self, root):
    def dfs(root):
        if not root:
            return 0, 0, float('inf'), float('-inf')
        N1, n1, min1, max1 = dfs(root.left)
        N2, n2, min2, max2 = dfs(root.right)
        n = n1 + 1 + n2 if max1 < root.val < min2 else float('-inf')
        return max(N1, N2, n), n, min(min1, root.val), max(max2, root.val)
    return dfs(root)[0]
```

My `dfs` returns four values:

- `N` is the size of the largest BST in the tree.
- If the tree is a BST, then `n` is the number of nodes, otherwise it's -infinity.
- If the tree is a BST, then `min` and `max` are the minimum/maximum value in the tree.

written by StefanPochmann original link here

Solution 3

```
public int largestBSTSubtree(TreeNode root) {
    if (root == null) return 0;
    if (root.left == null && root.right == null) return 1;
    if (isValid(root, null, null)) return countNode(root);
    return Math.max(largestBSTSubtree(root.left), largestBSTSubtree(root.right));
}

public boolean isValid(TreeNode root, Integer min, Integer max) {
    if (root == null) return true;
    if (min != null && min >= root.val) return false;
    if (max != null && max <= root.val) return false;
    return isValid(root.left, min, root.val) && isValid(root.right, root.val, max)
;
}

public int countNode(TreeNode root) {
    if (root == null) return 0;
    if (root.left == null && root.right == null) return 1;
    return 1 + countNode(root.left) + countNode(root.right);
}
```

written by hollysinka original link here