LeetCode 776

https://leetcode.com/problems/split-bst/description/

Yifeng Zeng

Description

776. Split BST

Given a Binary Search Tree (BST) with root node root, and a target value V, split the tree into two subtrees where one subtree has nodes that are all smaller or equal to the target value, while the other subtree has all nodes that are greater than the target value. It's not necessarily the case that the tree contains a node with value V.

Additionally, most of the structure of the original tree should remain. Formally, for any child C with parent P in the original tree, if they are both in the same subtree after the split, then node C should still have the parent P.

You should output the root TreeNode of both subtrees after splitting, in any order.

Example 1:

Input: root = [4,2,6,1,3,5,7], V = 2

Output: [[2,1],[4,3,6,null,null,5,7]]

Explanation:
Note that root, output[0], and output[1] are TreeNode objects, not arrays.

The given tree [4,2,6,1,3,5,7] is represented by the following diagram:

```
      4
    /   \
  2      6
 / \    / \
1   3  5   7
```

while the diagrams for the outputs are:

```
          4
        /   \
      3      6      and    2
            / \           /
           5   7         1
```

Idea Report

Suppose our out put is TreeNode[] res. The is a BST, so for any node, if it's value is equal or smaller than 'V', the root itself and its left children should be in res[0], but we don't know about root.right, so we need to do a recursion. On the contrary, if a node's value is larger than 'V', we know for sure itself and its right children should be in res[1], but we don't know about root.left, so we need to do a recursion from here. The temp result returned from recursion contains the correct answer for root.left or root.right. If root.val equal or smaller than 'V', then temp result contains root.right information, so at current level, root.right = temp[0] because temp[0] has all the nodes that values are equal or smaller than 'V', and we already know root and root.left are all equal or smaller than 'V', so we put root into res[0]. And from recursion, temp[1] has all the node larger than 'V' in root.right, so at current level, res[1] = temp[1]. On contrary, if root.val is larger than 'V', then root.left = temp[1] because temp[1] has all the node that are larger than 'V', and we also know root and root.right has all the node larger than 'V' at current level, so now root has all the nodes that larger than 'V', so res[1] = root at current level. And res[0] = temp[0] because temp[0] has all the nodes that equal or smaller 'V' from recursion.

Code:

```
class Solution {
    public TreeNode[] splitBST(TreeNode root, int V) {
        TreeNode[] res = new TreeNode[2];
        if (root == null) {
            return res;
        }

        if (root.val <= V) {
            TreeNode[] temp = splitBST(root.right, V);
            root.right = temp[0];
            res[0] = root;
            res[1] = temp[1];
        } else {
            TreeNode[] temp = splitBST(root.left, V);
            root.left = temp[1];
            res[0] = temp[0];
            res[1] = root;
        }

        return res;
    }
}
```

Summary

- Devide and Conquer, only focus one things we need to do at current recursion level, and left the recursion function to handel root.left and root.right.