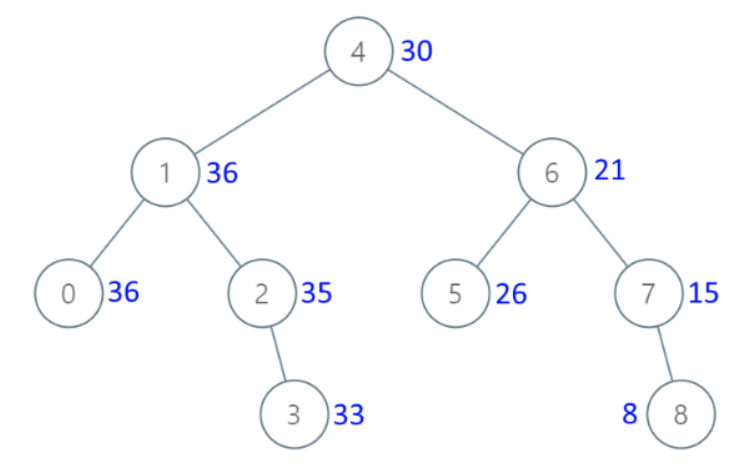
<https://leetcode.com/problems/convert-bst-to-greater-tree/description/>

Given the root of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

* 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:**



Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]

Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]

**Example 2:**

Input: root = [0,null,1]

Output: [1,null,1]

**Constraints:**

The number of nodes in the tree is in the range [0, 104].

-104 <= Node.val <= 104

All the values in the tree are **unique**.

root is guaranteed to be a valid binary search tree.

**Note:** This question is the same as 1038: <https://leetcode.com/problems/binary-search-tree-to-greater-sum-tree/>

**Attempt 1: 2022-12-11**

**Solution 1:  Reverse inorder recursive traversal**

**Style 1: With global variable (10 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

int sum = 0;

public TreeNode convertBST(TreeNode root) {

if(root == null) {

return null;

}

convertBST(root.right);

sum += root.val;

root.val = sum;

convertBST(root.left);

return root;

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Style 2: Without global variable but helper method (30 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public TreeNode convertBST(TreeNode root) {

helper(root, 0);

return root;

}

private int helper(TreeNode root, int sum) {

if(root == null) {

return sum;

}

int right = helper(root.right, sum);

int left = helper(root.left, root.val + right);

root.val += right;

// Tricky point: return 'left'

return left;

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Solution 2:  Reverse inorder iterative traversal (10 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public TreeNode convertBST(TreeNode root) {

int sum = 0;

Stack<TreeNode> stack = new Stack<TreeNode>();

// Assign root to a new node 'cur' for traverse and update any node

// we need to do this because we have to reserve original 'root' for

// return the whole tree but also need to update tree nodes with new

// node 'cur'

TreeNode cur = root;

while(cur != null || !stack.isEmpty()) {

while(cur != null) {

stack.push(cur);

cur = cur.right;

}

cur = stack.pop();

sum += cur.val;

cur.val = sum;

cur = cur.left;

}

return root;

}

}

Time Complexity: O(N)

Space Complexity: O(N)