# 题目

给出二叉搜索树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点node的新值等于原树中大于或等于node.val的值之和。

提醒一下，二叉搜索树满足下列约束条件：

节点的左子树仅包含键小于节点键的节点。

节点的右子树仅包含键大于节点键的节点。

左右子树也必须是二叉搜索树。

注意：本题和1038相同:

https://leetcode-cn.com/problems/binary-search-tree-to-greater-sum-tree/

# 分析

通常的中序遍历是先遍历左子树，再遍历根节点，最后遍历左子树，因为二叉搜索树的特点，所以最终是按照节点的值从小到大遍历。因为题目中要求，将二叉搜索树的每个节点的值替换成树中大于或者等于该节点值的所有节点值之和，所以需要按照节点值从大到小遍历。如果按照从大到小遍历，只要先遍历右子树，再遍历根节点，最后遍历左子树即可。

下面提供递归和迭代两种写法，若二叉树的节点数为n，深度为h，则它们的时间复杂度均为O(n)，空间复杂度均为O(h)。

## 方法一：递归法

class Solution {

public:

TreeNode\* convertBST(TreeNode\* root) {

sum = 0;

dfs(root);

return root;

}

private:

int sum;

void dfs(TreeNode\* root) {

if (root == nullptr) {

return;

}

dfs(root->right);

sum += root->val;

root->val = sum;

dfs(root->left);

}

};

## 方法二：迭代法

class Solution {

public:

TreeNode\* convertBST(TreeNode\* root) {

stack<TreeNode\*> sta;

TreeNode\* cur = root;

int sum = 0;

while (cur != nullptr || !sta.empty()) {

while (cur != nullptr) {

sta.push(cur);

cur = cur->right;

}

cur = sta.top();

sta.pop();

sum += cur->val;

cur->val = sum;

cur = cur->left;

}

return root;

}

};