# 题目

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

节点的左子树只包含小于当前节点的数。

节点的右子树只包含大于当前节点的数。

所有左子树和右子树自身必须也是二叉搜索树。

**示例1:**

输入:

2

/ \

1 3

输出: true

**示例 2:**

输入:

5

/ \

1 4

/ \

3 6

输出: false

解释: 输入为: [5,1,4,null,null,3,6]。

根节点的值为 5 ，但是其右子节点值为 4 。

# 分析

## 方法一：递归法

要判断一个二叉树是否是有效的二叉搜索树（BST），可以使用中序遍历。由于 BST 的中序遍历结果是有序的，因此可以在中序遍历的过程中判断当前节点的值是否大于前一个节点的值。

具体步骤如下：

1、使用递归进行中序遍历，同时传入一个指针prev用于指向前一个节点的值。

2、在递归过程中，首先递归遍历左子树，然后判断当前节点的值是否大于prev，如果大于，则更新prev为当前节点的值，否则返回 false。

3、最后递归遍历右子树。

注：这个与LeetCode 108将有序数组转换为二叉搜索树。

代码：

class Solution {

public:

bool isValidBST(TreeNode\* root) {

long long prev = LLONG\_MIN; // 使用 long long 类型来避免溢出问题

return inorderTraversal(root, prev);

}

bool inorderTraversal(TreeNode\* root, long long& prev) {

if (!root) return true;

if (!inorderTraversal(root->left, prev)) return false;

if (root->val <= prev) return false;

prev = root->val;

return inorderTraversal(root->right, prev);

}

};

这个算法的时间复杂度是 O(n)，其中 n 是节点数。

或：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

bool helper(TreeNode\* root, long long lower, long long upper) {

if (root == nullptr) return true;

if (root -> val <= lower || root -> val >= upper) return false;

return helper(root -> left, lower, root -> val) && helper(root -> right, root -> val, upper);

}

bool isValidBST(TreeNode\* root) {

return helper(root, LONG\_MIN, LONG\_MAX);

}

};