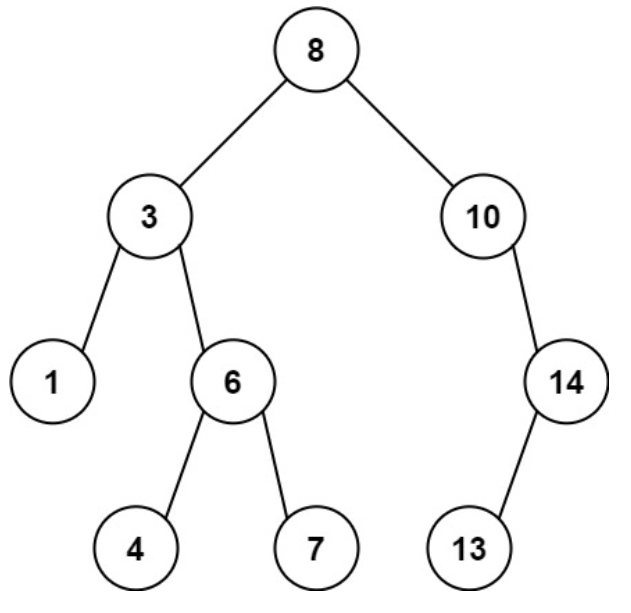
# 题目

给定二叉树的根节点root，找出存在于不同节点A和B之间的最大值V，其中V = |A.val - B.val|，且A是B的祖先。

（如果A的任何子节点之一为B，或者A的任何子节点是B的祖先，那么我们认为A是B的祖先）

示例 1：



输入：root = [8,3,10,1,6,null,14,null,null,4,7,13]

输出：7

解释：

我们有大量的节点与其祖先的差值，其中一些如下：

|8 - 3| = 5

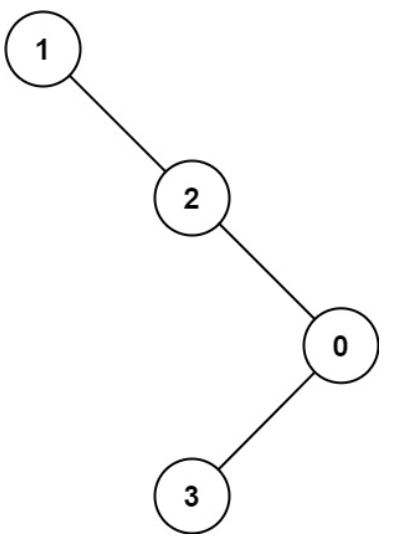
|3 - 7| = 4

|8 - 1| = 7

|10 - 13| = 3

在所有可能的差值中，最大值 7 由 |8 - 1| = 7 得出。

示例 2：



输入：root = [1,null,2,null,0,3]

输出：3

提示：

树中的节点数在2到5000之间。

0 <= Node.val <= 105

# 分析

## 方法一：深度优先搜索

思路：

题目要求找出所有祖先节点与它的子孙节点的绝对差值的最大值。按照枚举的思路，我们可以枚举子孙节点，然后找出它的所有祖先节点，计算绝对差值。同样地，我们也可以枚举祖先节点，然后找出它的所有子孙节点，计算绝对差值。

以第一种思路为例，并非所有祖先节点都需要被考虑到，我们只需要获取最小的祖先节点以及最大的祖先节点。我们对二叉树执行深度优先搜索，并且记录搜索路径上的节点的最小值mi与最大值ma。假设当前搜索的节点值为val，那么与该子孙节点与它的所有祖先节点的绝对差值最大值为 max(∣val−mi∣,∣val−ma∣)，搜索该节点的左子树与右子树时，对应的mi=min(mi,val)，ma=max(ma,val)。

为什么只需要获取最小的祖先节点以及最大的祖先节点?

假设某一子孙节点为x，对应的最小的祖先节点为mi，最大的祖先节点为ma。有任一祖先节点为y，显然mi≤y≤ma。如果x≤y，那么∣x−y∣=y−x≤ma−x=∣x−ma∣，如果 x>y，那么∣x−y∣=x−y≤x−mi=∣x−mi∣，因此最大的绝对差值与祖先节点y无关。

第二种思路是否可行？

可行，需要返回当前子树的最小值和最大值，方法类似。

代码：

class Solution {

public:

int maxAncestorDiff(TreeNode\* root) {

if (!root) return 0;

int result = 0;

dfs(root, root->val, root->val, result);

return result;

}

void dfs(TreeNode\* node, int curMax, int curMin, int& result) {

if (!node) return;

// 更新当前最大值和最小值

curMax = max(curMax, node->val);

curMin = min(curMin, node->val);

// 计算当前节点的最大差值，并更新结果

result = max(result, abs(curMax - curMin));

// 递归遍历左右子树

dfs(node->left, curMax, curMin, result);

dfs(node->right, curMax, curMin, result);

}

};

这个算法使用了深度优先搜索，每次递归时更新当前节点到根节点的最大值和最小值，并计算当前节点的最大差值，最终得到存在于不同节点A和B之间的最大值V。

或：

class Solution {

public:

int dfs(TreeNode \*root, int min\_val, int max\_val) {

if (root == nullptr) return 0;

int diff = max(abs(root->val - min\_val), abs(root->val - max\_val));

min\_val = min(min\_val, root->val);

max\_val = max(max\_val, root->val);

diff = max(diff, dfs(root->left, min\_val, max\_val));

diff = max(diff, dfs(root->right, min\_val, max\_val));

return diff;

}

int maxAncestorDiff(TreeNode\* root) {

return dfs(root, root->val, root->val);

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是二叉树的节点数目。遍历二叉树的所有节点需要O(n)。

空间复杂度：O(n)。最坏情况下，二叉树退化为链表，递归栈的空间为O(n)。