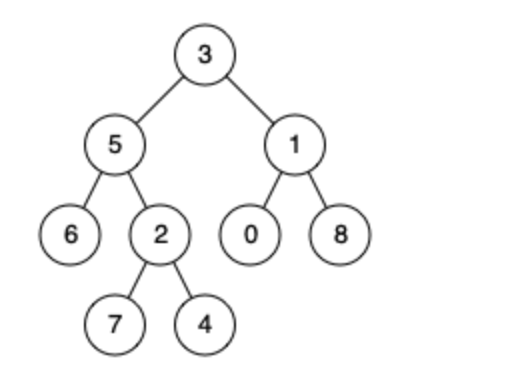
# 题目

给定一棵二叉树的根节点 root，返回给定节点 p 和 q 的最近公共祖先（LCA）节点。如果 p 或 q 之一 不存在 于该二叉树中，返回 null。树中的每个节点值都是互不相同的。

根据维基百科中对最近公共祖先节点的定义：“两个节点 p 和 q 在二叉树 T 中的最近公共祖先节点是 后代节点 中既包括 p 又包括 q 的最深节点（我们允许 一个节点为自身的一个后代节点 ）”。一个节点 x 的 后代节点 是节点 x 到某一叶节点间的路径中的节点 y。

示例 1:

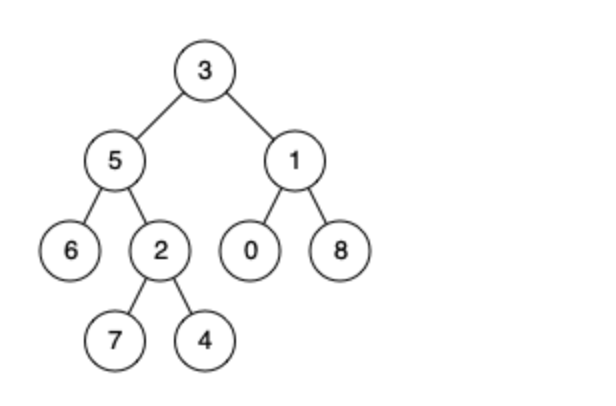


输入： root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

输出： 3

解释： 节点 5 和 1 的共同祖先节点是 3。

示例 2:

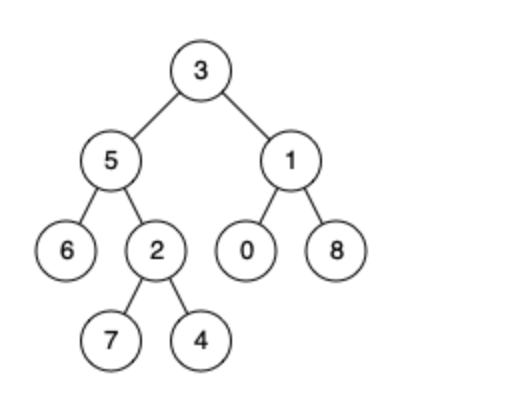


输入： root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

输出： 5

解释： 节点 5 和 4 的共同祖先节点是 5。根据共同祖先节点的定义，一个节点可以是自身的后代节点。

示例 3:



输入： root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 10

输出： null

解释： 节点 10 不存在于树中，所以返回 null。

提示:

树中节点个数的范围是 [1, 104]

-109 <= Node.val <= 109

所有节点的值 Node.val 互不相同

p != q

进阶：在不检查节点是否存在的情况下，你可以遍历树找出最近公共祖先节点吗？

# 分析

要解决这个问题，我们需要找到二叉树中两个给定节点 `p` 和 `q` 的最近公共祖先（LCA）。如果 `p` 或 `q` 中有一个不存在于树中，则返回 `null`。

思路分析：

1、核心定义：最近公共祖先是指既为 `p` 的祖先又为 `q` 的祖先的最深节点（一个节点可以是自身的祖先）。

2、关键问题：需要先确认 `p` 和 `q` 都存在于树中，再寻找它们的 LCA。

3、递归遍历：通过后序遍历（左→右→根）检查每个节点的左右子树中是否包含 `p` 或 `q`，并根据检查结果确定 LCA：

- 若当前节点是 `p` 或 `q`，则返回当前节点（标记该子树包含目标节点）。

- 若左子树和右子树分别包含 `p` 和 `q`（或反之），则当前节点是 LCA。

- 若只有左子树包含目标节点，则返回左子树的结果；若只有右子树包含，则返回右子树的结果。

4、存在性检查：遍历结束后，需确认 `p` 和 `q` 都被找到，否则返回 `null`。

代码：

/\*\*

\* 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 {

private:

TreeNode\* lca = nullptr; // 存储找到的最近公共祖先

// 递归函数：检查以root为根的子树中是否包含p或q，同时寻找LCA

bool dfs(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

if (root == nullptr) return false;

// 递归检查左子树

bool left = dfs(root->left, p, q);

// 递归检查右子树

bool right = dfs(root->right, p, q);

// 情况1：当前节点是p或q

bool curr = (root == p) || (root == q);

// 情况2：当前节点是LCA的条件：

// （左子树包含一个，右子树包含另一个） 或 （当前节点是一个，子树包含另一个）

if ((left && right) || (curr && (left || right))) {

lca = root;

}

// 返回当前子树是否包含p或q

return left || right || curr;

}

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

// 检查p和q是否都在树中，并寻找LCA

bool hasP = dfs(root, p, q);

// 重置lca，再次检查q（因为第一次dfs可能因找到LCA而提前标记）

lca = nullptr;

bool hasQ = dfs(root, q, p);

// 只有p和q都存在时，才返回找到的LCA

if (hasP && hasQ) {

// 重新调用一次以确定正确的LCA（因为前两次调用可能重置了lca）

dfs(root, p, q);

return lca;

} else {

return nullptr;

}

}

};

代码解释：

1、递归函数 dfs：

- 功能：检查当前子树是否包含p或q，并在满足条件时记录 LCA。

- 逻辑：

若当前节点是p或q（curr = true），则标记当前子树包含目标节点。

若左子树（`left`）和右子树（`right`）分别包含p和q，或当前节点是其中一个目标节点且子树包含另一个，则当前节点是 LCA，记录到lca。

返回left || right || curr，表示当前子树是否包含p或q。

2、主函数 lowestCommonAncestor：

- 首先通过两次dfs分别检查p和q是否存在于树中（hasP和hasQ）。

- 若两者都存在，再次调用dfs确定LCA并返回；否则返回null。

复杂度分析

- 时间复杂度：O(n)，其中n是二叉树的节点数。每个节点最多被访问 3 次（两次存在性检查 + 一次 LCA 确定），整体为线性时间。

- 空间复杂度：O(h)，其中h是树的高度。递归调用栈的深度取决于树的高度，最坏情况下（链状树）为O(n)，平均情况下（平衡树）为 O(log n)。

该方法通过后序遍历高效检查节点存在性并寻找 LCA，逻辑严谨，能够正确处理所有边界情况。