# 题目

给定一棵二叉树中的两个节点 p 和 q，返回它们的最近公共祖先节点（LCA）。

每个节点都包含其父节点的引用（指针）。Node 的定义如下：

class Node {

public int val;

public Node left;

public Node right;

public Node parent;

}

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:

形状, 圆圈

描述已自动生成

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 的最近公共祖先是 3。

示例 2:

形状, 圆圈

描述已自动生成

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 的最近公共祖先是 5，根据定义，一个节点可以是自身的最近公共祖先。

示例 3:

输入: root = [1,2], p = 1, q = 2

输出: 1

提示:

树中节点个数的范围是 [2, 105]。

-109 <= Node.val <= 109

所有的 Node.val 都是互不相同的。

p != q

p 和 q 存在于树中。

# 分析

要解决“已知父指针的二叉树中寻找两个节点的最近公共祖先（LCA）”的问题，核心优势是可通过父指针回溯节点的祖先路径。我们可以利用“路径相交”的思路，找到两条祖先路径的第一个公共节点，即为LCA。

思路分析：

1、核心观察：每个节点的父指针指向其直接祖先，因此从 p 和 q 分别向上回溯，可得到它们的完整祖先路径（包括自身）。

2、路径相交原理：

- 假设 p 到根的路径长度为 len\_p，q 到根的路径长度为 len\_q。

- 让较长路径的节点先向上回溯 |len\_p - len\_q| 步，使 p 和 q 处于“同一深度”（到根的距离相同）。

- 之后让 p 和 q 同时向上回溯，直到两者相遇——相遇的节点即为 LCA（因为此时它们的祖先完全一致，且是第一个公共节点）。

代码：

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* left;

Node\* right;

Node\* parent;

};

\*/

class Solution {

public:

Node\* lowestCommonAncestor(Node\* p, Node \* q) {

// 步骤1：计算p和q到根节点的路径长度（深度）

int len\_p = getDepth(p);

int len\_q = getDepth(q);

// 步骤2：让较长路径的节点先回溯，使p和q处于同一深度

while (len\_p > len\_q) {

p = p->parent;

len\_p--;

}

while (len\_q > len\_p) {

q = q->parent;

len\_q--;

}

// 步骤3：p和q同时回溯，直到相遇（相遇节点即为LCA）

while (p != q) {

p = p->parent;

q = q->parent;

}

return p;

}

private:

// 辅助函数：计算节点到根节点的路径长度（节点自身算1步）

int getDepth(Node\* node) {

int depth = 0;

while (node != nullptr) { // 根节点的parent为nullptr

depth++;

node = node->parent;

}

return depth;

}

};

代码解释：

1、计算路径长度（getDepth函数）：

- 从目标节点开始，通过parent指针向上遍历，每经过一个节点深度加1，直到遍历到根节点（parent为nullptr）。

- 例如：若p是叶子节点，根是其第3个祖先，则len\_p = 4（包含p自身和3个祖先）。

2、对齐节点深度：

- 若p的路径更长（len\_p > len\_q），则p向上回溯len\_p - len\_q步；反之则q回溯。

- 目的是让p和q到根的距离相同，确保后续同步回溯时能同时到达根节点。

3、同步回溯找LCA：

- 此时 p和 q 深度相同，同时向上回溯，每一步都检查是否相等。

- 由于 p 和 q 均存在于树中，它们的祖先路径必然在根节点相交，因此最终一定会相遇，相遇节点即为 LCA。

复杂度分析

- 时间复杂度：O(h)，其中 h 是二叉树的高度。

计算深度的过程最多遍历 h 个节点（从叶子到根）。

对齐深度和同步回溯的过程也最多遍历 h 个节点。

整体为线性时间，对于 h≤1e5（最坏链状树）完全可行。

- 空间复杂度：O(1)。仅使用几个变量存储深度和节点指针，无额外空间消耗。

关键优势

1、高效性：无需存储完整的祖先路径（如用哈希表记录路径），通过“对齐深度+同步回溯”直接找到LCA，空间复杂度最优。

2、直观性：利用路径相交的朴素逻辑，易于理解和实现，且避免了递归栈的开销。

该方法充分利用了父指针的特性，是解决“带父指针二叉树LCA”问题的最优方案之一。