0236. 二叉树的最近公共祖先

ITCharge 大约 3 分钟

- 标签：树、深度优先搜索、二叉树
- 难度：中等

题目链接

- 0236. 二叉树的最近公共祖先 - 力扣

题目大意

描述：给定一个二叉树的根节点 `root` ，以及二叉树中两个节点 `p` 和 `q` 。

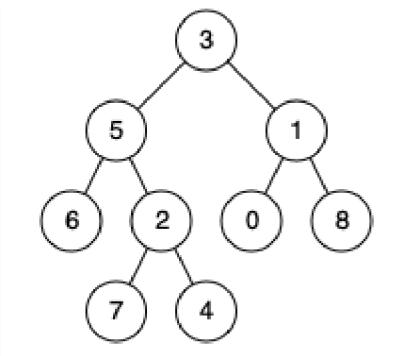
要求：找到该二叉树中指定节点 `p` 、 `q` 的最近公共祖先。

说明：

- **祖先**：如果节点 `p` 在节点 `node` 的左子树或右子树中，或者 `p == node` ，则称 `node` 是 `p` 的祖先。
- **最近公共祖先**：对于树的两个节点 `p` 、 `q` ，最近公共祖先表示为一个节点 `lca_node` ，满足 `lca_node` 是 `p` 、 `q` 的祖先且 `lca_node` 的深度尽可能大（一个节点也可以是自己的祖先）。
- 树中节点数目在范围 $[2, 10^5]$ 内。
- $-10^9 \leq Node.val \leq 10^9$。
- 所有 `Node.val` 互不相同。
- `p != q` 。
- `p` 和 `q` 均存在于给定的二叉树中。

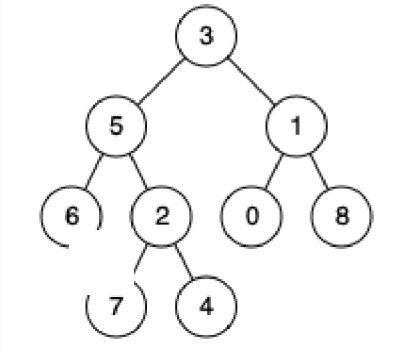
示例：

- 示例 1：




```py
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 和节点 1 的最近公共祖先是节点 3 。
```

- 示例 2：




```py
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 和节点 4 的最近公共祖先是节点 5 。因为根据定义最近公共祖先节点可以为节点本身。
```

解题思路

思路 1：递归遍历

设 `lca_node` 为节点 `p` 、 `q` 的最近公共祖先。则 `lca_node` 只能是下面几种情况：

1. `p` 、 `q` 在 `lca_node` 的子树中，且分别在 `lca_node` 的两侧子树中。
2. `p == lca_node` ，且 `q` 在 `lca_node` 的左子树或右子树中。

3. `q == lca_node` ，且 `p` 在 `lca_node` 的左子树或右子树中。

下面递归求解 `lca_node` 。递归需要满足以下条件：

- 如果 `p` 、 `q` 都不为空，则返回 `p` 、 `q` 的公共祖先。
- 如果 `p` 、 `q` 只有一个存在，则返回存在的一个。
- 如果 `p` 、 `q` 都不存在，则返回 `None` 。

具体思路为：

1. 如果当前节点 `node` 等于 `p` 或者 `q` ，那么 `node` 就是 `p` 、 `q` 的最近公共祖先，直接返回 `node` 。
2. 如果当前节点 `node` 不为 `None` ，则递归遍历左子树、右子树，并判断左右子树结果。
    1. 如果左右子树都不为空，则说明 `p` 、 `q` 在当前根节点的两侧，当前根节点就是他们的最近公共祖先。
    2. 如果左子树为空，则返回右子树。
    3. 如果右子树为空，则返回左子树。
    4. 如果左右子树都为空，则返回 `None` 。
3. 如果当前节点 `node` 为 `None` ，则说明 `p` 、 `q` 不在 `node` 的子树中，不可能为公共祖先，直接返回 `None` 。

思路 1：代码

```py
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        if root == p or root == q:
            return root

        if root:
            node_left = self.lowestCommonAncestor(root.left, p, q)
            node_right = self.lowestCommonAncestor(root.right, p, q)
            if node_left and node_right:
                return root
            elif not node_left:
                return node_right
            else:
                return node_left
        return None
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$。其中 n 是二叉树的节点数目。
- **空间复杂度**：$O(n)$。