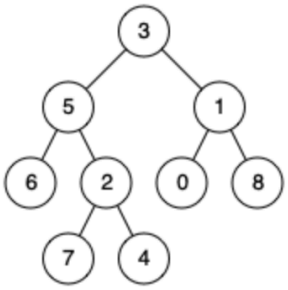
二叉树的公共祖先

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

例如，给定如下二叉树：root = [3,5,1,6,2,0,8,null,null,7,4]



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 和节点 1 的最近公共祖先是节点 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。
```

说明：

- 所有节点的值都是唯一的。
- p、q 为不同节点且均存在于给定的二叉树中。

1 递归法：

1 递归终结条件

当越过叶节点，则直接返回 null；

当 root等于 p, q，则直接返回 root；

2 处理当前层

3 下探到下一层

开启递归左子节点，返回值记为 left；

开启递归右子节点，返回值记为 right；

4 返回值：

根据left和right的返回值：

1 left， right为空：root的左右子树都不含p,q 返回 null

2 left， right都都为空：root的正好包含p，q，且p，q分别在左右子树，root就是公共祖先，返回 root

3 left为空，right不为空，说明p，q不在root的左子树，返回right：

p,q其中一个在 root的 右子树 中，此时 right 指向 p（假设为 p）；

p,q两节点都在 root 的 右子树 中，此时的 right 指向 最近公共祖先节点；

4 left不为空，right为空，说明p，q不在root的右子树，返回left：

#4 清理当前层

```
def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """

        # 递归终结条件：找到p或者q
         if not root or root == p or root == q:
             return root
         left = self.lowestCommonAncestor(root.left, p, q)
         right = self.lowestCommonAncestor(root.right, p ,q)
         if not left :
             return right
         if not right:
             return left
         return root
```

时间复杂度：O(N)，遍历节点造成的复杂度

空间复杂度：O(N)，是递归调用的复杂度，递归调用的栈深度取决于二叉树的高度，二叉树最坏情况下为一条链，此时高度为 N，因此空间复杂度为 $O(N)$。

```
    def lowestCommonAncestor(self, root, p, q):
        if root in (None, p, q): return root
        left, right = (self.lowestCommonAncestor(kid, p, q)
                       for kid in (root.left, root.right))
        return root if left and right else left or right
```

```
def lowestCommonAncestor(self, root, p, q):
    def path(root, goal):
        path, stack = [], [root]
        while True:
            node = stack.pop()
            if node:
                if node not in path[-1:]:
                    path += node,
                    if node == goal:
                        return path
                     stack += node, node.right, node.left
                else:
                    path.pop()
     return next(a for a, b in zip(path(root, p), path(root, q))[::
```