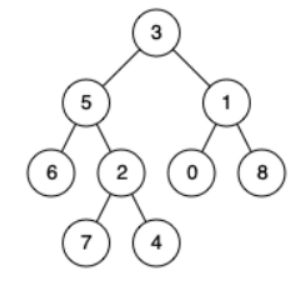
<https://leetcode.ca/all/1644.html>

Given the root of a binary tree, return *the lowest common ancestor (LCA) of two given nodes,* p *and* q. **If either node p or q does not exist in the tree**, return null. All values of the nodes in the tree are unique.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): "The lowest common ancestor of two nodes p and q in a binary tree T is the lowest node that has both p and q as descendants (where we allow **a node to be a descendant of itself**)". A descendant of a node x is a node y that is on the path from node x to some leaf node.

Example 1:

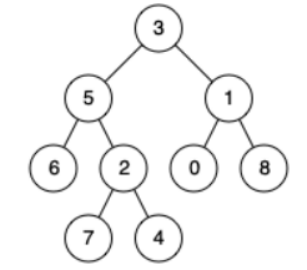


Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

Output: 3

Explanation: The LCA of nodes 5 and 1 is 3.

Example 2:

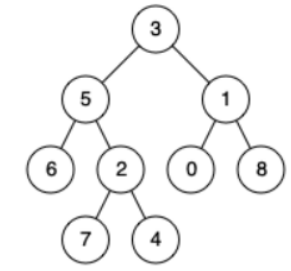


Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

Output: 5

Explanation: The LCA of nodes 5 and 4 is 5. A node can be a descendant of itself according to the definition of LCA.

Example 3:



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

Output: null

Explanation: Node 10 does not exist in the tree, so return null.

Constraints:

* The number of nodes in the tree is in the range [1, 104].
* -109 <= Node.val <= 109
* All Node.val are unique.
* p != q

Follow up: Can you find the LCA traversing the tree, without checking nodes existence?

**Note: The difference between L1644.Lowest Common Ancestor of a Binary Tree II and L236.Lowest Common Ancestor of a Binary Tree is L1644 allow p and q may not exist in the tree.**

**Attempt 1: 2022-12-28**

**Solution 1:  Divide and Conquer (30 min)**

class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

one.left = two;

one.right = five;

two.left = three;

two.right = four;

five.right = six;

TreeNode lca = s.lowestCommonAncestor(one, three, seven);

System.out.println(lca);

}

private int count = 0;

private TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

TreeNode result = helper(root, p, q);

// Only when two nodes both exist will return their LCA, otherwise is LCA is NULL

if(count == 2) {

return result;

} else {

return null;

}

}

public TreeNode helper(TreeNode root, TreeNode p, TreeNode q) {

if(root == null) {

return null;

}

// Differ than L236.Lowest Common Ancestor of a Binary Tree

// we have to actually check if the TreeNode p or q exist or not in the tree,

// if exist then add count

if(root == p || root == q) {

count++;

return root;

}

TreeNode left = helper(root.left, p, q);

TreeNode right = helper(root.right, p, q);

if(left != null && right != null) {

return root;

}

if(left != null) {

return left;

} else {

return right;

}

}

}

Complexity Analysis

Time Complexity: O(N). Where N is the number of nodes in the binary tree. In the worst case we might be visiting all the nodes of the binary tree.

Space Complexity: O(N). This is because the maximum amount of space utilized by the recursion stack would be N since the height of a skewed binary tree could be N.