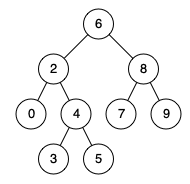
<https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/>

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): “The lowest common ancestor is defined between two nodes p and q as the lowest node in

T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

**Example 1:**

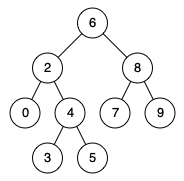


Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

Output: 6

Explanation: The LCA of nodes 2 and 8 is 6.

**Example 2:**



Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

Output: 2

Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

**Example 3:**

Input: root = [2,1], p = 2, q = 1

Output: 2

**Constraints:**

The number of nodes in the tree is in the range [2, 10^5].

-10^9 <= Node.val <= 10^9

All Node.val are **unique**.

p != q

p and q will exist in the BST.

**Attempt 1: 2022-12-03**

**Solution 1:  Divide and Conquer (5 min, same as L236.Lowest Common Ancestor of a Binary Tree, traditional way)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\*    int val;

\*    TreeNode left;

\*    TreeNode right;

\*    TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

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

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

            return root;

        }

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

        TreeNode right = lowestCommonAncestor(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.

**Solution 2:  Traversal use BST nature**

**Style 1: DFS (10 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\*    int val;

\*    TreeNode left;

\*    TreeNode right;

\*    TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

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

        if(root == null) {

            return null;

        }

        // If both n1 and n2 are smaller than root, then LCA lies in left

        if(root.val > p.val && root.val > q.val) {

            return lowestCommonAncestor(root.left, p, q);

        }

        // If both n1 and n2 are bigger than root, then LCA lies in right

        if(root.val < p.val && root.val < q.val) {

            return lowestCommonAncestor(root.right, p, q);

        }

        // Directly return current node as it satisfy the only situation left

        // root.val >= p.val && root.val <= q.val

        return root;

    }

}

**Style 2: DFS (10 min)**

class Solution {

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

        while(true) {

            if(root.val > p.val && root.val > q.val) {

                root = root.left;

            } else if(root.val < p.val && root.val < q.val) {

                root = root.right;

            } else {

                return root;

            }

        }

    }

}

**Refer to**

**对于二叉搜索树，公共祖先的值一定大于等于较小的节点，小于等于较大的节点。换言之，在遍历树的时候，如果当前结点大于两个节点，则结果在当前结点的左子树里，如果当前结点小于两个节点，则结果在当前节点的右子树里。最后一种情况就是当前节点大于或等于较小节点，小于或等于较大节点，直接返回当前节点即可。**

If we are given a BST where every node has parent pointer, then LCA can be easily determined by traversing up using parent pointer and printing the first intersecting node. We can solve this problem using BST properties. We can recursively traverse the BST from root. The main idea of the solution is, while traversing from top to bottom, the first node n we encounter with value between n1 and n2, i.e., n1 < n < n2 or same as one of the n1 or n2, is LCA of n1 and n2 (assuming that n1 < n2). So just recursively traverse the BST in, if node’s value is greater than both n1 and n2 then our LCA lies in left side of the node, if it’s is smaller than both n1 and n2, then LCA lies on right side. Otherwise root is LCA (assuming that both n1 and n2 are present in BST)

**Refer to**

[L236.Lowest Common Ancestor of a Binary Tree (Ref.L865,L235)](note://E191ABBC6A9B4A3C989AF0136CABCFA4)