LeetCode 236 — Lowest Common Ancestor of a Binary Tree

🔗 https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

1. Problem Title & Link

Title: LeetCode 236: Lowest Common Ancestor of a Binary Tree
Link: https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

2. Problem Statement (Short Summary)

Given the root of a binary tree and two nodes p and q, return their **Lowest Common Ancestor (LCA)**.
LCA = the **lowest (deepest)** node in the tree that has **both p and q as descendants**
(A node can be a descendant of itself.)

3. Examples (Input → Output)

Example 1

Tree:

```
     3
    / \
   5   1
  / \ / \
 6  2 0  8
   / \
  7   4
```

Input: p = 5, q = 1
Output: 3

Example 2

Input: p = 5, q = 4
Output: 5
Because a node can be the ancestor of itself.

Example 3

Tree:

```
 2
/ \
1  3
```

Input: p=1, q=3
Output: 2

4. Constraints

- Node count ≤ 10^5
- Values may repeat (important) — identity matters
- Binary tree (NOT BST)
- Guaranteed that both p and q exist in the tree

5. Core Concept (Pattern / Topic)

⭐ **DFS + Postorder Traversal (Tree Recursion)**

Also belongs to:

- Tree recursion
- LCA patterns
- Divide & Conquer

6. Thought Process (Step-by-Step Explanation)

Goal

Find the point where:

- One node is in the left subtree
- The other is in the right subtree
  → That node is the LCA.

Key Recursive Logic

For each node return:

- p if found
- q if found
- non-null from either subtree → possible LCA

The three cases to identify LCA:

1. **Left returns a node & right returns a node**
   → current node is LCA
2. **Current node itself is p or q**
   → return current node
3. **Only one subtree has p or q**
   → propagate upward

Intuition (Divide & Conquer)

- Ask left subtree → did you find p or q?

- Ask right subtree → same
- If both sides return something → THIS node = LCA
- Else return whichever side has a node

7. Visual / Intuition Diagram

Example:

```
   3
  / \
 5   1
```

Searching for LCA(5,1):

Left subtree returns 5

Right subtree returns 1

Current node = 3 → LCA

Another case:

```
 5
  \
   4
```

Searching for LCA(5,4):

Right returns 4

Current is 5 (matches p)

→ LCA = 5

8. Pseudocode

```
function LCA(root, p, q):
    if root is null:
        return null
    if root == p or root == q:
        return root

    left = LCA(root.left, p, q)
    right = LCA(root.right, p, q)

    if left != null and right != null:
        return root

    return left if left != null else right
```

9. Code Implementation

✅ **Python**

```
class Solution:
    def lowestCommonAncestor(self, root, p, 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 left and right:
            return root
        return left if left else right
```

✅ **Java**

```
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;

        return (left != null) ? left : right;
    }
}
```

10. Time & Space Complexity

Time: O(n)

Because you visit each node once.

Space: O(h) recursion stack

- h = height of tree
- Worst case (skewed): O(n)
- Best case (balanced): O(log n)

11. Common Mistakes / Edge Cases

❌ Treating tree as BST (this is general BT)

❌ Returning value instead of node reference

❌ Thinking LCA always exists above both nodes

❌ Forgetting that node can be ancestor of itself

Edge cases:

✔ p = q

✔ p is ancestor of q

✔ q is ancestor of p

✔ Tree is skewed

12. Detailed Dry Run (Step-by-Step Table)

Tree:

```
    3
   / \
  5   1
```

p=5, q=1

Node	Left Result	Right Result	Return
5	p found	-	5
1	q found	-	1
3	left=5	right=1	3

Final LCA = **3**

13. Common Use Cases

- Finding common ancestor in directory structures
- Resolve merge conflicts in trees
- Used in compilers → AST analysis
- Network routing in hierarchical structures

14. Common Traps

⚠ Using BST logic (p < root < q) — WRONG

⚠ Returning boolean instead of node

⚠ Forgetting base case if root == p or root == q

⚠ Overwriting answers from recursion incorrectly

15. Builds To (Related LeetCode Problems)

- LC 235 — LCA in BST
- LC 1123 — LCA of deepest leaves

- LC 865 — Smallest subtree with all deepest nodes
- LC 543 — Diameter of binary tree
- LC 124 — Max path sum

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
DFS recursion	O(n)	O(h)	Standard & best
DFS iterative + parent map	O(n)	O(n)	When parent access needed
Binary Lifting	O(n log n)	O(n log n)	For multiple repeated LCA queries

17. Why This Solution Works (Short Intuition)

Each subtree reports whether it found p or q.
The first node where **both sides report something** is the **lowest point where p and q meet** → LCA.

18. Variations / Follow-Up Questions

- Find LCA when nodes might NOT exist
- Find LCA for **k nodes**
- LCA in **binary search tree** (easier)
- Return **distance** between p and q
- Preprocess tree for **O(1)** LCA queries