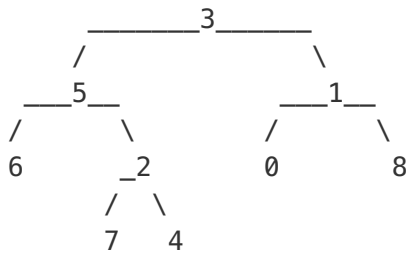
Lowest Common Ancestor of a Binary Tree

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

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow **a node to be a descendant of itself**).”



For example, the lowest common ancestor (LCA) of nodes `5` and `1` is `3`. Another example is LCA of nodes `5` and `4` is `5`, since a node can be a descendant of itself according to the LCA definition.

Solution 1

Same solution in several languages. It's recursive and expands the meaning of the function. If the current (sub)tree contains both p and q, then the function result is their LCA. If only one of them is in that subtree, then the result is that one of them. If neither are in that subtree, the result is null/None/nil.

Update: I also wrote two iterative solutions now, one of them being a version of the solution here. They're more complicated than this simple recursive solution, but I do find them interesting.

C++

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if (!root || root == p || root == q) return root;
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);
    return !left ? right : !right ? left : root;
}
```

Python

```
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
```

Or using that `None` is considered smaller than any node:

```
def lowestCommonAncestor(self, root, p, q):
    if root in (None, p, q): return root
    subs = [self.lowestCommonAncestor(kid, p, q)
            for kid in (root.left, root.right)]
    return root if all(subs) else max(subs)
```

Ruby

```
def lowest_common_ancestor(root, p, q)
    return root if [nil, p, q].index root
    left = lowest_common_ancestor(root.left, p, q)
    right = lowest_common_ancestor(root.right, p, q)
    left && right ? root : left || right
end
```

Java

```java
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);
    return left == null ? right : right == null ? left : root;
}
```

written by StefanPochmann original link here

Solution 2

```
public 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;
    }
}
```

written by yuhangjiang original link here

Solution 3

```
class Solution {
public:
    TreeNode *lowestCommonAncestor(TreeNode *root, TreeNode *p, TreeNode *q) {
        if (root == p || root == q || root == NULL) return root;
        TreeNode *left = lowestCommonAncestor(root->left, p, q), *right = lowestC
ommonAncestor(root->right, p, q);
        return left && right ? root : left ? left : right;
    }
};
```

written by prime_tang original link here