Previous Next

236. Lowest Common Ancestor of a Binary Tree

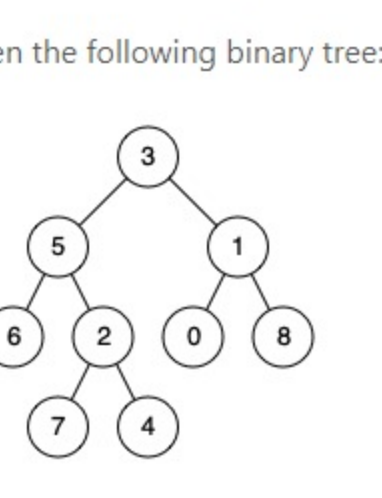
Nov. 15, 2018 | 215.2K views

Average Rating: 4.09 (209 votes)

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 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**)."

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]



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:

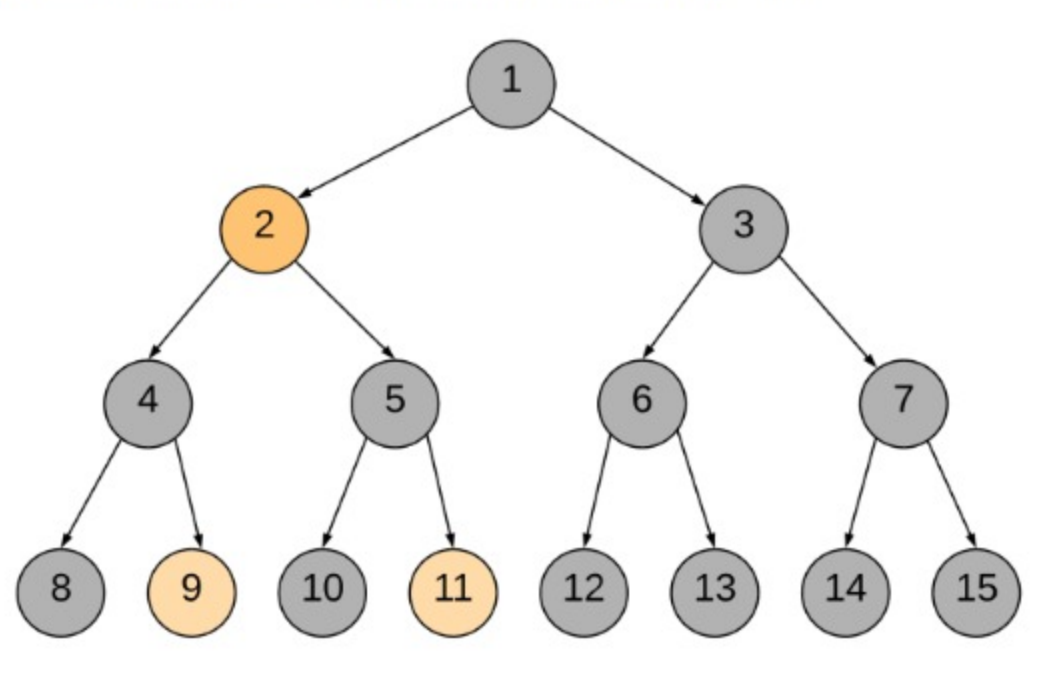
```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4
 is 5, since a node can be a descendant of itself according to the LCA definition.
```

Note:

- All of the nodes' values will be unique.
- p and q are different and both values will exist in the binary tree.

Solution

First the given nodes `p` and `q` are to be searched in a binary tree and then their lowest common ancestor is to be found. We can resort to a normal tree traversal to search for the two nodes. Once we reach the desired nodes `p` and `q`, we can backtrack and find the lowest common ancestor.



Lowest Common Ancestor for **Node 9** and **Node 11** is **Node 2**

Approach 1: Recursive Approach

Intuition

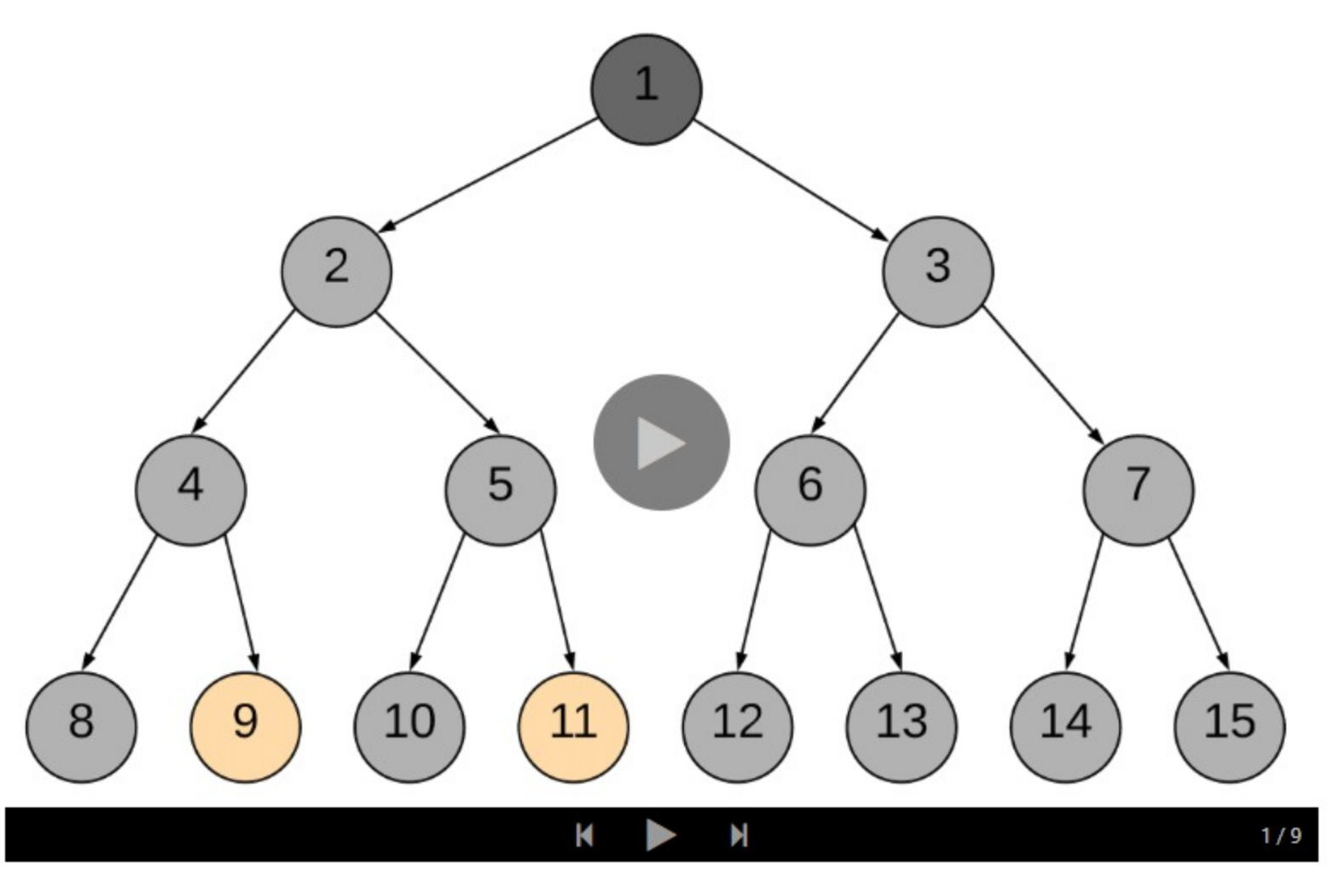
The approach is pretty intuitive. Traverse the tree in a depth first manner. The moment you encounter either of the nodes `p` or `q`, return some boolean flag. The flag helps to determine if we found the required nodes in any of the paths. The least common ancestor would then be the node for which both the subtree recursions return a `True` flag. It can also be the node which itself is one of `p` or `q` and for which one of the subtree recursions returns a `True` flag.

Let us look at the formal algorithm based on this idea.

Algorithm

1. Start traversing the tree from the root node.
2. If the current node itself is one of `p` or `q`, we would mark a variable `mid` as `True` and continue the search for the other node in the left and right branches.
3. If either of the left or the right branch returns `True`, this means one of the two nodes was found below.
4. If at any point in the traversal, any two of the three flags `left`, `right` or `mid` become `True`, this means we have found the lowest common ancestor for the nodes `p` and `q`.

Let us look at a sample tree and we search for the lowest common ancestor of two nodes `9` and `11` in the tree.



Following is the sequence of nodes that are followed in the recursion:

```
1 --> 2 --> 4 --> 8
BACKTRACK 8 --> 4
4 --> 9 (ONE NODE FOUND, return True)
BACKTRACK 9 --> 4 --> 2
2 --> 5 --> 10
BACKTRACK 10 --> 5
5 --> 11 (ANOTHER NODE FOUND, return True)
BACKTRACK 11 --> 5 --> 2

2 is the node where we have left = True and right = True and hence it is the lowest co
```

Java | Python | Copy

```
class Solution:

    def __init__(self):
        # Variable to store LCA node.
        self.ans = None

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

            # If reached the end of a branch, return False.
            if not current_node:
                return False

            # Left Recursion
            left = recurse_tree(current_node.left)

            # Right Recursion
            right = recurse_tree(current_node.right)

            # If the current node is one of p or q
            mid = current_node == p or current_node == q
```

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.

Approach 2: Iterative using parent pointers

Intuition

If we have parent pointers for each node we can traverse back from `p` and `q` to get their ancestors. The first common node we get during this traversal would be the LCA node. We can save the parent pointers in a dictionary as we traverse the tree.

Algorithm

1. Start from the root node and traverse the tree.
2. Until we find `p` and `q` both, keep storing the parent pointers in a dictionary.
3. Once we have found both `p` and `q`, we get all the ancestors for `p` using the parent dictionary and add to a set called `ancestors`.
4. Similarly, we traverse through ancestors for node `q`. If the ancestor is present in the ancestors set for `p`, this means this is the first ancestor common between `p` and `q` (while traversing upwards) and hence this is the LCA node.

Java | Python | Copy

```
class Solution:

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

        # Stack for tree traversal
        stack = [root]

        # Dictionary for parent pointers
        parent = {root: None}

        # Iterate until we find both the nodes p and q
        while p not in parent or q not in parent:

            node = stack.pop()

            # While traversing the tree, keep saving the parent pointers.
            if node.left:
                parent[node.left] = node
                stack.append(node.left)
            if node.right:
                parent[node.right] = node
```

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 : $O(N)$. In the worst case space utilized by the stack, the parent pointer dictionary and the ancestor set, would be N each, since the height of a skewed binary tree could be N.

Approach 3: Iterative without parent pointers

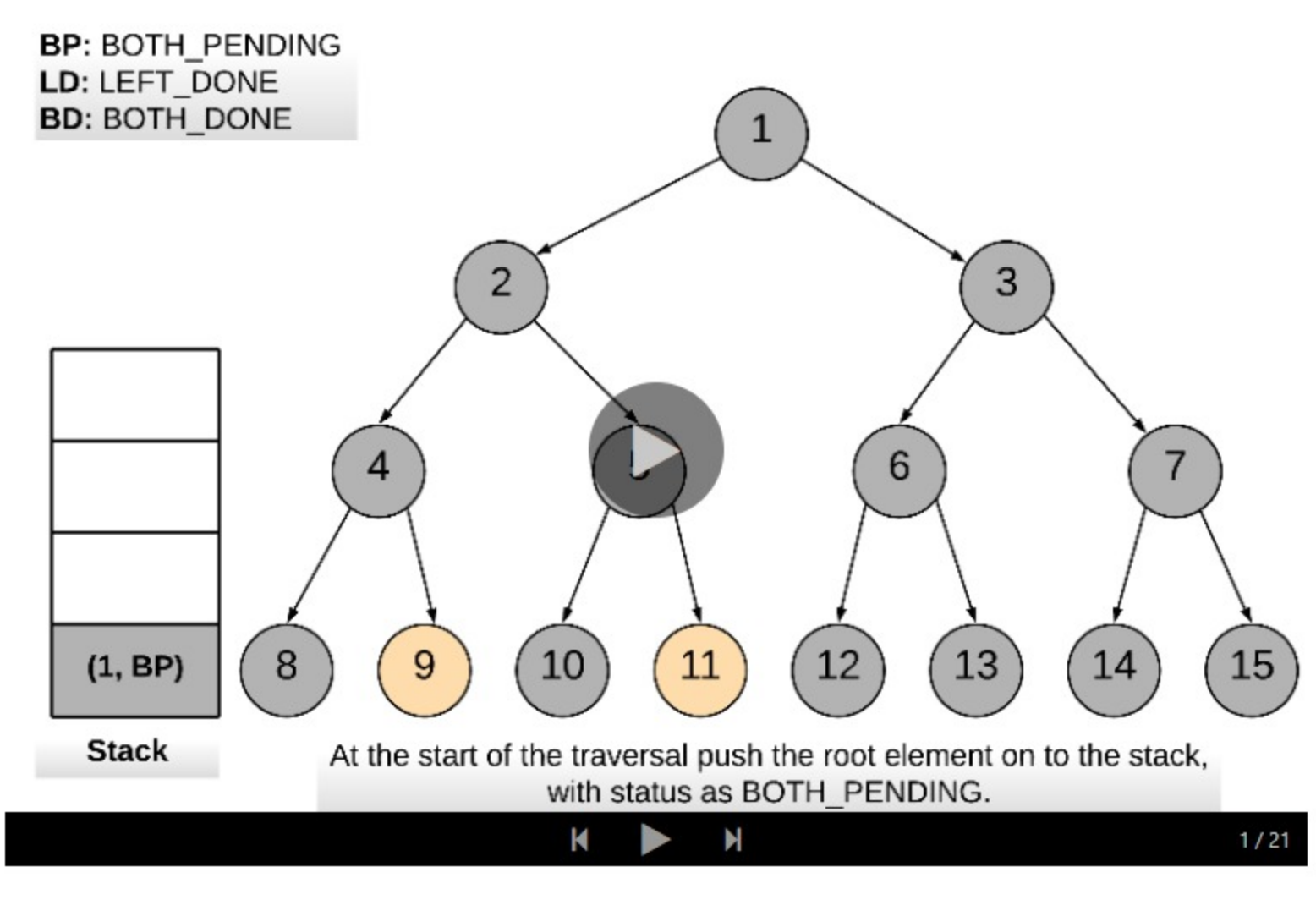
Intuition

In the previous approach, we come across the LCA during the backtracking process. We can get rid of the backtracking process itself. In this approach we always have a pointer to the probable LCA and the moment we find both the nodes we return the pointer as the answer.

Algorithm

1. Start with root node.
2. Put the `(root, root_state)` on to the stack. `root_state` defines whether one of the children or both children of `root` are left for traversal.
3. While the stack is not empty, peek into the top element of the stack represented as `(parent_node, parent_state)`.
4. Before traversing any of the child nodes of `parent_node` we check if the `parent_node` itself is one of `p` or `q`.
5. First time we find either of `p` or `q`, set a boolean flag called `one_node_found` to `True`. Also start keeping track of the lowest common ancestors by keeping a note of the top index of the stack in the variable `LCA_index`. Since all the current elements of the stack are ancestors of the node we just found.
6. The second time `parent_node == p or parent_node == q` it means we have found both the nodes and we can return the `LCA node`.
7. Whenever we visit a child of a `parent_node` we push the `(parent_node, updated_parent_state)` onto the stack. We update the state of the parent since a child/branch has been visited/processed and accordingly the state changes.
8. A node finally gets popped off from the stack when the state becomes `BOTH_DONE` implying both left and right subtrees have been pushed onto the stack and processed. If `one_node_found` is `True` then we need to check if the top node being popped could be one of the ancestors of the found node. In that case we need to reduce `LCA_index` by one. Since one of the ancestors was popped off.

> Whenever both `p` and `q` are found, `LCA_index` would be pointing to an index in the stack which would contain all the common ancestors between `p` and `q`. And the `LCA_index` element has the `lowest` ancestor common between p and q.



The animation above shows how a stack is used to traverse the binary tree and keep track of the common ancestors between nodes `p` and `q`.

Java | Python | Copy

```
class Solution:

    # Three static flags to keep track of post-order traversal.

    # Both left and right traversal pending for a node.
    # Indicates the nodes children are yet to be traversed.
    BOTH_PENDING = 2
    # Left traversal done.
    LEFT_DONE = 1
    # Both left and right traversal done for a node.
    # Indicates the node can be popped off the stack.
    BOTH_DONE = 0

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

        # Initialize the stack with the root node.
        stack = [(root, Solution.BOTH_PENDING)]

        # This flag is set when either one of p or q is found.
        one_node_found = False
```

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. The advantage of this approach is that we can prune backtracking. We simply return once both the nodes are found.
- Space Complexity : $O(N)$. In the worst case the space utilized by stack would be N since the height of a skewed binary tree could be N.

Rate this article: ★★★★★

Previous Next

Comments: 53 Sort By

Type comment here... (Markdown is supported)

Preview Post

godayaldivya STAFF ★ 231 November 18, 2018 3:03 AM

Hi Guys,
I have been monitoring the response the article is receiving in the form of ratings and I see a downward trend. As an author, I feel it is my responsibility to make sure the article is easily understandable and has all the optimal solutions. I would love to have some sort of feedback from you, the readers as to what can be improved in the article and what problems do you see currently. Thank you !

Read More

95 Share Reply

SHOW 22 REPLIES

jinil ★ 37 June 24, 2019 11:43 AM

```
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        if root is None:
            return None
```

Read More

20 Share Reply

SHOW 1 REPLY

NideeshT ★ 589 July 16, 2019 6:23 AM

Java Code + Whiteboard Youtube Video Explanation accepted -https://www.youtube.com/watch?v=uKhLoNaG9LI (clickable link)

Lowest Common Ancestor...

Read More

28 Share Reply

SHOW 4 REPLIES

yingjielin ★ 29 March 22, 2019 4:09 AM

I think the time complexity analysis for Solution 1 is not accurate. Since the recursiveTree function only returns a boolean, there is no way to tell whether both p and q are found in any subtree. Therefore, even if both p and q are found in the left subtree of a node, full efforts are still made to check the right subtree. Therefore, the time complexity is still O(N), but this is not the worst case, but every case.

18 Share Reply

SHOW 3 REPLIES

jinil ★ 37 June 24, 2019 11:41 AM Report

```
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        if root is None:
            return None
```

Read More

15 Share Reply

softwareshortcut ★ 436 May 2, 2019 6:18 AM

1st solution is pretty well thought. It looks so simple but might be hard to come up with during an interview in only 10-15 minutes unless you've seen it before. Using the addition of *mid+left+right* groups a bunch of test cases together. My solution had the same approach but a lot more test cases.

9 Share Reply

SHOW 2 REPLIES

madantv ★ 7 July 23, 2019 9:54 AM

/**

- Definition for a binary tree node.
- public class TreeNode {

Read More

7 Share Reply

SHOW 2 REPLIES

blueglaucus ★ 3 January 15, 2020 3:48 PM

Is it possible to do solution 1 without using an instance variable to store the LCA node?

1 Share Reply

SHOW 1 REPLY

professor19 ★ 147 June 20, 2019 12:45 PM Report

I think the solution 2 approach when we add element in set it is log(n) rather than constant, so complexity is O(N logN)

1 Share Reply

SHOW 2 REPLIES

kansalhk06 ★ 18 July 2, 2019 12:17 PM

Just an edge case:
Solution 1 doesn't work if p and q both are same nodes, it will return NULL but is expected to return the same node.

0 Share Reply

SHOW 3 REPLIES