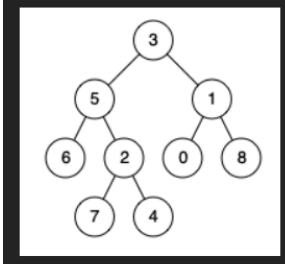
236. Lowest Common Ancestor of a Binary Tree

1. Problem Statement:

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

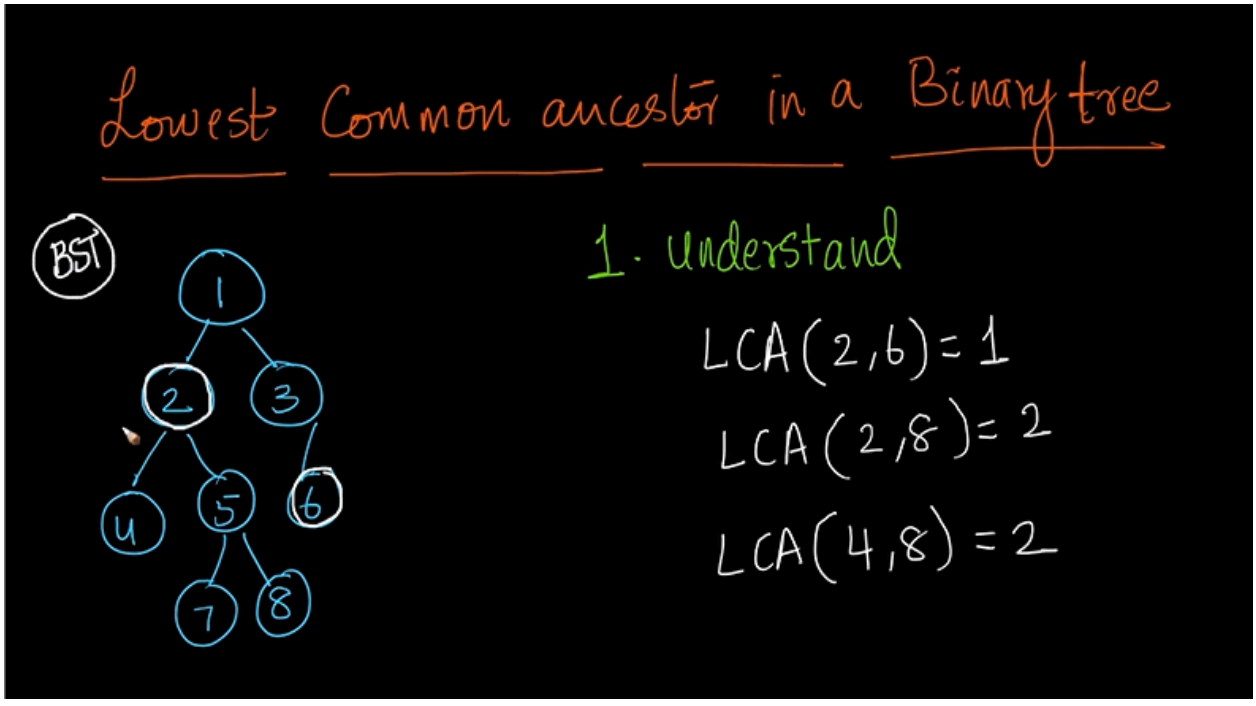
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:



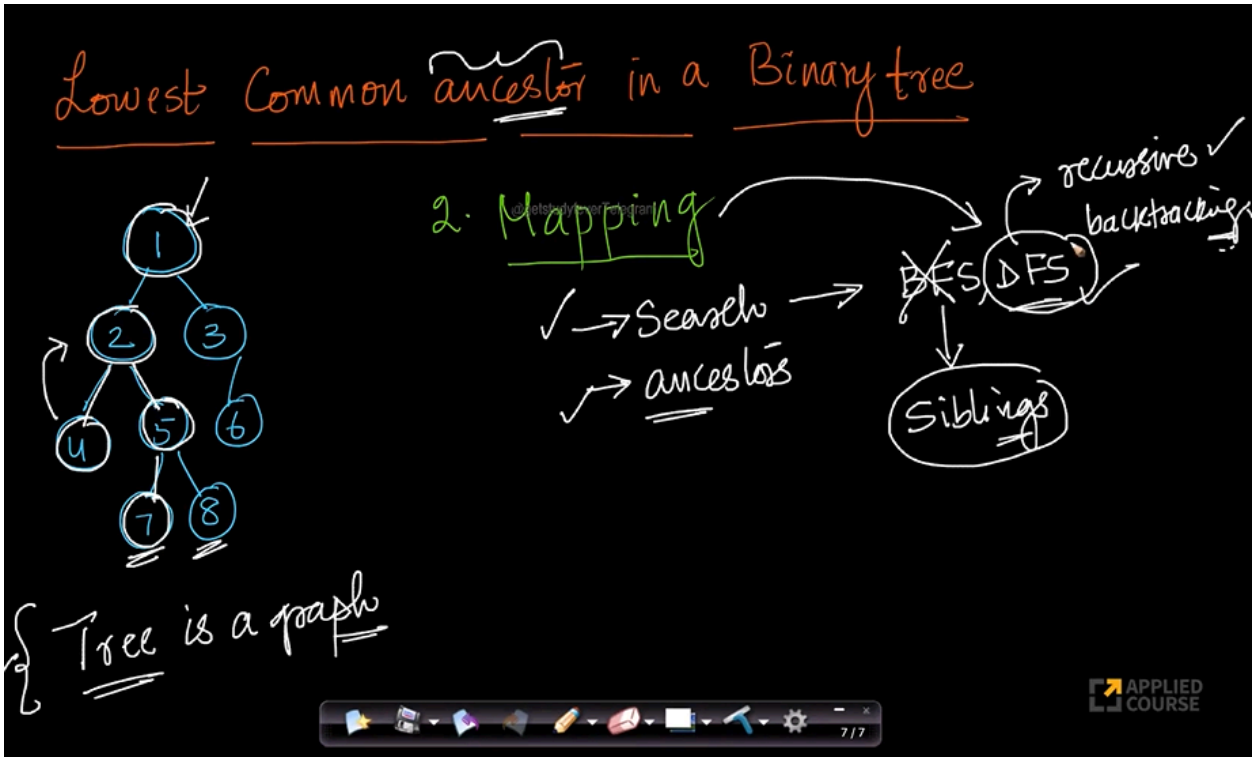

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

2. First lets understand the problem statement

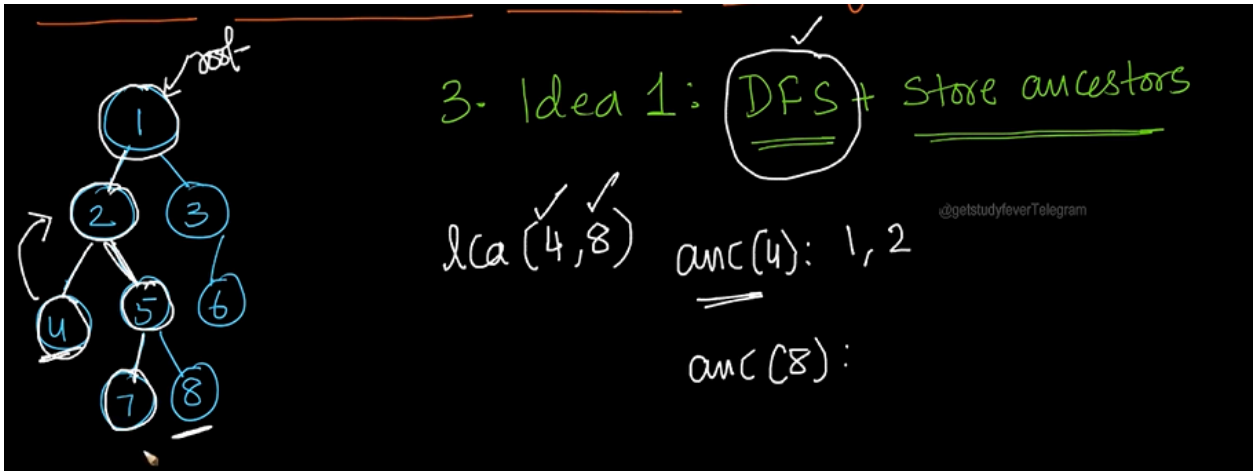


Approach:

1. First we have to apply some mapping to the solution
2. First we have search the given nodes…and then find the ancestor
3. So for searching …we can use BFS or DFS…BFS helps us in finding the siblings of a given node…
4. But DFS is applied using recursion and it backtracks to previous elements…which helps us in finding the ancestors…so we'll go ahead with DFS

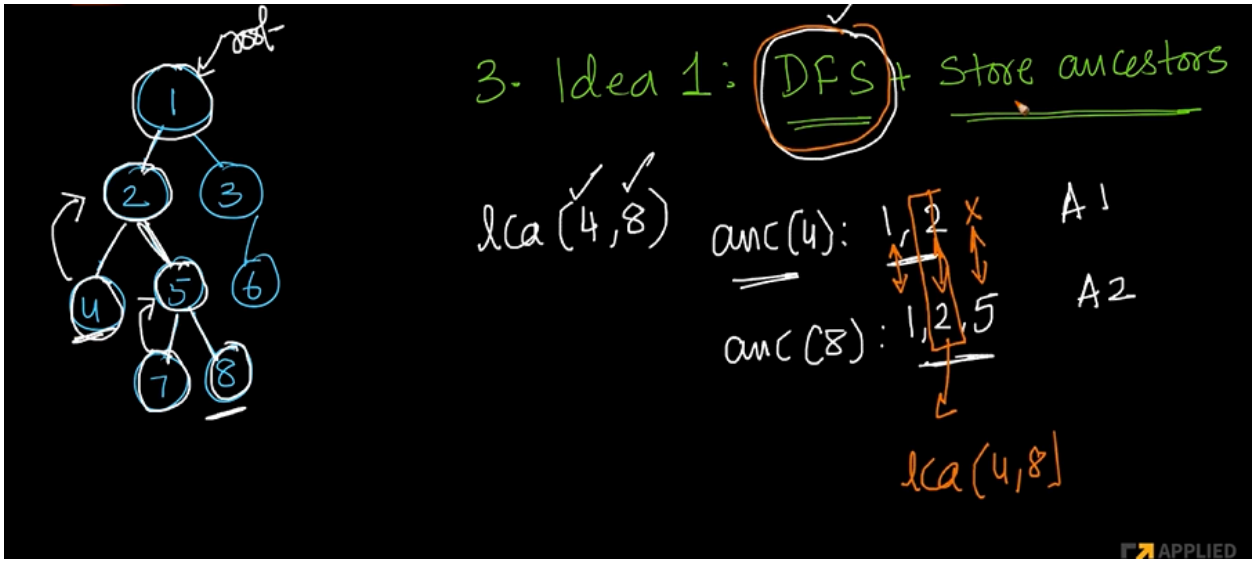


5. So we'll have build DFS like algorithm to achieve this
6. Now here what we do is…using DFS first we find node 4 and store its path

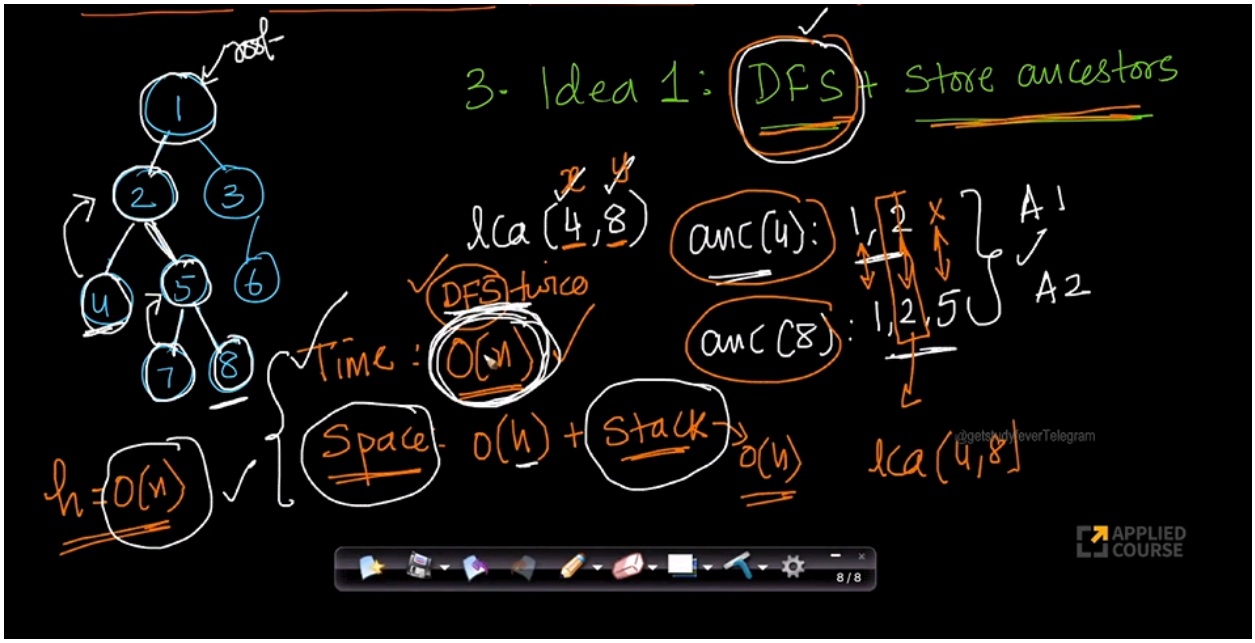


And in next iteration we'll search for 8 and store its path

7. Now LCA(4,8) is 2



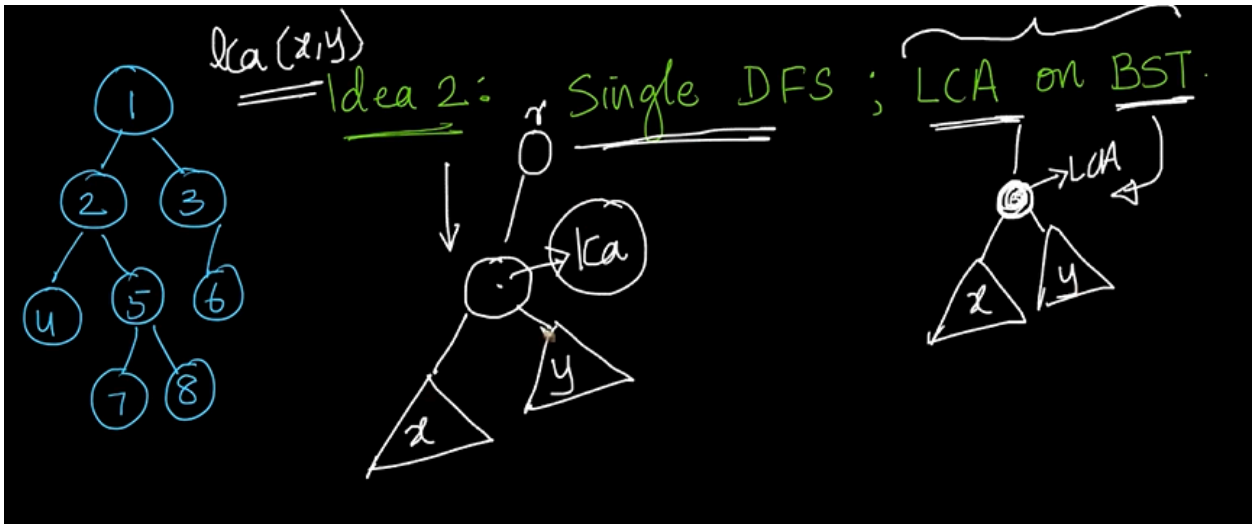
8. Here time complexity would be O(n) and space complexity would be O(h) + stack

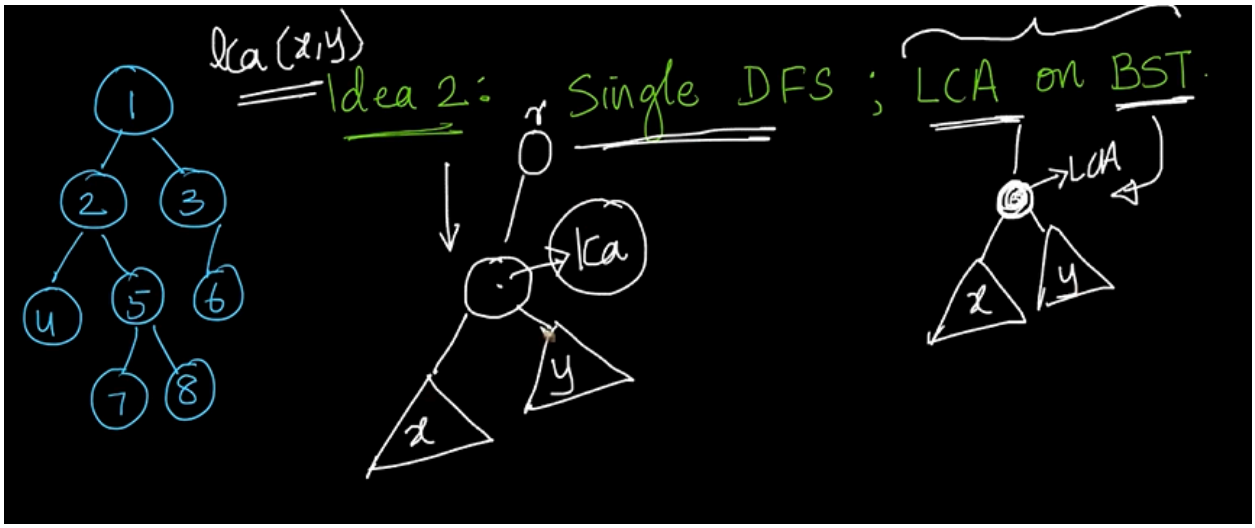


9. Now can we improve upon time or space complex?

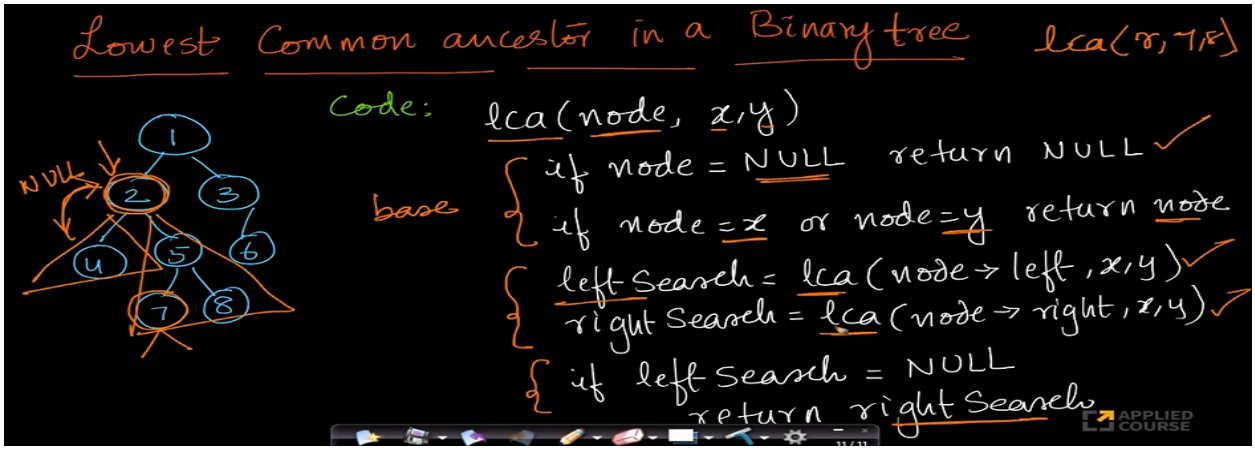
Approach 2

1. Here Approach 2 would be only to use single DFS search and find LCA

2. 

   here the top elements would be our LCA(idea of BST)...let try this core idea
3. Lets consider LCA(7,8) as example
4. Here first we search for 7 in our tree…and when we found the 7..it signals its parent node(5) that it is present here…and similarly when we found the node 8…it signals its parent node(5) that it is here…so when a node receives two signals that it is the LCA
   

5. Pseudocode:



Python code:

1.
```python
class Solution:
  def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':


    if not root or root == p or root == q:
        return root

    l = self.lowestCommonAncestor(root.left,p,q)
    r = self.lowestCommonAncestor(root.right,p,q)

    if l and r:
        return root
    return l or r
```

2.

Approach

1. The function lowestCommonAncestor takes in three parameters: the root of a binary tree (root) and two nodes of the binary tree (p and q).
2. The first if statement checks if the root is None or if it is equal to either p or q. If either of these conditions is true, it means that we have found one of the nodes we are looking for, and we return the root.
3. Next, we recursively call the lowestCommonAncestor function on the left and right subtrees of the root, passing in the same nodes p and q. We store the results of these recursive calls in variables l and r, respectively.
4. The second if statement checks if both l and r are not None. If this condition is true, it means that we have found both p and q in different subtrees of the current root, and therefore the current root is the lowest common ancestor. We return the current root.
5. If the second if statement is not satisfied, we return either l or r, depending on which one is not None. This is because if only one of l and r is not None, it means that the other node is not in the subtree of the current root, so we return the node that is in the subtree.
6. If none of the previous conditions is satisfied, it means that both l and r are None, so we return None. This happens when we have reached the end of a branch of the binary tree without finding either p or q.