Lowest Common Ancestor

235. Lowest Common Ancestor of a Binary Search Tree
236. Lowest Common Ancestor of a Binary Tree
1644. Lowest Common Ancestor of a Binary Tree II
1650. Lowest Common Ancestor of a Binary Tree III
1676. Lowest Common Ancestor of a Binary Tree IV
1740. Find Distance in a Binary Tree

Yifan Fei
Feb 12nd 2021

LCA definition

In graph theory and computer science, the **lowest common ancestor** (**LCA**) of two nodes v and w in a tree or directed acyclic graph (DAG) T is the lowest (i.e. deepest) node that has both v and w as descendants, where we define each node to be a descendant of itself (so if v has a direct connection from w, w is the lowest common ancestor).

235

```python
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        # soln 1, recursion
        if p.val < root.val and q.val < root.val:
            return self.lowestCommonAncestor(root.left, p, q)
        elif p.val > root.val and q.val > root.val:
            return self.lowestCommonAncestor(root.right, p, q)
        else:
            return root

        # soln 2
        while root:
            if root.val < p.val and root.val < q.val:
                # on right side of root
                root = root.right
            elif root.val > p.val and root.val > q.val:
                # on left side of root
                root = root.left
            else:
                # root in middle
                return root
```

236

```python
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        if not root:
            return None
        if root in (p, q):
            return root

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

        if left and right:
            return root
        if left or right:
            return right or left
```

1644

```python
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        self.flagp = False
        self.flagq = False
        res = self.helper(root, p, q)
        if not self.flagp or not self.flagq:
            return None
        return res

    def helper(self, root, p, q):
        if not root:
            return None
        left, right = self.helper(root.left, p, q), self.helper(root.right, p, q)
        if root == p:
            self.flagp = True
            return root
        if root == q:
            self.flagq = True
            return root
        if left and right:
            return root
        if left or right:
            return right or left
```

1650

```python
class Solution:
    def lowestCommonAncestor(self, p: 'Node', q: 'Node') -> 'Node':
        # soln 0, bottom up, 2 pointers
        p1, p2 = p, q
        while p1 != p2:
            # print(p1.val, p2.val)
            # when p1 points to root (i.e p1.parent is None), assign q to p1
            p1 = p1.parent if p1.parent else q
            p2 = p2.parent if p2.parent else p

        return p1

        # soln 1
        visited = set()
        while q:
            visited.add(q.val)
            q = q.parent

        while p:
            if p.val in visited: return p
            visited.add(p.val)
            p = p.parent
        return None
```

1676

```python
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', nodes: 'List[TreeNode]') -> 'TreeNode':
        nodes = set(nodes)

        def LCA(root):
            if not root:
                return None
            if root in nodes:
                return root
            l, r = LCA(root.left), LCA(root.right)
            if l and r:
                return root
            if l or r:
                return r or l

        return LCA(root)
```

1740

```python
class Solution:
    def findDistance(self, root: TreeNode, p: int, q: int) -> int:
        def LCA(root, p, q):
            if not root: return
            if p == root.val or q == root.val:
                return root

            l, r = LCA(root.left, p, q), LCA(root.right, p, q)
            if l and r:
                return root
            if l or r:
                return r or l

        def dist(node, target):
            if not node:
                return float('inf')
            if node.val == target:
                return 0
            return 1 + min(dist(node.left, target), dist(node.right, target))

        lca = LCA(root, p, q)
        return dist(lca, p) + dist(lca, q)
```