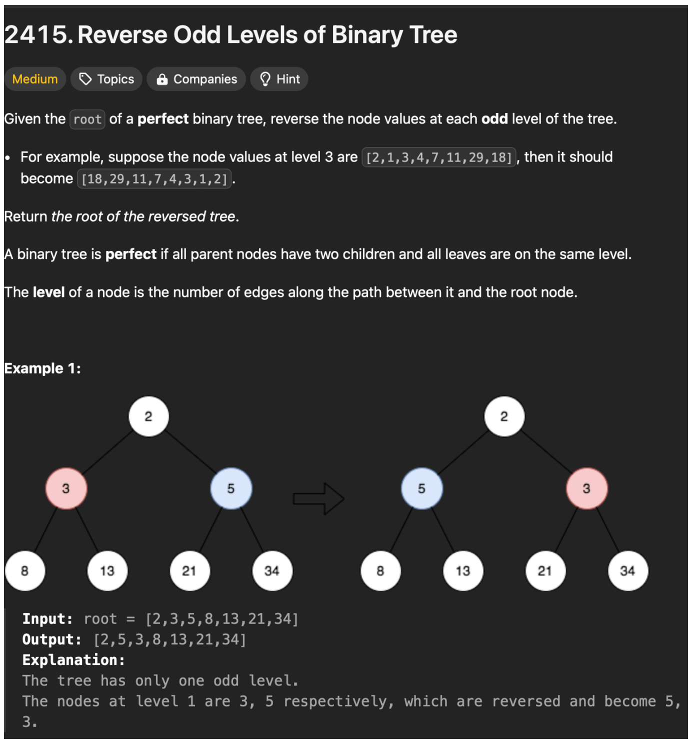


2415. Reverse Odd Levels of Binary Tree

Given the `root` of a **perfect** binary tree, reverse the node values at each **odd** level of the tree.

- For example, suppose the node values at level 3 are `[2,1,3,4,7,11,29,18]`, then it should become `[18,29,11,7,4,3,1,2]`.

Return *the root of the reversed tree*.

A binary tree is **perfect** if all parent nodes have two children and all leaves are on the same level.

The **level** of a node is the number of edges along the path between it and the root node.

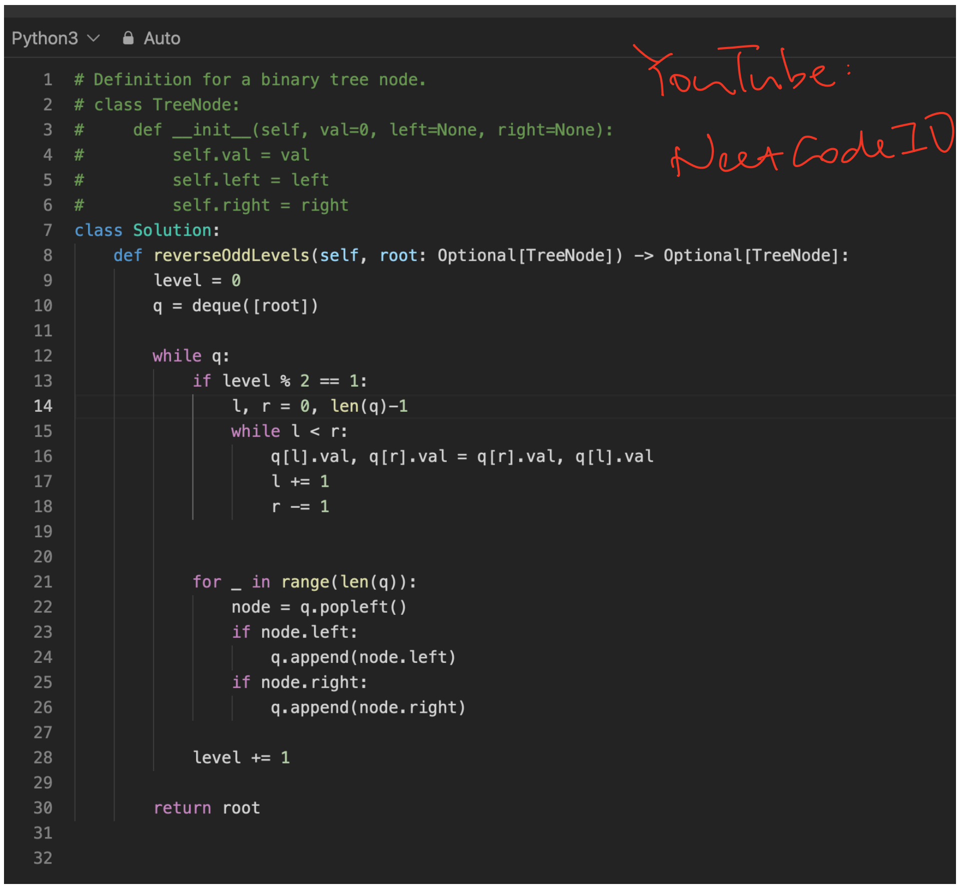
Example 1:

```
Input: root = [2,3,5,8,13,21,34]
Output: [2,5,3,8,13,21,34]
Explanation:
The tree has only one odd level.
The nodes at level 1 are 3, 5 respectively, which are reversed and become 5,
3.
```

- BFS

- only odd level

- swap val



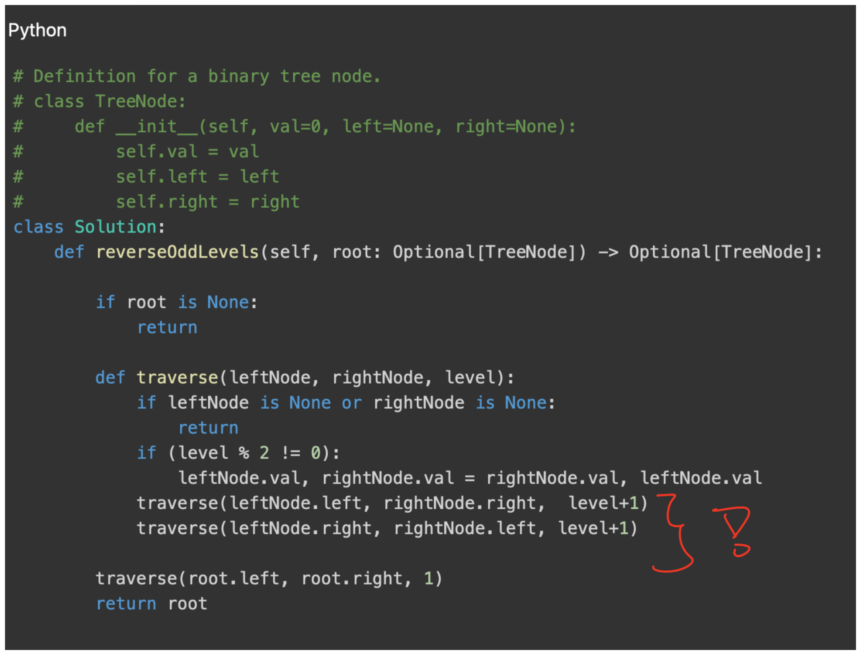
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def reverseOddLevels(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        level = 0
        q = deque([root])

        while q:
            if level % 2 == 1:
                l, r = 0, len(q)-1
                while l < r:
                    q[l].val, q[r].val = q[r].val, q[l].val
                    l += 1
                    r -= 1


            for _ in range(len(q)):
                node = q.popleft()
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)

            level += 1

        return root
```

YouTube:
NeetCodeIO

Deque is preferred over a list when append and pop is frequently used.

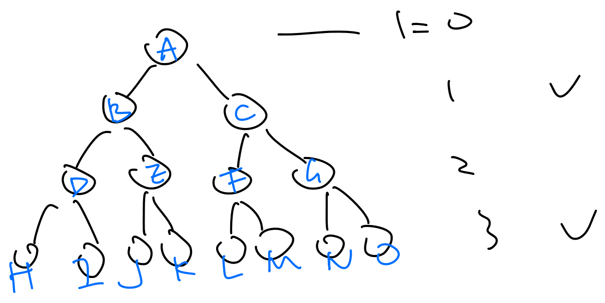
.popleft()
等于取 index = 0 .

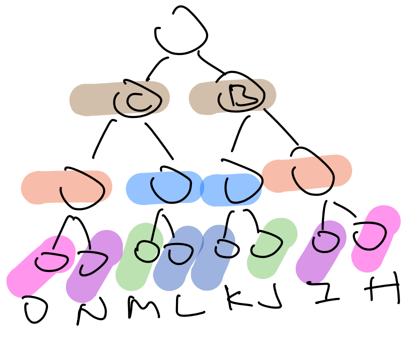
算是 Brute Force
efficiency 不高



Python

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def reverseOddLevels(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

        if root is None:
            return

        def traverse(leftNode, rightNode, level):
            if leftNode is None or rightNode is None:
                return
            if (level % 2 != 0):
                leftNode.val, rightNode.val = rightNode.val, leftNode.val
            traverse(leftNode.left, rightNode.right,  level+1)
            traverse(leftNode.right, rightNode.left, level+1)

        traverse(root.left, root.right, 1)
        return root
```





recursive. 反向想法可以