623. Add One Row to Tree

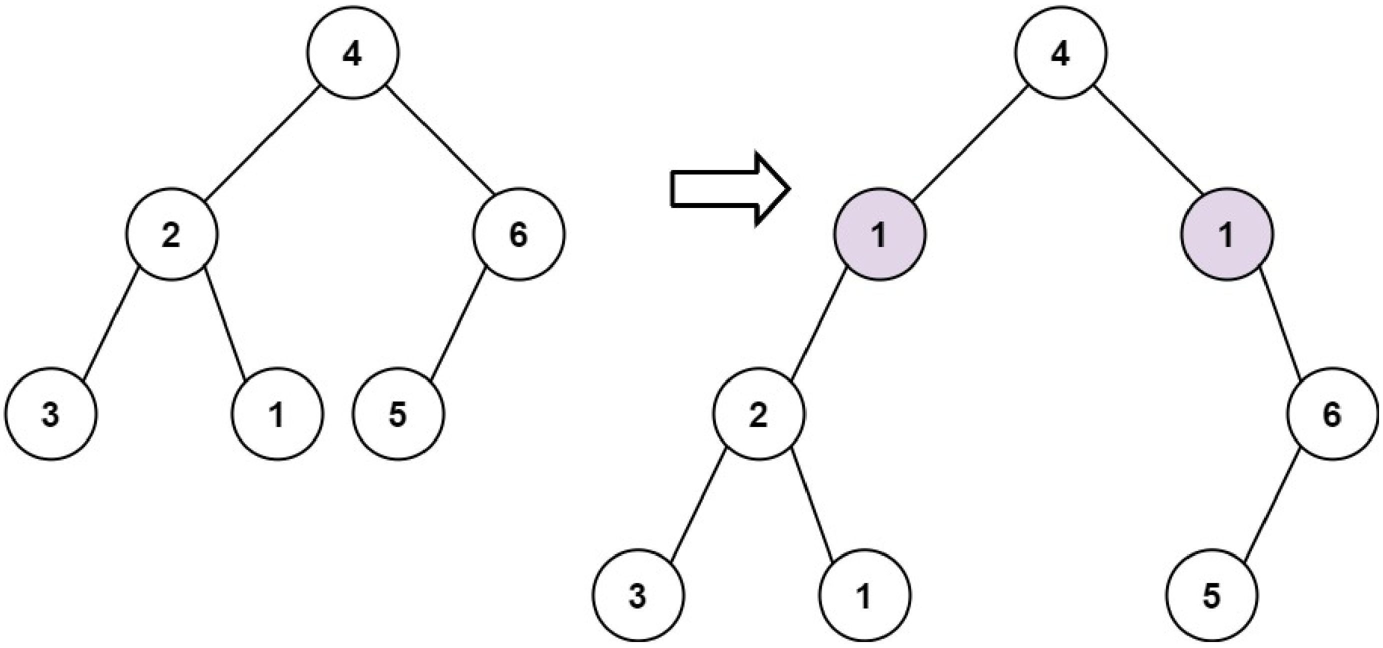
Given the `root` of a binary tree and two integers `val` and `depth`, add a row of nodes with value `val` at the given depth `depth`.

Note that the `root` node is at depth `1`.

The adding rule is:

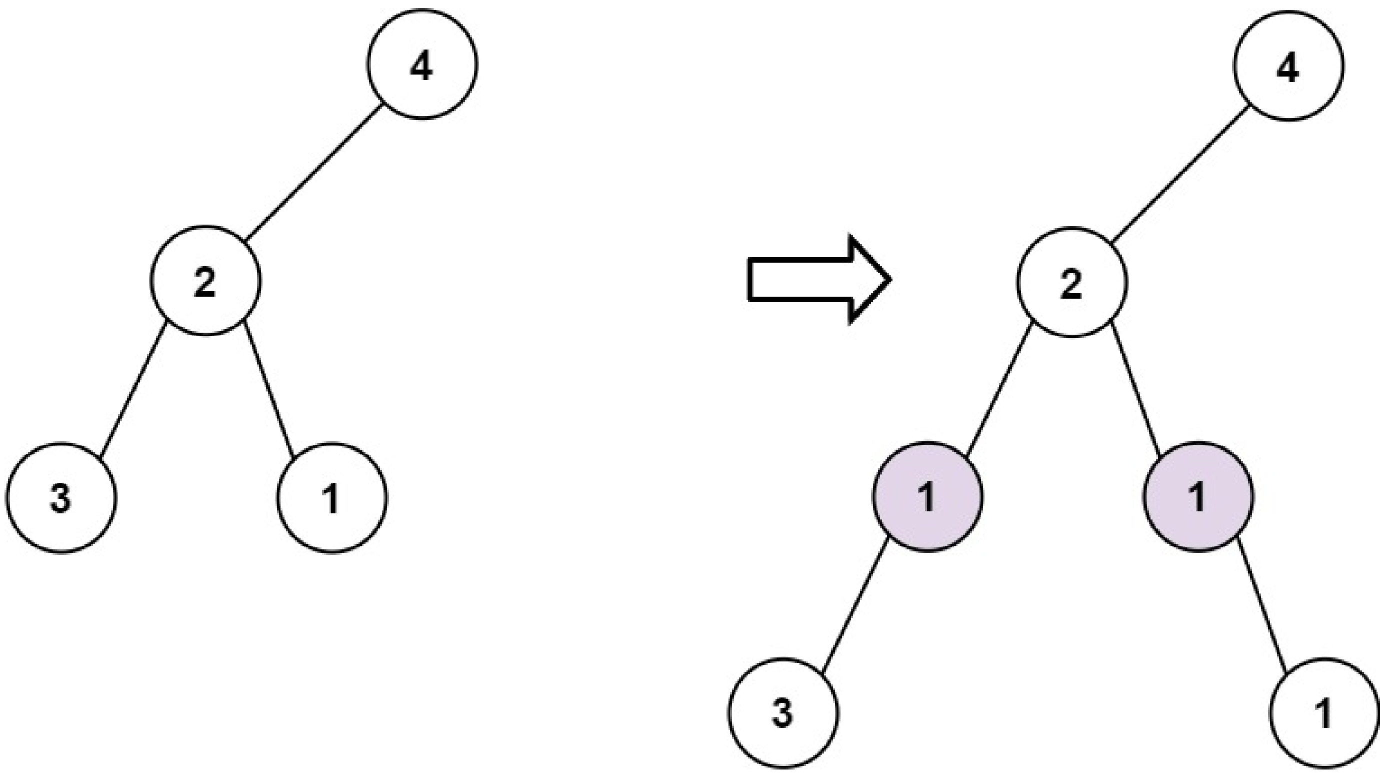
- Given the integer `depth`, for each not null tree node `cur` at the depth `depth - 1`, create two tree nodes with value `val` as `cur`'s left subtree root and right subtree root.
- `cur`'s original left subtree should be the left subtree of the new left subtree root.
- `cur`'s original right subtree should be the right subtree of the new right subtree root.
- If `depth == 1` that means there is no depth `depth - 1` at all, then create a tree node with value `val` as the new root of the whole original tree, and the original tree is the new root's left subtree.

Example 1:



```
Input: root = [4,2,6,3,1,5], val = 1, depth = 2
Output: [4,1,1,2,null,null,6,3,1,5]
```

Example 2:



```
Input: root = [4,2,null,3,1], val = 1, depth = 3
Output: [4,2,null,1,1,3,null,null,1]
```

Constraints:

- The number of nodes in the tree is in the range `[1, 10<sup>4</sup>]`.
- The depth of the tree is in the range `[1, 10<sup>4</sup>]`.
- `-100 <= Node.val <= 100`
- `-10<sup>5</sup> <= val <= 10<sup>5</sup>`
- `1 <= depth <= the depth of tree + 1`

```
# 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 addOneRow(self, root: Optional[TreeNode], val: int, depth: int) ->
Optional[TreeNode]:
        if depth==1:
            rootN = TreeNode(val)
            rootN.left = root
            return rootN
        return self.helper(root,1,depth,val)
```

```python
def helper(self,root,level,depth,val):
    if root is None:
        return
    if level==depth-1:
        leftNode = TreeNode(val)
        rightNode = TreeNode(val)
        N1 = root.left
        N2 = root.right
        root.left = leftNode
        leftNode.left = N1
        root.right = rightNode
        rightNode.right = N2
        return root
    root.left = self.helper(root.left,level+1,depth,val)
    root.right = self.helper(root.right,level+1,depth,val)
    return root
```