

## 865. Smallest Subtree with all the Deepest Nodes

Solved 

Medium

 Topics

 Companies

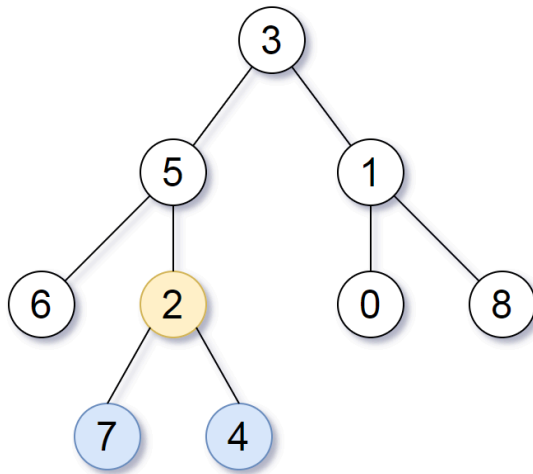
Given the `root` of a binary tree, the depth of each node is **the shortest distance to the root**.

Return *the smallest subtree* such that it contains **all the deepest nodes** in the original tree.

A node is called **the deepest** if it has the largest depth possible among any node in the entire tree.

The **subtree** of a node is a tree consisting of that node, plus the set of all descendants of that node.

**Example 1:**



**Input:** root = [3,5,1,6,2,0,8,null,null,7,4]

**Output:** [2,7,4]

**Explanation:** We return the node with value 2, colored in yellow in the diagram.

The nodes coloured in blue are the deepest nodes of the tree.

Notice that nodes 5, 3 and 2 contain the deepest nodes in the tree but node 2 is the smallest subtree among them, so we return it.

#### Example 2:

**Input:** root = [1]

**Output:** [1]

**Explanation:** The root is the deepest node in the tree.

#### Example 3:

**Input:** root = [0,1,3,null,2]

**Output:** [2]

**Explanation:** The deepest node in the tree is 2, the valid subtrees are the subtrees of nodes 2, 1 and 0 but the subtree of node 2 is the smallest.

#### Constraints:

- The number of nodes in the tree will be in the range [1, 500].
- $0 \leq \text{Node.val} \leq 500$
- The values of the nodes in the tree are **unique**.

## Python:

class Solution:

def subtreeWithAllDeepest(self, root: TreeNode) -> TreeNode:

def dfs(node):

if not node:

return (None, 0)

left\_node, left\_depth = dfs(node.left)

right\_node, right\_depth = dfs(node.right)

# If left subtree is deeper

if left\_depth > right\_depth:

return (left\_node, left\_depth + 1)

```

# If right subtree is deeper
if right_depth > left_depth:
    return (right_node, right_depth + 1)

# Both sides have equal depth
# Current node is the LCA of deepest nodes
return (node, left_depth + 1)

```

```

return dfs(root)[0]

```

## JavaScript:

```

var subtreeWithAllDeepest = function(root) {

    function dfs(node) {
        if (node === null) {
            return { node: null, depth: 0 };
        }

        const left = dfs(node.left);
        const right = dfs(node.right);

        // If left subtree is deeper
        if (left.depth > right.depth) {
            return { node: left.node, depth: left.depth + 1 };
        }

        // If right subtree is deeper
        if (right.depth > left.depth) {
            return { node: right.node, depth: right.depth + 1 };
        }

        // Both sides have equal depth
        // Current node is the LCA of deepest nodes
        return { node: node, depth: left.depth + 1 };
    }

    return dfs(root).node;
};

```

## Java:

```

class Solution {

    // Helper class to store subtree root and depth
    private static class Pair {

```

```

TreeNode node;
int depth;

Pair(TreeNode node, int depth) {
    this.node = node;
    this.depth = depth;
}
}

private Pair dfs(TreeNode root) {
    if (root == null) {
        return new Pair(null, 0);
    }

    Pair left = dfs(root.left);
    Pair right = dfs(root.right);

    // If left subtree is deeper
    if (left.depth > right.depth) {
        return new Pair(left.node, left.depth + 1);
    }

    // If right subtree is deeper
    if (right.depth > left.depth) {
        return new Pair(right.node, right.depth + 1);
    }

    // Both sides have equal depth
    // Current node is the LCA of deepest nodes
    return new Pair(root, left.depth + 1);
}

public TreeNode subtreeWithAllDeepest(TreeNode root) {
    return dfs(root).node;
}
}

```