# #LeetCode 222

https://leetcode.com/problems/count-complete-tree-nodes/description/

Yifeng Zeng

# #题目描述

222. Count Complete Tree Nodes

# #思路报告

The primitive idea is to traverse the whole tree with a counter. When a node is traversed we increase the counter. We can use any traverse method.

代码如下：

```
public int countNodes(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int count = 0;
    Deque<TreeNode> stack = new ArrayDeque<>();
    stack.push(root);

    while (!stack.isEmpty()) {
        TreeNode cur = stack.pop();
        count++;
        if (cur.right != null) {
            stack.push(cur.right);
        }
        if (cur.left != null) {
            stack.push(cur.left);
        }
    }

    return count;
}
```

We can traverse any tree to count the number of nodes with O(n) time. So we did not leverage the property of the complete tree. So there must be something we can improve. We know that for a complete tree node root, the left or right child has to be a perfect tree or both has to be. For a perfect tree, we can know the its number of nodes buy using $H^2$ - 1, where H is the height of the root, and we can get the H in O(H) time. So now we split the problem into some subproblem. Find the H of root.left and root.right, if they are the same height, root.left is a perfect tree and root.right is a complete tree, we can calculate the number of nodes of root.left by using $H^2$ - 1, plus the root itself. And recursively find the number of nodes of root.right. If the H of root.left and root.right is not the same, then root.right is a perfect tree, and root.left is a sub complete tree. We can do another recursive call to get the final result. So the time complexity is $O(H^2)$ where H = logn.

Code:

```
public int countNodes(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int left = findH(root.left);
    int right = findH(root.right);

    if (left == right) {
        return (1 << left) + countNodes(root.right);
    }

    return (1 << right) + countNodes(root.left);
}

private int findH(TreeNode root) {
    if (root == null) {
        return 0;
    }
    int h = 1;
    while (root.left != null) {
        root = root.left;
        h++;
    }
    return h;
}
```

# #套路总结

- When use a primitive idea, but without leveraging some property of the input, there might be

a way to improve.

- Use 1 << H to get $2^H$.
- Java运算符优先级有一个坑：

I used this first:

```
1 << left + countNodes(root.right);
```

Only to find out that '+' has higher priority of '<<' so I should the following:

```
(1 << left) + countNodes(root.right);
```