222. Count Complete Tree Nodes

Easy 8K 442

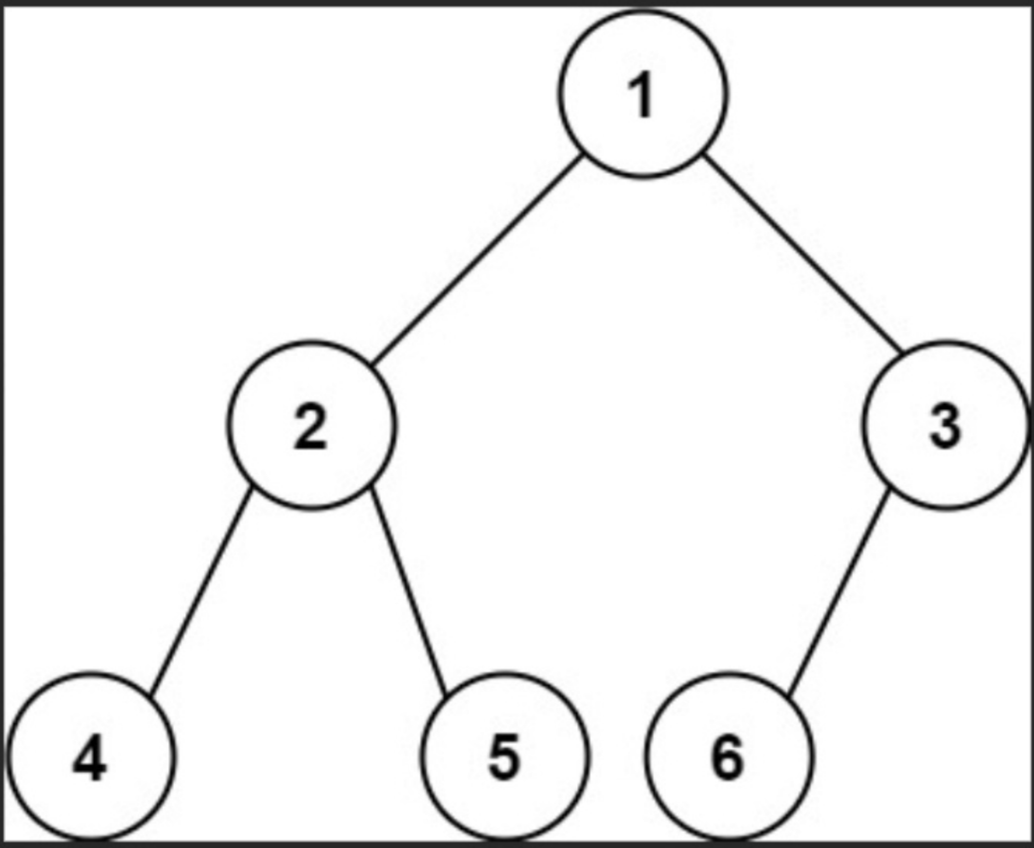
Companies

Given the `root` of a **complete** binary tree, return the number of the nodes in the tree.

According to **Wikipedia**, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between `1` and 2^h nodes inclusive at the last level `h`.

Design an algorithm that runs in less than `O(n)` time complexity.

Example 1:




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

Example 2:

```
Input: root = []
Output: 0
```

Example 3:

```
Input: root = [1]
Output: 1
```

Constraints:

- The number of nodes in the tree is in the range $[0, 5 * 10^4]$.
- $0 <= Node.val <= 5 * 10^4$
- The tree is guaranteed to be **complete**.

Accepted 596K | Submissions 956.4K | Acceptance Rate 62.3%

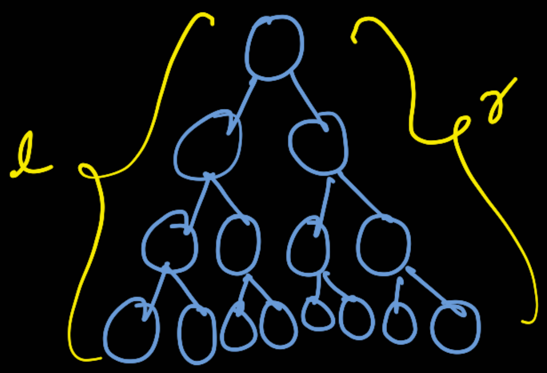
Approach 1:

we can do normal dfs or bfs and get the no. of nodes.

$T(n): O(n)$

But here we are not making use of the given description about tree. It is given that that is a CBT.

Approach 2:



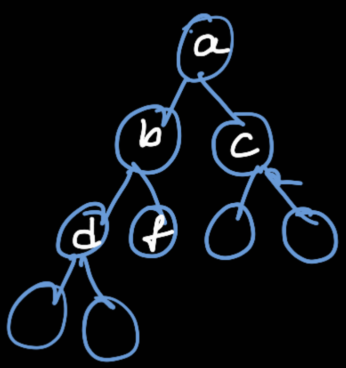
here if we find l and r by going recursively in their directions

we see that

$l = r$

which tells us that it is a FBT. So

Total nodes $= 2^{l} - 1$



$l \neq r$

So its not FBT with root as a.

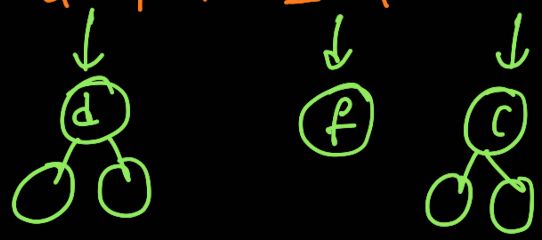
its not FBT with root as b also

its FBT with root as c

its FBT with root as

d and also with root as f.

Total nodes = a + b + 2^2-1 + 2^1-1 + 2^2-1



= 9

Algorithm:

→ Starting with root find l and r
→ if they are equal return 2^l-1
→ if not
then return 1 + Count(root left)
+ Count(root right)

1 → i.e. root node

```
int leftheight( root)
{
    if(root is null) return 0

    return 1+ leftheight(root→left)
}


int rightheight( root)
{
    if(root is null) return 0

    return 1+ rightheight(root→right)
```

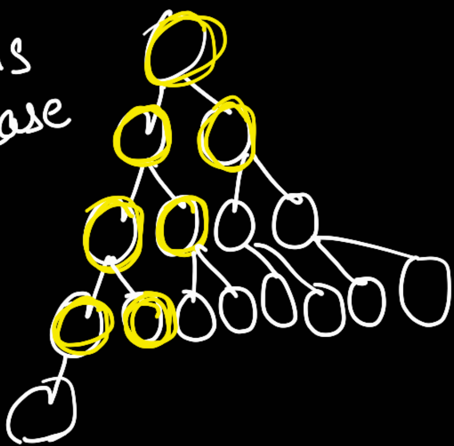
```
}

int count(root)
{
    l = leftheight(root)
    r = leftheight(root)

    if(l==r) return 2^l - 1

    return 1 + count(root->left)
             + count(root->right)
}
```

$T(n) : O(\log n)^2$

Q: How $T(n)$ is $O(\log n)^2$?

let the Tree is

note: This is the worst case input



we do exactly highlighted no. of ($2\log n$) calls. which is basically the height no. of calls.

So

$$T(n) = \log n \, (2 \log n)$$

$$= 2 (\log n)^2$$

$$= c \cdot (\log n)^2$$