Previous Next

222. Count Complete Tree Nodes

June 19, 2019 | 56.1K views

Average Rating: 4.70 (81 votes)

Given a **complete** binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely filled, 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.

Example:

```
Input:
    1
   / \
  2   3
 / \  /
4  5 6

Output: 6
```

Solution

Approach 1: Linear Time

Intuition

This problem is quite popular at Google during the last year. The naive solution here is a linear time one-liner which counts nodes recursively one by one.

Implementation

Java | Python | Copy

```
class Solution:
    def countNodes(self, root: TreeNode) -> int:
        return 1 + self.countNodes(root.right) + self.countNodes(root.left) if root else 0
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$.
- Space complexity : $\mathcal{O}(d) = \mathcal{O}(\log N)$ to keep the recursion stack, where d is a tree depth.

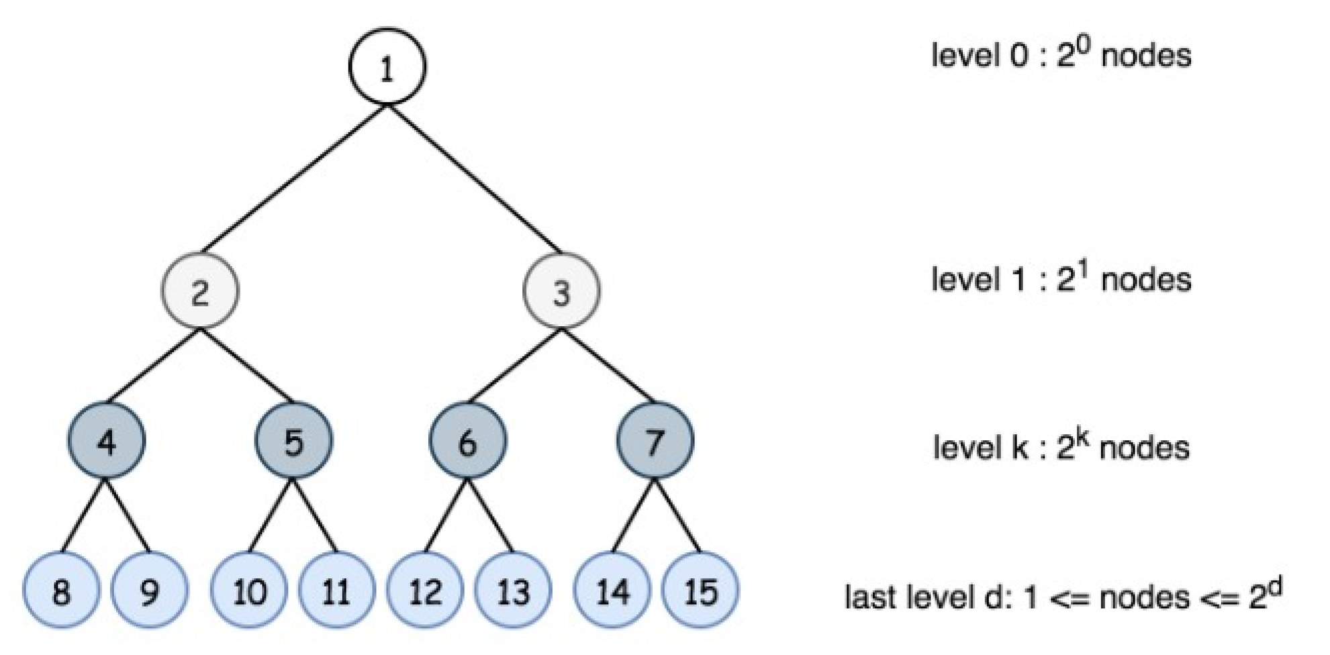
Approach 2: Binary search

Intuition

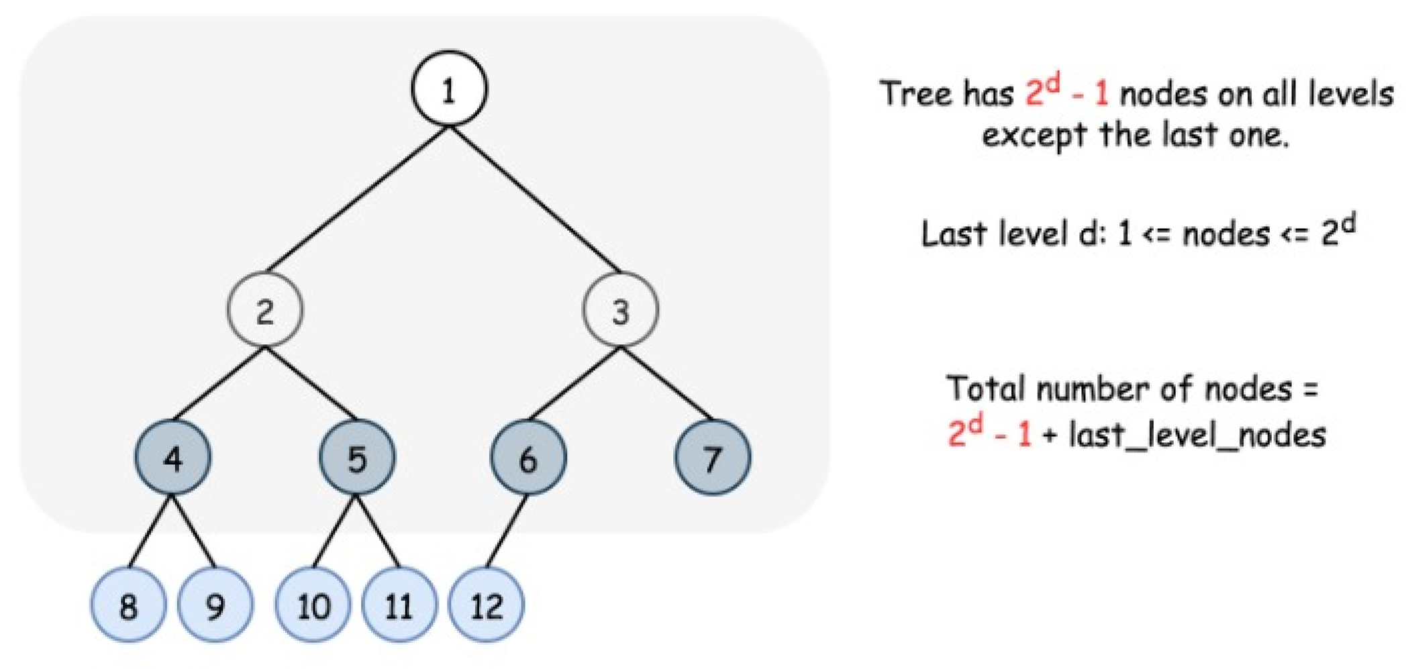
Approach 1 doesn't profit from the fact that the tree is a complete one.

> In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.

That means that complete tree has 2^k nodes in the kth level if the kth level is not the last one. The last level may be not filled completely, and hence in the last level the number of nodes could vary from 1 to 2^d, where d is a tree depth.



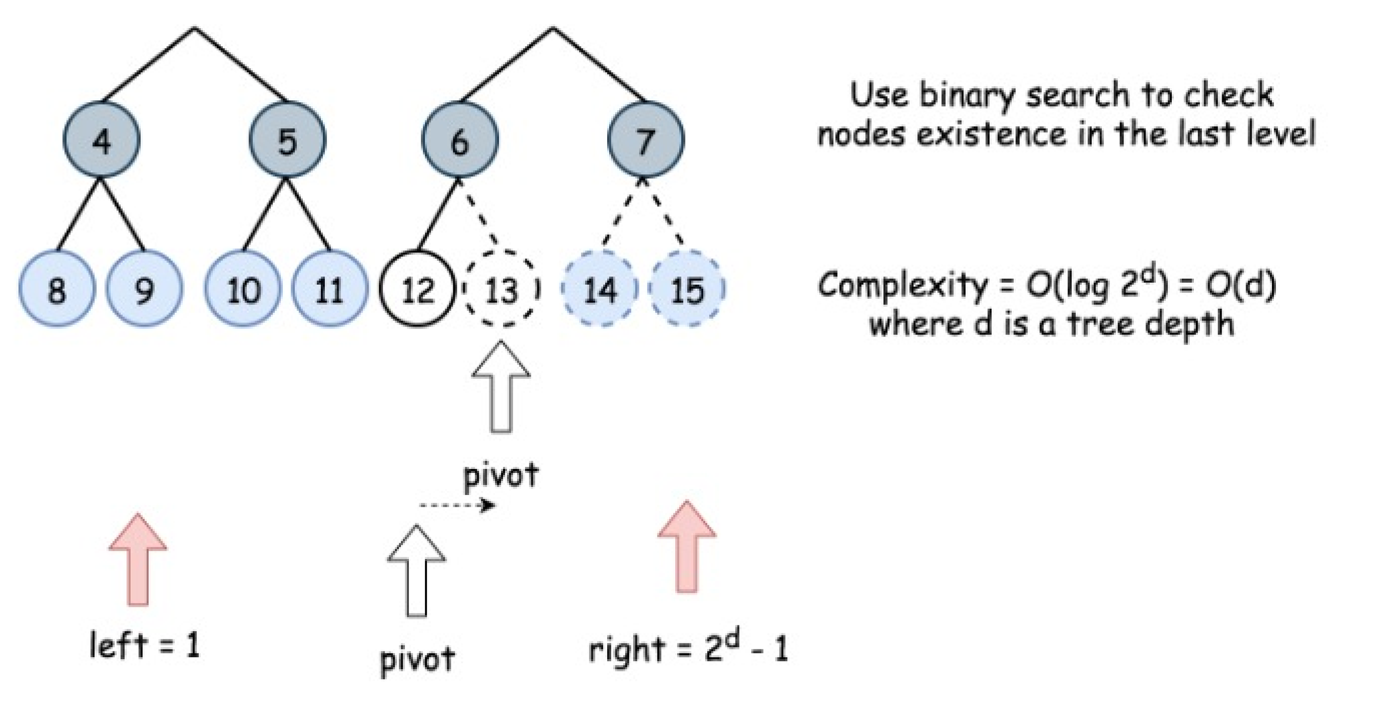
Now one could compute the number of nodes in all levels but the last one: $\sum_{k=0}^{k=d-1} 2^k = 2^d - 1$. That reduces the problem to the simple check of how many nodes the tree has in the last level.



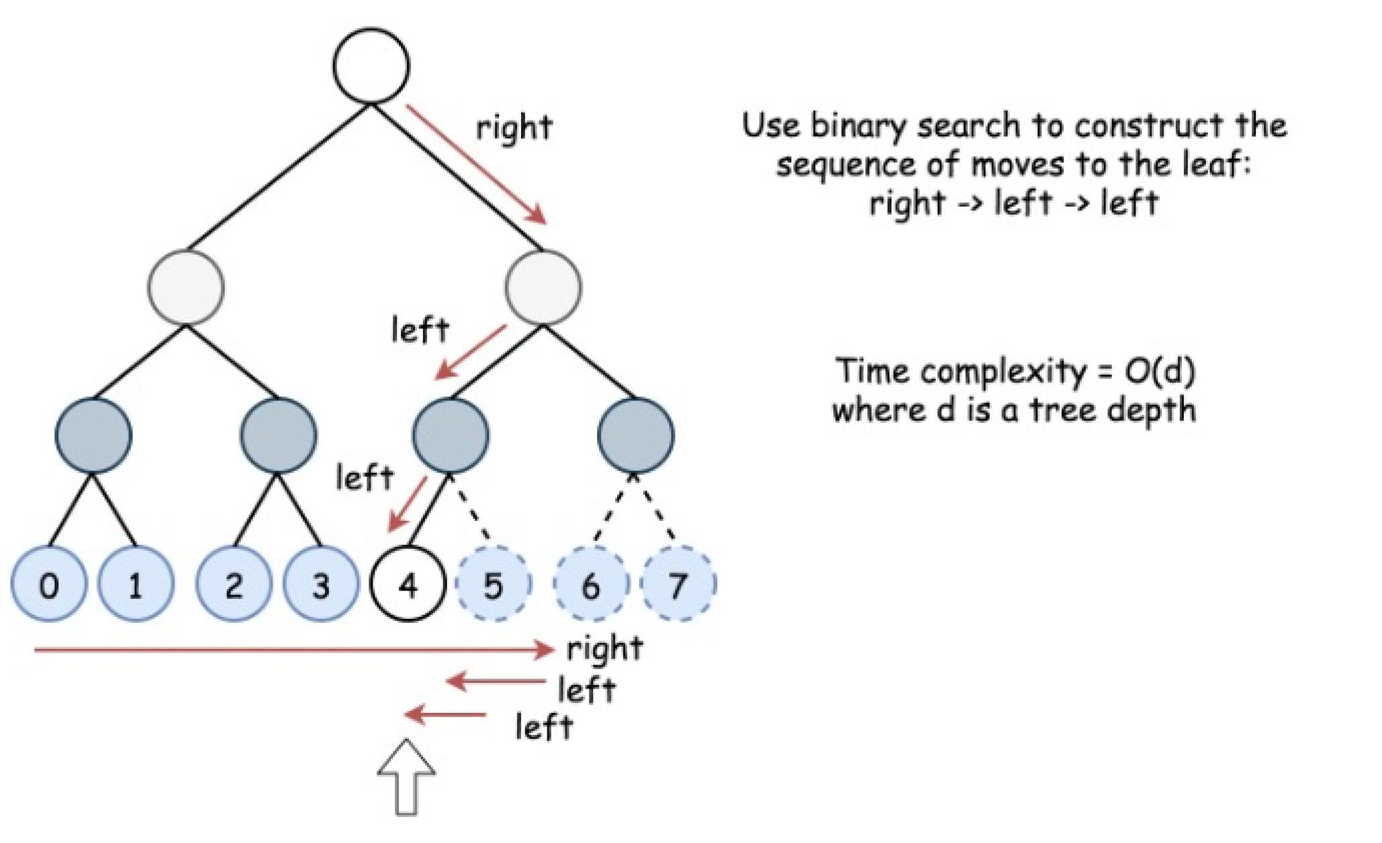
Now there are two questions:

1. How many nodes in the last level have to be checked?
2. What is the best time performance for such a check?

Let's start from the first question. It's a complete tree, and hence all nodes in the last level are *as far left as possible*. That means that instead of checking the existence of all 2^d possible leafs, one could use binary search and check $\log(2^d) = d$ leafs only.



Let's move to the second question, and enumerate potential nodes in the last level from 0 to $2^d - 1$. How to check if the node number idx exists? Let's use binary search again to reconstruct the sequence of moves from root to idx node. For example, idx = 4. idx is in the second half of nodes `0,1,2,3,4,5,6,7` and hence the first move is to the right. Then idx is in the first half of nodes `4,5,6,7` and hence the second move is to the left. The idx is in the first half of nodes `4,5` and hence the next move is to the left. The time complexity for one check is $\mathcal{O}(d)$.



1 and 2 together result in $\mathcal{O}(d)$ checks, each check at a price of $\mathcal{O}(d)$. That means that the overall time complexity would be $\mathcal{O}(d^2)$.

Algorithm

- Return 0 if the tree is empty.
- Compute the tree depth `d`.
- Return 1 if `d == 0`.
- The number of nodes in all levels but the last one is $2^d - 1$. The number of nodes in the last level could vary from 1 to 2^d. Enumerate potential nodes from 0 to $2^d - 1$ and perform the binary search by the node index to check how many nodes are in the last level. Use the function `exists(idx, d, root)` to check if the node with index idx exists.
- Use binary search to implement `exists(idx, d, root)` as well.
- Return $2^d - 1$ + the number of nodes in the last level.

Implementation

Java | Python | Copy

```
25                node = node.right
26                left = pivot + 1
27        return node is not None
28
29    def countNodes(self, root: TreeNode) -> int:
30        # if the tree is empty
31        if not root:
32            return 0
33
34        d = self.compute_depth(root)
35        # if the tree contains 1 node
36        if d == 0:
37            return 1
38
39        # Last level nodes are enumerated from 0 to 2**d - 1 (left -> right).
40        # Perform binary search to check how many nodes exist.
41        left, right = 1, 2**d - 1
42        while left <= right:
43            pivot = left + (right - left) // 2
44            if self.exists(pivot, d, root):
45                left = pivot + 1
46            else:
47                right = pivot - 1
48
49        # The tree contains 2**d - 1 nodes on the first (d - 1) levels
50        # and left nodes on the last level.
51        return (2**d - 1) + left
```

Complexity Analysis

- Time complexity : $\mathcal{O}(d^2) = \mathcal{O}(\log^2 N)$, where d is a tree depth.
- Space complexity : $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 40 Sort By

Type comment here... (Markdown is supported)

Preview Post

bridge 136 August 13, 2019 12:13 AM
binary search in binary search, this problem is pretty interesting
99 Share Reply
SHOW 1 REPLY

hardfault 266 January 3, 2020 5:27 AM
I solved it in O(n) and was beaming with pride until I saw the binary search approach. :O
48 Share Reply
SHOW 4 REPLIES

parambole 78 November 24, 2019 12:20 AM
This is probably one of the most well-written solutions in LC
36 Share Reply
SHOW 1 REPLY

bhavneet 15 March 10, 2020 7:23 PM
it should be hard!
15 Share Reply
SHOW 1 REPLY

mohib13 12 September 26, 2019 5:42 PM
Line 40 // Last level nodes are enumerated from 0 to 2**d - 1 (left -> right).
Line 42, left = 1 instead of 0. Why?
10 Share Reply
SHOW 2 REPLIES

novice87 254 June 22, 2019 6:34 AM
Can someone explain this line :

```
int left = 1, right = (int)Math.pow(2, d) - 1;
```

Read More
8 Share Reply
SHOW 7 REPLIES

alexander1089 12 June 24, 2020 4:24 AM
I always get confused in binary search in deciding the following things, should left = mid or mid + 1, same for right, and whether the termination condition should be (left < right) or (left <= right). is there a trick to know that for a particular problem ?
7 Share Reply

tamervi 13 October 8, 2019 9:35 AM
Why not simply pivot = (left + right)/2 ?
pivot = left + (right - left)/2 = left + right/2 - left/2 = left/2 + right/2 = (left + right)/2
6 Share Reply
SHOW 1 REPLY

engineerraj 13 February 10, 2020 11:36 AM
Pretty neat ! I wrote the one liner and was thinking why the heck was this a medium question. lesson learnt devil is in the details .. Thanks for the clear explanation!
4 Share Reply

jinwei14 50 September 13, 2019 3:03 PM
the naive approach is just counting the number of nodes. What is the difference between Count Complete Tree Nodes?????Can someone help?
4 Share Reply
SHOW 3 REPLIES

‹ 1 2 3 4 ›