129. Sum Root to Leaf Numbers

Medium 7.1K 110

Companies

You are given the `root` of a binary tree containing digits from `0` to `9` only.

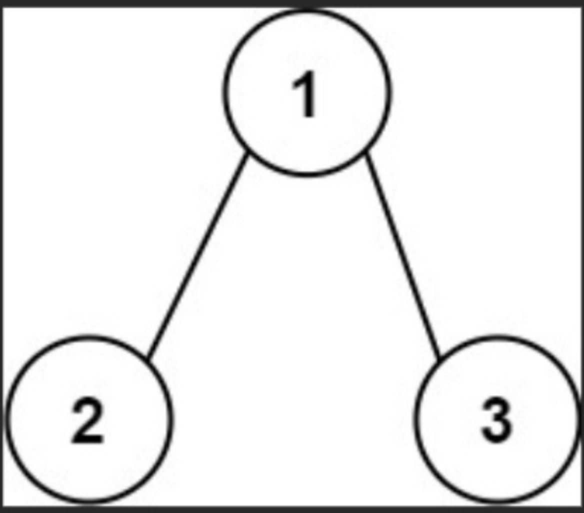
Each root-to-leaf path in the tree represents a number.

- For example, the root-to-leaf path `1 -> 2 -> 3` represents the number `123`.

Return *the total sum of all root-to-leaf numbers*. Test cases are generated so that the answer will fit in a **32-bit** integer.

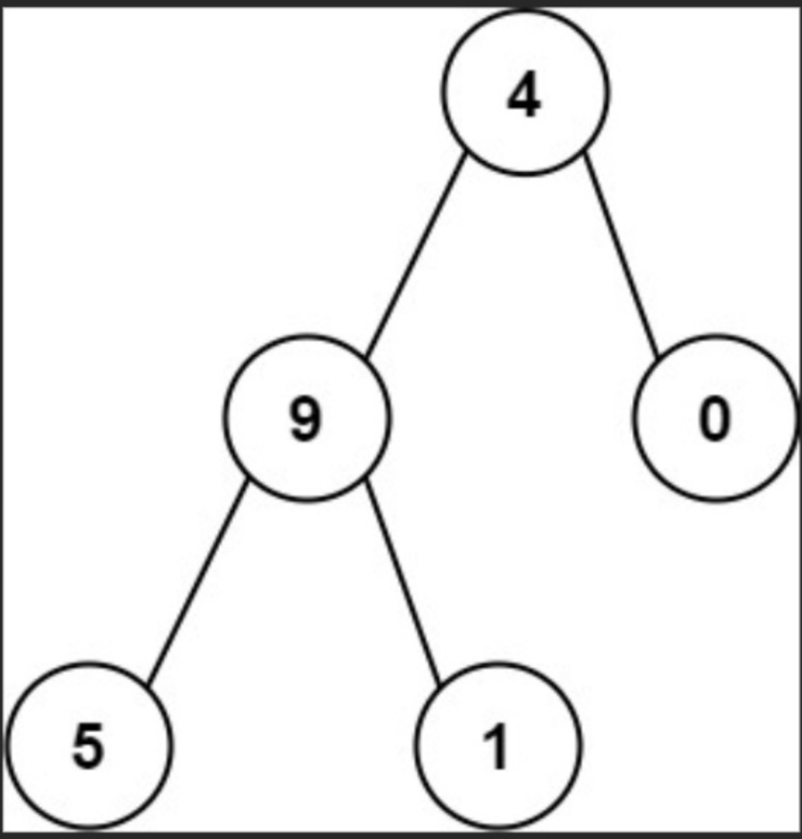
A **leaf** node is a node with no children.

Example 1:



```
Input: root = [1,2,3]
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
```

Example 2:



```
Input: root = [4,9,0,5,1]
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Approach 1: Recursive implementation

whenever we see a leaf node add the num to Sum.

```
Sum = 0,
find ( root , num = 0 )
{
    num = num * 10 + root -> val
    -> no need to check if root is null becoz
       we only traverse non null nodes.

    if ( root is a leaf node )
        Sum = Sum + num

    if ( root has non null left node )
        find ( root -> left , num )

    if ( root has non null right node )
        find ( root -> right , num )
}
```

T(n) : O(n)

Approach 2: Iterative implementation

we can keep track of Current Sum along with node as a pair in queue

```
queue <pair <Treenode*, int >> q , num=0
                                   Sum=0
q.push({ root , root->val })

while(q is not empty)
{
        auto node = q.front().first
          int num = q.front().second

      if(node is a leaf node)
            Sum = Sum+num

      if(node has non null left node)
            q.push({node->left , num*10 +
                                 node->left->val})

      if(node has non null right node)
            q.push({ node->right , num*10 +
                                   node->right->val})
}
```

T(n) : O(n)
S(n) : O(max nodes at a level)