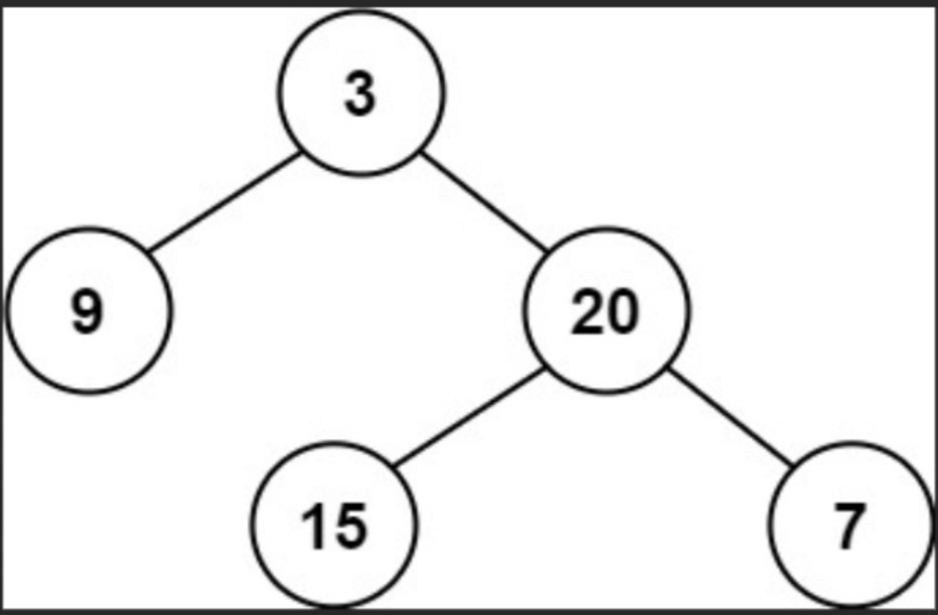
110. Balanced Binary Tree

Easy 9.8K 557

Companies

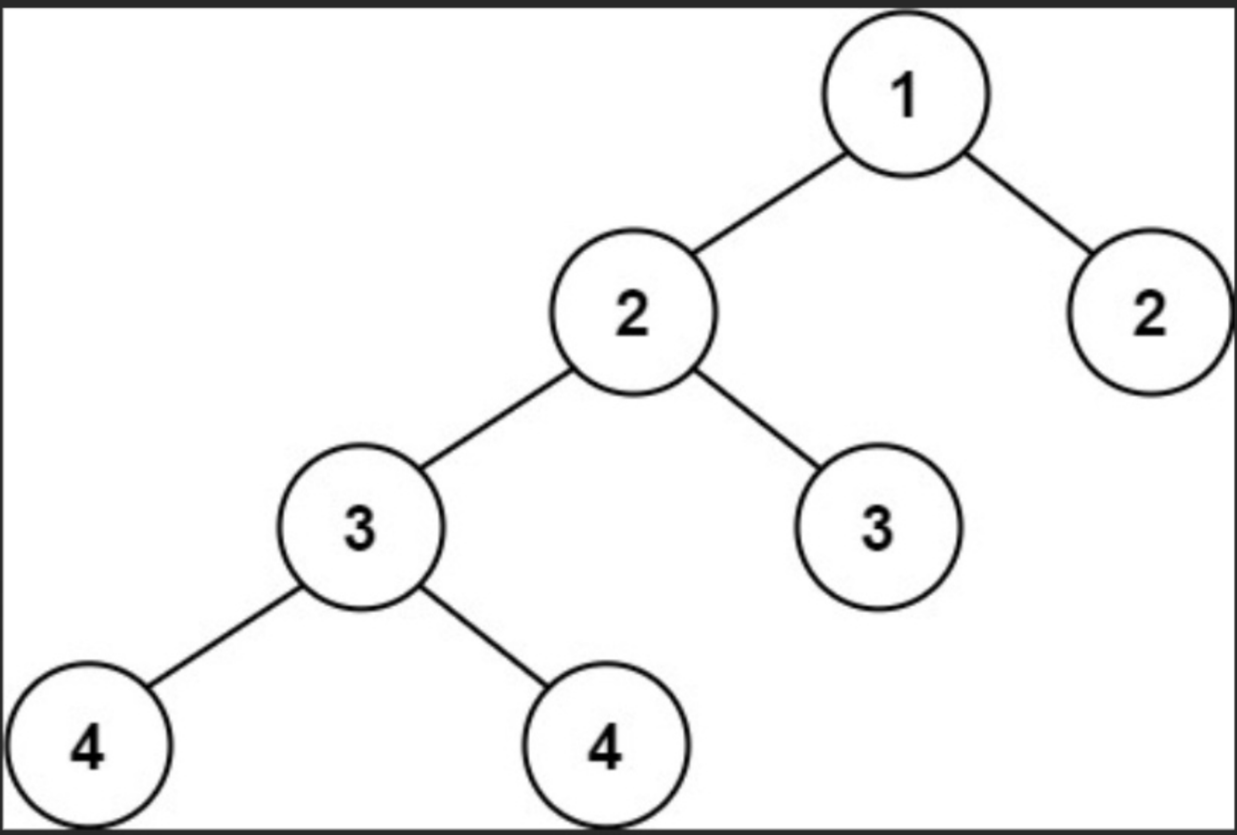
Given a binary tree, determine if it is **height-balanced**.

Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: true
```

Example 2:



```
Input: root = [1,2,2,3,3,null,null,4,4]
Output: false
```

Example 3:

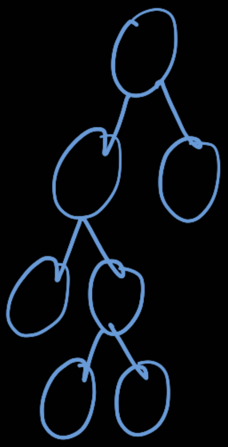
```
Input: root = []
Output: true
```

Constraints:

- The number of nodes in the tree is in the range `[0, 5000]`.
- `$-10^4$ <= Node.val <= $10^4$`

Accepted 1.2M | Submissions 2.4M | Acceptance Rate 50.3%

Approach 1: Using height() function



at everynode we find max height in LST and maxheight in RST and then we check if their difference is ≤ 1
If it is we move to next node
or else we return false.

```
bool isbalanced(root)
{

    l = maxheight(root→left)
    r = maxheight(root→right)

    if ((l-r) > 1)
        return false

    return  isbalanced(root→left) &&
            isbalanced(root→right)
}
```

T(n): $O(n*n)$
$= O(n^2)$

i.e. for skewed
Tree

Approach 2: Using Post Order Traversal

Instead of finding height at each node individually which is doing repetitive calculations we can get that heights at a node in single traversal

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

        l = check (root -> left)
        if( l == -1)    return -1

        r = check (root -> right)
        if ( r == -1)    return -1

        if( abs ( l - r) > 1)
            return -1

     return max( l, r )

}


bool   isBalanced ( root)
{
```

```
return    check(root) != -1
```

$T(n) : O(n)$