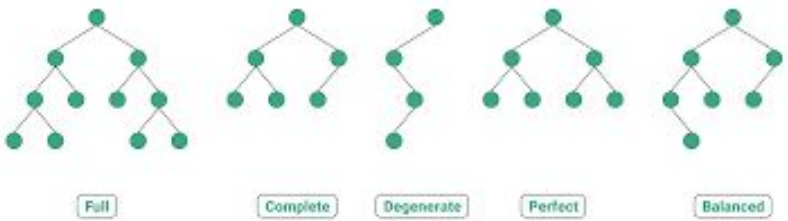
Leetcode 110.
Balanced Binary Tree

2024.01.07

원루빈

1 Binary Tree

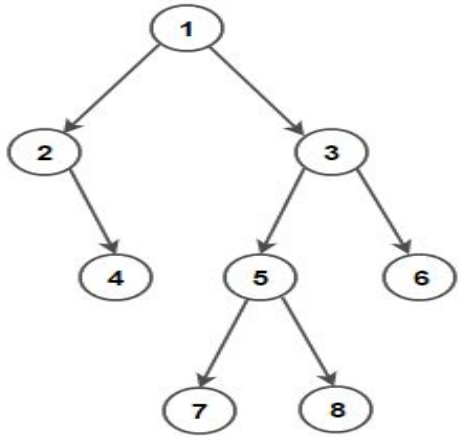
[DEF] A binary tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child.



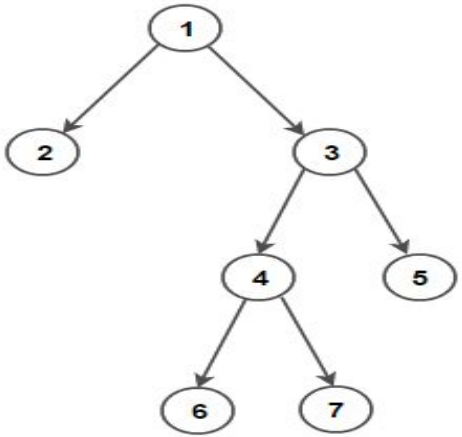

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.parent = None
        self.left = None
        self.right = None


class BinaryTree:
    NodeCls = Node

    def __init__(self):
        self.root = None

    def insert(self, key, parent):
        new = self.NodeCls(key)
        if parent is None:
            if self.root is None:
                self.root = new
                return new
            raise Exception("a root already exists")

        if not parent.left:
            parent.left = new
            new.parent = parent

        elif not parent.right:
            parent.right = new
            new.parent = parent

        else:
            raise Exception("a node cannot have more than two children")
        return new
```

2 Height-Balanced Binary Tree

[DEF] A height-balanced binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.



Height Balanced Tree

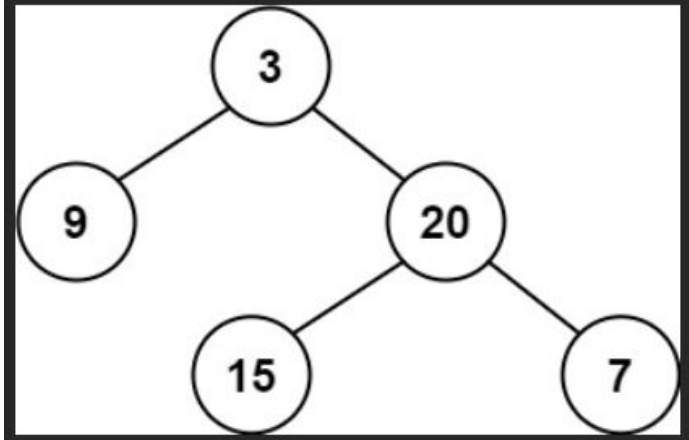


Not a Height Balanced Tree

Problem Description

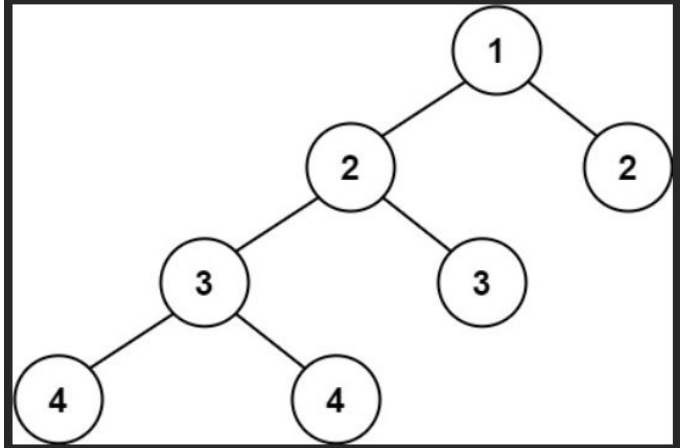
Given a binary tree, determine if it is height-balanced .

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

3 Step-by-Step Approach

1. **Recursion**:
   Write a helper function that recursively checks each node in the tree.
   (info: if current node is balanced, height of the subtree)

2. **Height Calculation**:
   For each node, calculate the height of its left & right subtrees. (recursively)

3. **Balance Check**:
   current node's left & right balanced? height difference <= 1?

3 Step-by-Step Approach

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

1. **Recursion**:

   Write a helper function that recursively checks each node in the tree.

   (info: if current node is balanced, height of the subtree)

Base Case: if there is no node

Recursively check → 2 things of left & right subtrees

[1] is it balanced?
[2] what is the height?

3 Step-by-Step Approach

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

2. **Height Calculation**:

   For each node, calculate the height of its left & right subtrees. (recursively)

```python
# Calculate the height of the current node
height = max(left_height, right_height) + 1
```

3. **Balance Check**:

   current node's left & right balanced? height difference <= 1?

```python
# Check if the current node is balanced
is_balanced = left_balanced and right_balanced and abs(left_height - right_height) <= 1
```

4 Code Solution

```python
class Solution:
    def isBalanced(self, root):
        def dfs(node):
            # Base case:
            if not node:
                return True, 0

            # Recursively check left & right subtrees
            left_balanced, left_height = dfs(node.left)
            right_balanced, right_height = dfs(node.right)

            # Check if the current node is balanced
            is_balanced = left_balanced and right_balanced and abs(left_height - right_height) <= 1

            # Calculate the height of the current node
            height = max(left_height, right_height) + 1

            return is_balanced, height

        return dfs(root)[0]
```

감사합니다!

THANK YOU