2236. Root Equals Sum of Children



Problem Description

In this problem, we're working with a very small binary tree, one that only has three nodes: a root node, a left child, and a right child. We're asked to determine if the value of the root node is equal to the sum of the values of its two children. If this condition is true, we should return true; otherwise, we return false. It's a straightforward problem focused on understanding the basic properties of a binary tree and how to access the values of its nodes.

Intuition

for complex traversal or recursion. The intuition behind the solution is based on the direct relationship between the nodes. By definition, the sum of the children's values can be compared directly with the root's value. We can arrive at this solution approach by considering the following:

Given that the tree has exactly three nodes, this problem is greatly simplified compared to most tree problems. There's no need

Since the tree structure is guaranteed, we know that the root, the left child, and the right child all exist.

- We can access the values of these three nodes directly using root.val, root.left.val, and root.right.val.
- The sum of the values of the left and right children can be obtained with a simple addition: root.left.val + root.right.val. • By comparing this sum to root.val, we can determine whether they are equal and thus return true or false accordingly.
- This solution leverages the fact that, in Python, comparisons return a boolean value, which aligns perfectly with the requirement

Solution Approach

The implementation of the solution is straightforward given the simplicity of the problem. Here's a step-by-step walkthrough of the algorithm used in the provided solution code:

of our function to return a boolean.

self.val = val

self.left = left

self.right = right

1. Accept the root of the binary tree as an argument. The problem statement guarantees that the tree consists of exactly three nodes. 2. Since the tree structure is known and fixed, we have direct access to the root's value as well as its left and right children's values.

- 3. Use the equality comparison operator == to compare the root's value root.val with the sum of its children's values root.left.val + root.right.val.
- 4. The comparison root.val == root.left.val + root.right.val will evaluate to either true or false. 5. Since the desired output of checkTree function is a boolean, directly return the result of that comparison.
- structure allows us to perform a direct comparison without the need for additional code or complex logic. The code snippet follows this logic concisely:

There are no complex data structures, patterns, or algorithms needed to solve this problem. The simplicity of the binary tree's

Definition for a binary [tree](/problems/tree_intro) node. class TreeNode: def __init__(self, val=0, left=None, right=None):

```
class Solution:
    def checkTree(self, root: Optional[TreeNode]) -> bool:
        return root.val == root.left.val + root.right.val
  In this code, the comparison root.val == root.left.val + root.right.val is the core of the solution. This comparison uses
  fundamental arithmetic (addition) and a basic equality check. It exploits the fact that in Python, a comparison operation itself
  yields a boolean result. This concise code results in an elegant and efficient implementation of the required functionality.
```

Here's the step-by-step walkthrough, with this specific tree in mind:

Accessing the root's value, we have root val which is 10.

the need for traversing the tree or implementing complex logic.

Example Walkthrough Let's consider a simple example to illustrate the solution approach. Imagine we have a binary tree as follows: 10

the root's value.

The input provided to the checkTree function is the root of the binary tree, which in this example has the value 10. The binary tree consists of exactly three nodes, which satisfies the problem requirement.

We then perform a comparison operation: root.val == root.left.val + root.right.val. Substituting the values from our example gives us 10 == 4 + 6.

Similarly, we can access the left child's value root.left.val which is 4, and the right child's value root.right.val which is 6.

The comparison 10 == 10 evaluates to true, which is the result we expect, given that the sum of the children's values equals

- In this example, the checkTree function would therefore return true, correctly indicating that the value of the root is equal to the
- Solution Implementation

sum of its children's values. This demonstrates the effectiveness of directly comparing the values to solve this problem without

Attributes: val (int): The value of the node.

Definition for a binary tree node.

A class to represent a node in a binary tree.

left (TreeNode): A reference to the left subtree.

right (TreeNode): A reference to the right subtree.

// Constructor for tree node with specified value and children

Initializes a TreeNode with a value and optional left and right subtrees.

```
def __init__(self, val=0, left=None, right=None):
```

Python

class TreeNode:

```
Parameters:
       val (int): The value to store in the node. Default is 0.
        left (TreeNode): The left subtree. Default is None.
       right (TreeNode): The right subtree. Default is None.
       self.val = val
       self.left = left
       self.right = right
class Solution:
   def check_tree(self, root: TreeNode) -> bool:
       Checks if the value of the root node is equal to the sum of the values
       of its left and right child nodes.
       Parameters:
       root (TreeNode): The root node of the binary tree.
       Returns:
       bool: True if the root node's value is equal to the sum of its children's values,
              False otherwise.
       # Check if root exists to prevent AttributeError on accessing None.
        if root is None:
            return False
       # Calculate the sum of the values of the left and right child nodes.
       children_sum = (root.left.val if root.left else 0) + (root.right.val if root.right else 0)
       # Return whether the root's value is equal to the sum of its children's values.
       return root.val == children_sum
Java
/**
* Definition for a binary tree node.
class TreeNode {
   int val; // Value of the node
   TreeNode left; // Reference to the left child node
   TreeNode right; // Reference to the right child node
   // Constructor for tree node with no children
   TreeNode() {}
   // Constructor for tree node with a specified value
   TreeNode(int val) { this.val = val; }
```

```
TreeNode(int val, TreeNode left, TreeNode right) {
```

class Solution {

/**

this.val = val;

this.left = left;

this.right = right;

return false;

```
* Checks if the value of the root is equal to the sum of its left and right child nodes.
     * @param root The root of the binary tree.
     * @return true if the root's value is the sum of its children's values, false otherwise.
    public boolean checkTree(TreeNode root) {
       // Check if the value of the root node is the sum of values of the left and right child nodes.
       // It assumes root, root.left, and root.right are not null.
        return root.val == root.left.val + root.right.val;
C++
// Definition for a binary tree node.
struct TreeNode {
                       // The value of the node
    int val;
   TreeNode *left; // Pointer to the left child
    TreeNode *right;
                        // Pointer to the right child
    // Constructor to initialize node with default value 0 and null children
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    // Constructor to initialize node with a given integer value and null children
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    // Constructor to initialize node with a given value and left and right children
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
class Solution {
public:
    // This method checks if the root node's value is equal to the sum of its left and right children's values.
    bool checkTree(TreeNode* root) {
       // Ensure that the left and right children are not nullptr before accessing their values
        if (root == nullptr || root->left == nullptr || root->right == nullptr) {
            // If one of the nodes is nullptr, we cannot perform the check, so we return false
```

// This function checks if the value of the root node of a binary tree is equal to the sum of the values of its left and right ch

// Perform the check by comparing the root's value with the sum of its children's values

// Return true if the value of the root node equals the sum of the values

// of its left and right child nodes. Otherwise, return false.

return root->val == root->left->val + root->right->val;

// A TypeScript interface for the binary tree node structure.

function checkTree(root: TreeNode | null): boolean {

// Check if the root node is not null.

```
return root.val === (root.left ? root.left.val : 0) + (root.right ? root.right.val : 0);
// If the root is null, the binary tree does not exist, hence return false.
return false;
```

};

TypeScript

interface TreeNode {

val: number;

if (root) {

left: TreeNode | null;

right: TreeNode | null;

Definition for a binary tree node.

```
class TreeNode:
   A class to represent a node in a binary tree.
   Attributes:
   val (int): The value of the node.
   left (TreeNode): A reference to the left subtree.
   right (TreeNode): A reference to the right subtree.
   def ___init___(self, val=0, left=None, right=None):
        Initializes a TreeNode with a value and optional left and right subtrees.
        Parameters:
       val (int): The value to store in the node. Default is 0.
        left (TreeNode): The left subtree. Default is None.
        right (TreeNode): The right subtree. Default is None.
        self.val = val
        self.left = left
        self.right = right
class Solution:
   def check_tree(self, root: TreeNode) -> bool:
       Checks if the value of the root node is equal to the sum of the values
       of its left and right child nodes.
        Parameters:
        root (TreeNode): The root node of the binary tree.
       Returns:
        bool: True if the root node's value is equal to the sum of its children's values,
              False otherwise.
       # Check if root exists to prevent AttributeError on accessing None.
       if root is None:
            return False
       # Calculate the sum of the values of the left and right child nodes.
        children_sum = (root.left.val if root.left else 0) + (root.right.val if root.right else 0)
       # Return whether the root's value is equal to the sum of its children's values.
        return root.val == children_sum
```

Time and Space Complexity The given Python function checkTree checks if the sum of the values of the left and right children of the root of a binary tree is

tree), the time complexity is constant.

structures are created), the space complexity is constant.

equal to the value of the root itself. **Time Complexity**

The time complexity of the function is 0(1). This is because the function performs a constant number of operations: it accesses

the value of the root node and the values of its immediate left and right children, and then it performs an addition and a comparison. Since these operations are constant and do not depend on the size of the input (i.e., the number of nodes in the

Space Complexity The space complexity of the function is also 0(1). The function uses a fixed amount of space: it stores two integer values (the sum of the left and right child values and the value of the root) and returns a boolean. Since the space used does not depend on the size of the input and no additional space is allocated during the execution of the function (e.g., no recursive calls or new data