112. Path Sum

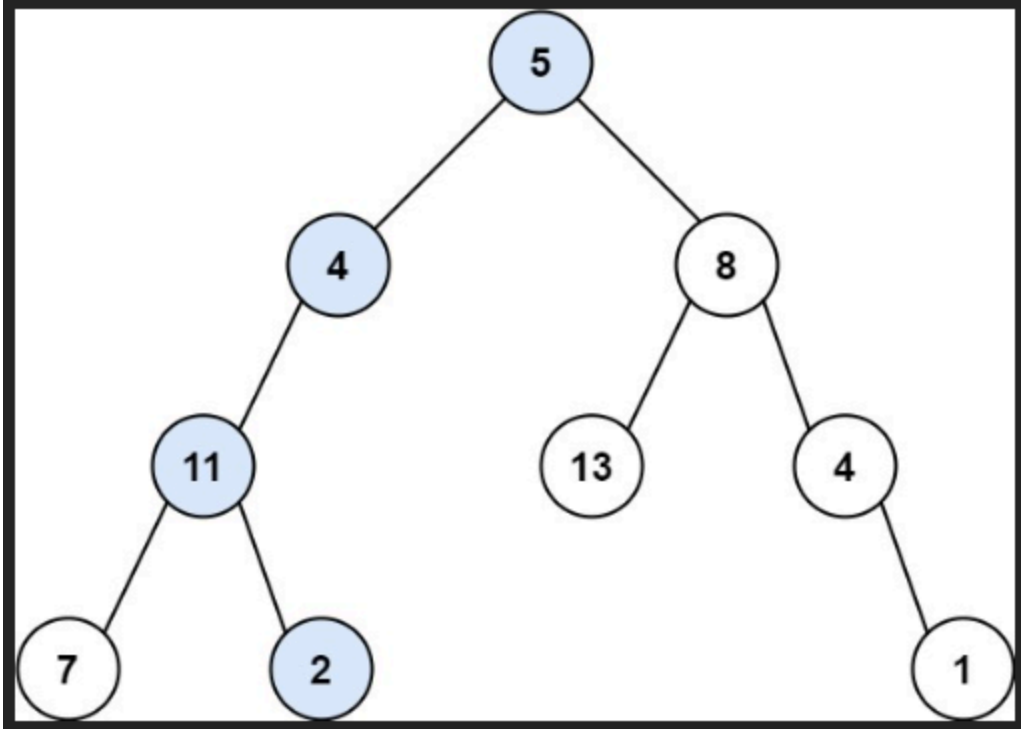
Solved

Easy | Topics | Companies

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

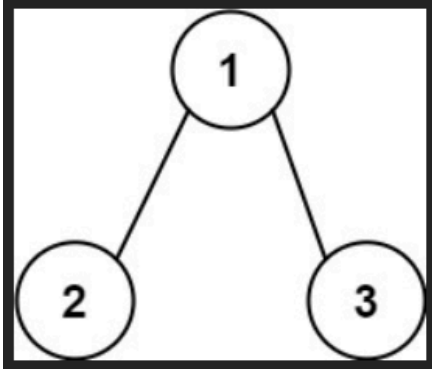
A **leaf** is a node with no children.

Example 1:




```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
```

Example 2:




```
Input: root = [1,2,3], targetSum = 5
Output: false
Explanation: There are two root-to-leaf paths in the tree:
(1 --> 2): The sum is 3.
(1 --> 3): The sum is 4.
There is no root-to-leaf path with sum = 5.
```

Example 3:

```
Input: root = [], targetSum = 0
Output: false
Explanation: Since the tree is empty, there are no root-to-leaf paths.
```

Constraints:

- The number of nodes in the tree is in the range `[0, 5000]`.
- `-1000 <= Node.val <= 1000`
- `-1000 <= targetSum <= 1000`

Python:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

from typing import Optional
```

```
class Solution:
    def hasPathSum(self, root: Optional['TreeNode'], targetSum: int) -> bool:
        if not root:
            return False

        # Check if it's a leaf node
        if not root.left and not root.right:
            return targetSum == root.val

        # Recurse on left and right subtrees
        return (self.hasPathSum(root.left, targetSum - root.val) or
                self.hasPathSum(root.right, targetSum - root.val))
```

JavaScript:

```
// Definition for a binary tree node.
function TreeNode(val, left = null, right = null) {
    this.val = val;
    this.left = left;
    this.right = right;
}

var hasPathSum = function(root, targetSum) {
    if (root === null) return false;

    // Check if we are at a leaf
    if (root.left === null && root.right === null) {
        return targetSum === root.val;
    }

    // Recursively check left and right subtrees
    return hasPathSum(root.left, targetSum - root.val) ||
           hasPathSum(root.right, targetSum - root.val);
};
```

Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
```

```
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean hasPathSum(TreeNode root, int targetSum) {
        // Base case: empty tree
        if (root == null) return false;

        // Check if it's a leaf node
        if (root.left == null && root.right == null) {
            return targetSum == root.val;
        }

        // Recursively check left and right subtrees
        int remainingSum = targetSum - root.val;
        return hasPathSum(root.left, remainingSum) || hasPathSum(root.right, remainingSum);
    }
}
```