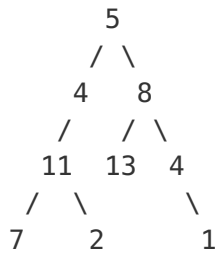
Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:
Given the below binary tree and `sum = 22`,



return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

Solution 1

The basic idea is to subtract the value of current node from sum until it reaches a leaf node and the subtraction equals 0, then we know that we got a hit. Otherwise the subtraction at the end could not be 0.

```
public class Solution {
    public boolean hasPathSum(TreeNode root, int sum) {
        if(root == null) return false;

        if(root.left == null && root.right == null && sum - root.val == 0) return
true;

        return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, su
m - root.val);
    }
}
```

written by boy27910230 original link here

Solution 2

```
bool hasPathSum(TreeNode *root, int sum) {
        if (root == NULL) return false;
        if (root->val == sum && root->left ==  NULL && root->right == NULL) retur
n true;
        return hasPathSum(root->left, sum-root->val) || hasPathSum(root->right, s
um-root->val);
    }
```

written by pankit original link here

Solution 3

In the postorder traversal, the node will be removed from the stack only when the right sub-tree has been visited.so the path will be stored in the stack. we can keep check the SUM, the length from root to leaf node. at leaf node, if SUM == sum, OK, return true. After postorder traversal, return false.

I have compared this solution with recursion solutions. In the leetcode OJ, the run time of two solutions is very near.

below is my iterator code.

```
class Solution {
public:
    bool hasPathSum(TreeNode *root, int sum) {
        stack<TreeNode *> s;
        TreeNode *pre = NULL, *cur = root;
        int SUM = 0;
        while (cur || !s.empty()) {
            while (cur) {
                s.push(cur);
                SUM += cur->val;
                cur = cur->left;
            }
            cur = s.top();
            if (cur->left == NULL && cur->right == NULL && SUM == sum) {
                return true;
            }
            if (cur->right && pre != cur->right) {
                cur = cur->right;
            } else {
                pre = cur;
                s.pop();
                SUM -= cur->val;
                cur = NULL;
            }
        }
        return false;
    }
};
```

written by SJames original link here