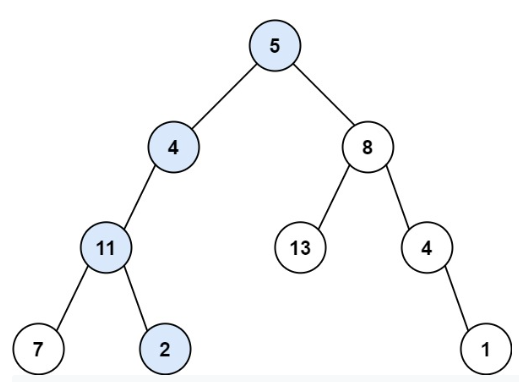
<https://leetcode.com/problems/path-sum/>

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:**

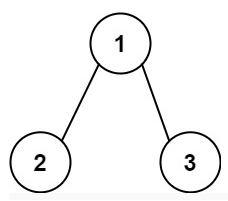


Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22

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 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

**Attempt 1: 2022-11-03**

**Solution 1:  Recursive traversal with String to find and store paths first then calculate target sum, fully based on L257.Binary Tree Paths, we can improve performance by remove find and store paths which transplant from L257 and only keep target sum calculation part for this problem only (10min)**

/\*\*

\* 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) {

List<String> result = new ArrayList<String>();

helper(result, root, targetSum, "");

return result.size() > 0;

}

private void helper(List<String> result, TreeNode root, int targetSum, String path) {

// No need target < 0, test out by input [-2,null,-3] and targetSum=-5

// if(root == null || targetSum < 0) {

if(root == null) {

return;

}

path = path.length() == 0 ? path + root.val : path + "->" + root.val;

targetSum -= root.val;

if(root.left == null && root.right == null && targetSum == 0) {

result.add(path);

return;

}

helper(result, root.left, targetSum, path);

helper(result, root.right, targetSum, path);

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Solution 2:  Recursive traversal with StringBuilder and Backtracking to find and store paths first then calculate target sum, fully based on L257.Binary Tree Paths, we can improve performance by remove find and store paths which transplant from L257 and only keep target sum calculation part for this problem only (10min)**

/\*\*

\* 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) {

List<String> result = new ArrayList<String>();

StringBuilder sb = new StringBuilder();

helper(result, root, targetSum, sb);

return result.size() > 0;

}

private void helper(List<String> result, TreeNode root, int targetSum, StringBuilder sb) {

// No need target < 0, test out by input [-2,null,-3] and targetSum=-5

// if(root == null || targetSum < 0) {

if(root == null) {

return;

}

if(sb.length() == 0) {

sb.append(root.val);

} else {

sb.append("->").append(root.val);

}

targetSum -= root.val;

if(root.left == null && root.right == null && targetSum == 0) {

result.add(sb.toString());

return;

}

int len = sb.length();

helper(result, root.left, targetSum, sb);

sb.setLength(len);

helper(result, root.right, targetSum, sb);

sb.setLength(len);

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Solution 3:  Divide and Conquer (10min)**

/\*\*

\* 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) {

return helper(root, targetSum);

}

private boolean helper(TreeNode root, int targetSum) {

// Base case

if(root == null) {

return false;

}

// Leaf node case

// Style 1

//if(root.left == null && root.right == null && targetSum == root.val) {

// return true;

//}

// Style 2

if(root.left == null && root.right == null) {

return targetSum == root.val;

}

// Divide

boolean left = helper(root.left, targetSum - root.val);

boolean right = helper(root.right, targetSum - root.val);

// Conquer

return left || right;

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/path-sum/discuss/2658019/LeetCode-The-Hard-Way-Explained-Line-By-Line>

Please check out [LeetCode The Hard Way](https://wingkwong.github.io/leetcode-the-hard-way/) for more solution explanations and tutorials.

I'll explain my solution line by line daily and you can find the full list in my [Discord](https://discord.gg/Nqm4jJcyBf).

If you like it, please give a star, watch my [Github Repository](https://github.com/wingkwong/leetcode-the-hard-way) and upvote this post.

/\*\*

\* 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;

\* }

\* }

\*/

// Time Complexity: O(N)

// Space Complexity: O(N)

class Solution {

public boolean hasPathSum(TreeNode root, int targetSum) {

// return false if the root is null

if(root == null) return false;

// if it reaches to the end and the val is equal to the sum, return true

if(root.left == null && root.right == null && root.val == targetSum) return true;

// otherwise, we traverse left node and right node with the new targetSum `targetSum - root.val`

return hasPathSum(root.left, targetSum - root.val) || hasPathSum(root.right, targetSum - root.val);

}

}

**Also there is a discuss based on Style 1 and Style 2 below**

// Style 1

//if(root.left == null && root.right == null && targetSum == root.val) {

// return true;

//}

// Style 2

if(root.left == null && root.right == null) {

return targetSum == root.val;

}

<https://leetcode.com/problems/path-sum/discuss/36378/AcceptedMy-recursive-solution-in-Java/34571>

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

B:if(root.left == null && root.right == null) return sum == root.val;

the two expression are not equivalence. when root is a leaf node in A solution, if sum - root.val == 0 got false, the code will enter next recursive. but in B, whether sum equals to root.val or not, it will return a result, the code exit, never enter next recursive

<https://leetcode.com/problems/path-sum/discuss/2658019/LeetCode-The-Hard-Way-Explained-Line-By-Line/1630311>

A small suggestion. Instead of f(!root->left && !root->right && root->val == targetSum) return true; you could do if(!root->left && !root->right) return root->val == targetSum;, because you want to return for a leave anyway, this avoids doing 2 more recursive calls, just to return then because there are no further nodes in the tree.

**Solution 4:  Iterative Preorder traversal with Two Stacks (10min,  based on L144.Binary Tree Preorder Traversal)**

/\*\*

\* 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) {

if(root == null) {

return false;

}

Stack<TreeNode> stack = new Stack<TreeNode>();

Stack<Integer> pathSum = new Stack<Integer>();

stack.push(root);

pathSum.push(root.val);

while(!stack.isEmpty()) {

TreeNode node = stack.pop();

int sum = pathSum.pop();

if(node.left == null && node.right == null) {

if(sum == targetSum) {

return true;

}

}

if(node.right != null) {

stack.push(node.right);

pathSum.push(sum + node.right.val);

}

if(node.left != null) {

stack.push(node.left);

pathSum.push(sum + node.left.val);

}

}

return false;

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Why need Two Stacks ?**

**Because one stack for node storage, another stack correspondingly used for current path sum storage, if no current path sum storage, it will be difficult to track path sum change when we pop out node for other paths, which like a 'backtrack' action, the solution is when pop out a node, we will correspondingly pop out the 'path sum' exactly when approach that node**

**Refer to**

<https://leetcode.com/problems/path-sum/discuss/36580/Java-solution-both-recursion-and-iteration>

Note: The reference code is not strictly preorder traversal, because preorder traversal suppose store right subtree instead of left subtree onto stack first.

public boolean hasPathSum(TreeNode root, int sum) {

// iteration method

if (root == null) {return false;}

Stack<TreeNode> path = new Stack<>();

Stack<Integer> sub = new Stack<>();

path.push(root);

sub.push(root.val);

while (!path.isEmpty()) {

TreeNode temp = path.pop();

int tempVal = sub.pop();

if (temp.left == null && temp.right == null) {if (tempVal == sum) return true;}

else {

if (temp.left != null) {

path.push(temp.left);

sub.push(temp.left.val + tempVal);

}

if (temp.right != null) {

path.push(temp.right);

sub.push(temp.right.val + tempVal);

}

}

}

return false;

}

**Solution 5:  Iterative Preorder traversal with One Stack (120 min, difficult to understand)**

/\*\*

\* 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) {

if(root == null) {

return false;

}

Stack<TreeNode> s = new Stack<TreeNode>();

int pathSum = 0;

TreeNode prev = null;

while(root != null || !s.isEmpty()) {

while(root != null) {

s.push(root);

pathSum += root.val;

root = root.left;

}

root = s.peek();

if(root.right != null && root.right != prev) {

root = root.right;

continue;

}

if(root.left == null && root.right == null && pathSum == targetSum) {

return true;

}

s.pop();

prev = root;

pathSum -= root.val;

root = null;

}

return false;

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/path-sum/discuss/36391/Java-iterative-solution-with-one-stack>

This solution is really brilliant.

1. setting root = null, ---> so you don't visit visited left node again.
2. if (root.right != null && root.right != prev

the part: root.right != prev --> so you don't visit the visited right node again.

public boolean hasPathSum(TreeNode root, int sum) {

Stack<TreeNode> visitedNodes = new Stack<>();

TreeNode prev = null;

while(root!=null || !visitedNodes.isEmpty()){

while(root!=null){

visitedNodes.push(root);

sum -= root.val;

prev = root;

root = root.left;

}

root = visitedNodes.peek();

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

if(root.right != null && root.right != prev){

root = root.right;

}else{

sum += root.val;

prev = visitedNodes.pop();

root = null;

}

}

return false;

}

**More detail of Solution 5 refer to L113/P9.2.Path Sum II (Refer L94.Binary Tree Inorder Traversal) Solution 3:  Iterative Inorder traversal with One Stack, same pattern but add track of path**