Previous Next

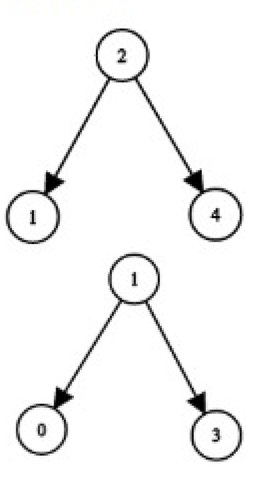
1214. Two Sum BSTs

Feb. 23, 2020 | 3K views

Average Rating: 4.86 (7 votes)

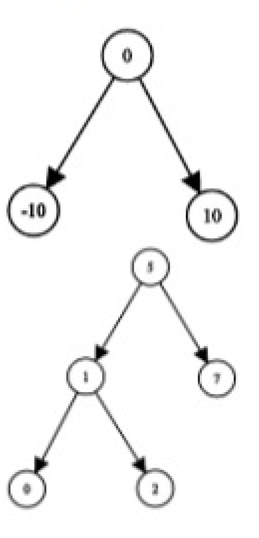
Given two binary search trees, return `True` if and only if there is a node in the first tree and a node in the second tree whose values sum up to a given integer `target`.

Example 1:



```
Input: root1 = [2,1,4], root2 = [1,0,3], target = 5
Output: true
Explanation: 2 and 3 sum up to 5.
```

Example 2:



```
Input: root1 = [0,-10,10], root2 = [5,1,7,0,2], target = 18
Output: false
```

Constraints:

- Each tree has at most `5000` nodes.
- `-10^9 <= target, node.val <= 10^9`

Solution

Overview

This problem is a combination of two other problems:

- Inorder Traversal of BST.
- Two Sum.

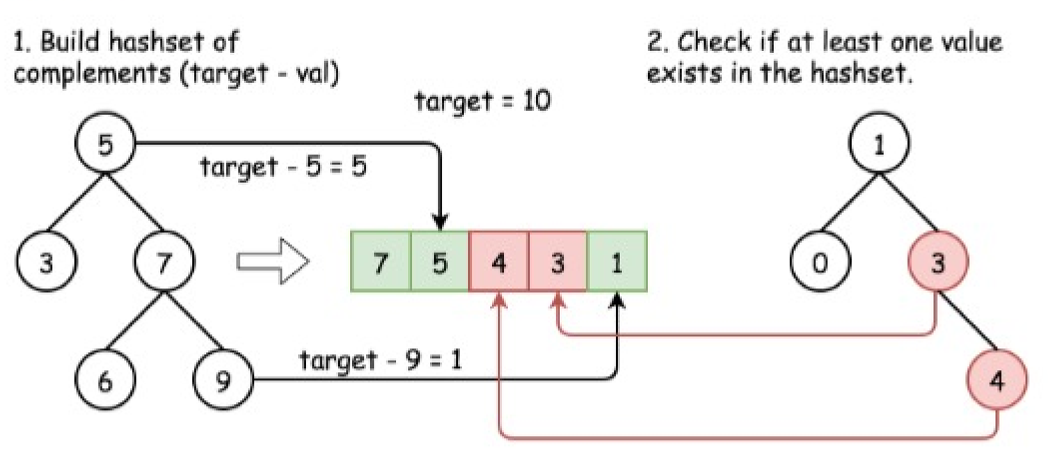
You may want to check these articles first, or to continue to read having in mind that everything will be explained a bit shorter.

Solution Pattern

> Two sum problems can be solved by using hashset of complements. Complement = target - element.

The idea is simple:

- Traverse the first tree, and store the complements (target - val) of all node values in a hashset.
- Traverse the second tree and check if any of its elements exists in the hashset. If yes - return True. If no - return False.



Prerequisites: How to traverse BST

The only remaining question is how to traverse BST. The best choice for BST is usually the DFS inorder traversal:

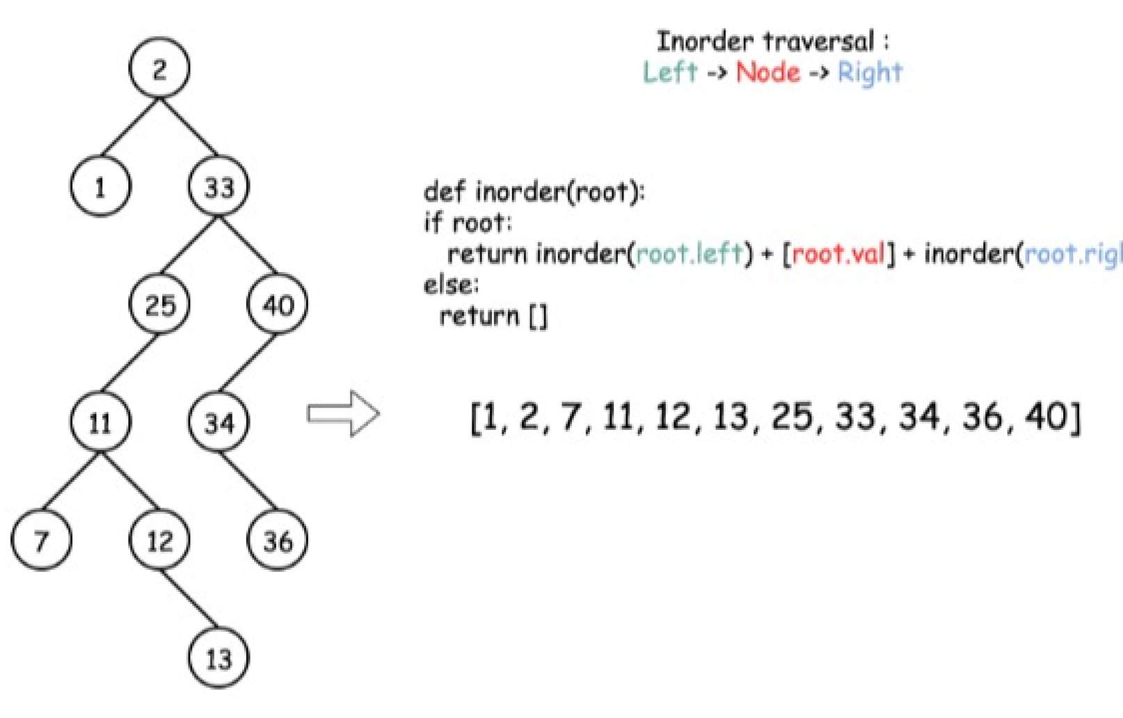
- Recursive inorder traversal is the simplest one to write, it's one liner in Python and 5-liner in Java.
- Iterative inorder traversal has the best time performance.

Here are some reminders about the DFS inorder traversal:

1. There are three DFS ways to traverse the tree: *preorder*, *postorder* and *inorder*. Please check two minutes picture explanation, if you don't remember them quite well: here is Python version and here is Java version.
2. > The result of the DFS inorder traversal of BST is an array sorted in the ascending order.
3. To compute inorder traversal follow the direction `Left -> Node -> Right`.

Java | Python | Copy

```python
def inorder(root):
    return inorder(root.left) + [root.val] + inorder(root.right) if root else []
```



Approach 1: Recursive Inorder Traversal

Let's start with the simplest solution:

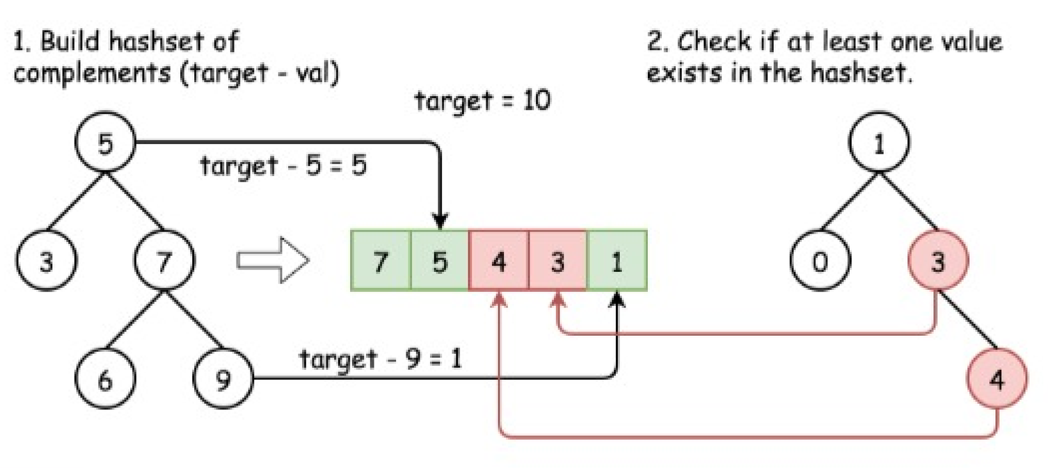
- Traverse the first tree using recursive inorder traversal. Store the complements (target - val) of all node values in a hashset.
- Traverse the second tree using recursive inorder traversal. Check if any of its elements exists in the hashset. If yes - return True. If no - return False.

Both steps are done with recursive functions:

- `in_hashset` to build hashset of complements (target - val) while traversing the first tree.
- `in_check` to check if at least one element of the second tree exists in hashset.

These functions are slightly modified versions of recursive inorder traversal presented in the overview.

Implementation



Java | Python | Copy

```python
class Solution:
    def twoSumBSTs(self, root1: TreeNode, root2: TreeNode, target: int) -> bool:
        def in_hashset(r):
            return in_hashset(r.left) | {target - r.val} | in_hashset(r.right) if r else set()

        def in_check(r):
            return in_check(r.left) or (r.val in s) or in_check(r.right) if r else False

        s = in_hashset(root1)
        return in_check(root2)
```

Complexity Analysis

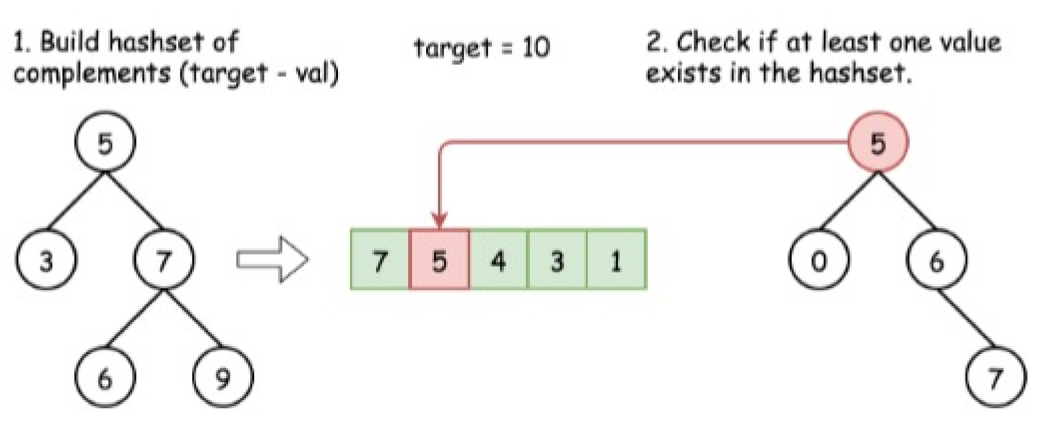
- Time complexity: $\mathcal{O}(N_1 + N_2)$, where N_1 and N_2 are the numbers of nodes in the first and the second tree respectively.
- Space complexity: $\mathcal{O}(2 \times N_1 + N_2)$, N_1 to keep the hashset and up to $N_1 + N_2$ for the recursive stacks.

Approach 2: Iterative Inorder Traversal

Intuition

The drawback of the recursive approach is that one has to traverse the entire second tree, even if it's not really needed.

For example, if `root2.val` value is already present in the hashset, there is no need to traverse further, one could stop immediately and return True.



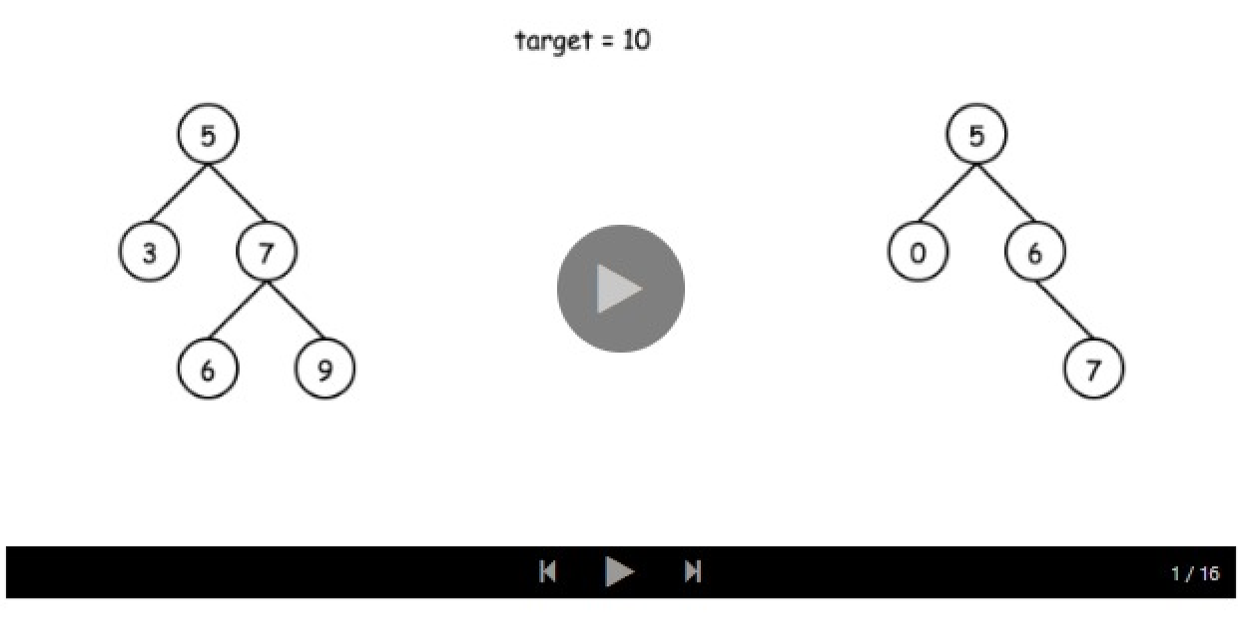
That could be implemented with the help of iterative traversal.

> Iterative inorder traversal is simple: go to the left as far as you can, pushing nodes in the stack. Then pop one node out of stack and do one step to the right. Repeat till the end of nodes in the tree and in the stack.

Algorithm

- Do iterative inorder traversal of the first tree to build hashset of complements (target - val).
- Do iterative inorder traversal of the second tree to check if at least one element of the second tree is in hashset. Stop the traversal and return True once you find such an element.
- We are here because tree2 doesn't contain any complement. Return False.

Implementation



Java | Python | Copy

```python
class Solution:
    def twoSumBSTs(self, root1: TreeNode, root2: TreeNode, target: int) -> bool:
        stack, s = [], set()
        # traverse the first tree
        # and store node complements (target - val) in hashset
        while stack or root1:
            while root1:
                stack.append(root1)
                root1 = root1.left
            root1 = stack.pop()
            s.add(target - root1.val)
            root1 = root1.right

        # traverse the second tree
        # and check if one of the values exists in hashset
        while stack or root2:
            while root2:
                stack.append(root2)
                root2 = root2.left
            root2 = stack.pop()
            if root2.val in s:
                return True
            root2 = root2.right

        return False
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N_1 + N_2)$, where N_1 and N_2 are the numbers of nodes in the first and the second tree respectively.
- Space complexity: $\mathcal{O}(N_1 + \max(N_1, N_2))$, N_1 to keep the hashset and up to $\max(N_1, N_2)$ for the stack.

Analysis written by @liaison and @andvary

Rate this article:

Previous Next

Comments: 9 Sort By

Type comment here... (Markdown is supported)

Preview Post

lsk6745 ★14 March 31, 2020 1:00 AM
In approach#1, why do we care if the traversal is In-order? HashSet doesn't preserve order, and the In-order inCheck also doesn't bring any benefits. Any comment? Thanks.
4 Share Reply

spiritson26 ★108 April 15, 2020 11:29 AM
I don't understand the point of this question when the tree is anyways flattened to the set and does a simple look up on that. I ideally expect to use some sort of BST properties to solve the question. Is there a way to solve this question in O(logN) complexity?
3 Share Reply

rajat28 ★2 April 13, 2020 7:09 AM
Wondering why this solution is not included, here is my solution without using hashset. Here I am maintaining two iterators for each tree, one scans tree left to right (inorder), other is reverse iterator from right to left (reverse inorder).

Time complexity: O(N1 + N2) to scan entire trees in worst case.
Read More
2 Share Reply

lidaivet ★43 a day ago
What difference does it make if I just keep a global hashset and traverse the first one, then traverse the second one to look for target at the second one? that would also be N1 + N2 no?
0 Share Reply

roireshef ★4 April 26, 2020 12:34 AM
Python O(N1+N2) runtime with O(max(h1, h2)) where h1, h2 are the heights of root1, root2.
Using generators:

```
def twoSumBSTs(self, root1: TreeNode, root2: TreeNode, target: int) -> bool:
    def inorder(node: TreeNode):
```

Read More
0 Share Reply

rxrxrx ★48 April 15, 2020 4:58 AM Report
Can someone help analyze the time complexity for my code below?

Very simple logic (I know it's slow): at each step, try move down BST1 or BST2, until find the target.
The worst case might be (target > max(BST1) + max(BST2)). Is it something like (logN1)2 + (logN2)2 where N1 is # of nodes in BST1, N2 is # of nodes in BST2.
Read More
0 Share Reply

lcdavid94 ★0 April 14, 2020 12:59 AM Report
Let's give a little respect to the fact that it is a BST instead of a normal binary tree...
It doesn't save any time but definitely saves space.

```
class Solution:
    def twoSumBSTs(self, root1: TreeNode, root2: TreeNode, target: int) -> bool:
```

Read More
0 Share Reply

enthiran ★1 March 7, 2020 3:56 AM
Here is my recursive solution:

```
public boolean twoSumBSTs(TreeNode root1, TreeNode root2, int target) {
    // traverse tree1 and put compliment (to avoid negative integer overflow) values in HashSet
```

Read More
0 Share Reply

soumyajitchatterjee73 ★15 February 24, 2020 12:22 PM
My O(n logn) solution n belongs to node of the trees

```
        boolean isSumFound(BST root, int target) {
            if(root == null) return false;
            boolean flag = false;
```

Read More
0 Share Reply

SHOW 1 REPLY