

653. Two Sum IV - Input is a BST

👍 579

👎 78

[Description \(/problems/two-sum-iv-input-is-a-bst/description/\)](/problems/two-sum-iv-input-is-a-bst/description/)[Hints \(/problems/two-sum-iv-input-is-a-bst/hints/\)](/problems/two-sum-iv-input-is-a-bst/hints/)[Submissions \(/problems/two-sum-iv-input-is-a-bst/submissions/\)](/problems/two-sum-iv-input-is-a-bst/submissions/)

Quick Navigation ▾

[View in Article ↗ \(/articles/two-sum-iv-input-is-a-bst/\)](/articles/two-sum-iv-input-is-a-bst/)

Notes

Solution

Approach #1 Using HashSet[Accepted]

The simplest solution will be to traverse over the whole tree and consider every possible pair of nodes to determine if they can form the required sum k . But, we can improve the process if we look at a little catch here.

If the sum of two elements $x + y$ equals k , and we already know that x exists in the given tree, we only need to check if an element y exists in the given tree, such that $y = k - x$. Based on this simple catch, we can traverse the tree in both the directions(left child and right child) at every step. We keep a track of the elements which have been found so far during the tree traversal, by putting them into a *set*.

For every current node with a value of p , we check if $k - p$ already exists in the array. If so, we can conclude that the sum k can be formed by using the two elements from the given tree. Otherwise, we put this value p into the *set*.

If even after the whole tree's traversal, no such element p can be found, the sum k can't be formed by using any two elements.

Java

Copy

```
1 public class Solution {
2     public boolean findTarget(TreeNode root, int k) {
3         Set < Integer > set = new HashSet();
4         return find(root, k, set);
5     }
6     public boolean find(TreeNode root, int k, Set < Integer > set) {
7         if (root == null)
8             return false;
9         if (set.contains(k - root.val))
10            return true;
11        set.add(root.val);
12        return find(root.left, k, set) || find(root.right, k, set);
13    }
14 }
15
```

Complexity Analysis

- Time complexity : $O(n)$. The entire tree is traversed only once in the worst case. Here, n refers to the number of nodes in the given tree.
- Space complexity : $O(n)$. The size of the *set* can grow upto n in the worst case.

Approach #2 Using BFS and HashSet [Accepted]

Algorithm

In this approach, the idea of using the *set* is the same as in the last approach. But, we can carry on the traversal in a Breadth First Search manner, which is a very common traversal method used in Trees. The way BFS is used can be summarized as given below. We start by putting the root node into a *queue*. We also

maintain a *set* similar to the last approach. Then, at every step, we do as follows:

1. Remove an element, p , from the front of the *queue*.
2. Check if the element $k - p$ already exists in the *set*. If so, return True.
3. Otherwise, add this element, p to the *set*. Further, add the right and the left child nodes of the current node to the back of the *queue*.
4. Continue steps 1. to 3. till the *queue* becomes empty.
5. Return false if the *queue* becomes empty.

By following this process, we traverse the tree on a level by level basis.

Java
Copy

```

1 public class Solution {
2     public boolean findTarget(TreeNode root, int k) {
3         Set < Integer > set = new HashSet();
4         Queue < TreeNode > queue = new LinkedList();
5         queue.add(root);
6         while (!queue.isEmpty()) {
7             if (queue.peek() != null) {
8                 TreeNode node = queue.remove();
9                 if (set.contains(k - node.val))
10                     return true;
11                 set.add(node.val);
12                 queue.add(node.right);
13                 queue.add(node.left);
14             } else
15                 queue.remove();
16         }
17         return false;
18     }
19 }
```

Complexity Analysis

- Time complexity : $O(n)$. We need to traverse over the whole tree once in the worst case. Here, n refers to the number of nodes in the given tree.
- Space complexity : $O(n)$. The size of the *set* can grow atmost upto n .

Approach #3 Using BST [Accepted]

Algorithm

In this approach, we make use of the fact that the given tree is a Binary Search Tree. Now, we know that the inorder traversal of a BST gives the nodes in ascending order. Thus, we do the inorder traversal of the given tree and put the results in a *list* which contains the nodes sorted in ascending order.

Once this is done, we make use of two pointers l and r pointing to the beginning and the end of the sorted *list*. Then, we do as follows:

1. Check if the sum of the elements pointed by l and r is equal to the required sum k . If so, return a True immediately.
2. Otherwise, if the sum of the current two elements is lesser than the required sum k , update l to point to the next element. This is done, because, we need to increase the sum of the current elements, which can only be done by increasing the smaller number.
3. Otherwise, if the sum of the current two elements is larger than the required sum k , update r to point to the previous element. This is done, because, we need to decrease the sum of the current elements, which can only be done by reducing the larger number.
4. Continue steps 1. to 3. till the left pointer l crosses the right pointer r .

5. If the two pointers cross each other, return a False value.

Note that we need not increase the larger number or reduce the smaller number in any case. This happens because, in case, a number larger than the current $list[r]$ is needed to form the required sum k , the right pointer could not have been reduced in the first place. The similar argument holds true for not reducing the smaller number as well.

Java

 Copy

Complexity Analysis

- Time complexity : $O(n)$. We need to traverse over the whole tree once to do the inorder traversal. Here, n refers to the number of nodes in the given tree.
- Space complexity : $O(n)$. The sorted $list$ will contain n elements.

Analysis written by: @vinod23 (<https://leetcode.com/vinod23>)

Comments: 18

Sort By ▼



Type comment here... (Markdown is supported)

 Preview



Post



Subhadeep2704 (/subhadeep2704) ★ 47 ⌚ June 9, 2018 11:34 AM

Isn't the first solution a DFS ?

(/subhadeep2704)

1 ▲ ▼  Share  Reply



HIDE 1 REPLY



(/lovesunmoonlight)

lovesunmoonlight (/lovesunmoonlight) ★ 0 ⌚ July 20, 2018 11:05 PM

actually it is from my opinion

0 ▲ ▼  Share  Reply