Two Sum IV - Input is a BST

Given a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Example 1:

Input:

```
    5
   / \
  3   6
 / \   \
2   4   7

Target = 9
```

Output: True

Example 2:

Input:

```
    5
   / \
  3   6
 / \   \
2   4   7

Target = 28
```

Output: False

Solution 1

Method 1.
This method also works for those who are not BSTs. The idea is to use a hashtable to save the values of the nodes in the BST. Each time when we insert the value of a new node into the hashtable, we check if the hashtable contains `k - node.val`.

Time Complexity: `O(n)`, Space Complexity: `O(n)`.

Java version:

```java
public boolean findTarget(TreeNode root, int k) {
    HashSet<Integer> set = new HashSet<>();
    return dfs(root, set, k);
}

public boolean dfs(TreeNode root, HashSet<Integer> set, int k){
    if(root == null)return false;
    if(set.contains(k - root.val))return true;
    set.add(root.val);
    return dfs(root.left, set, k) || dfs(root.right, set, k);
}
```

C++ version:

```cpp
bool findTarget(TreeNode* root, int k) {
    unordered_set<int> set;
    return dfs(root, set, k);
}

bool dfs(TreeNode* root, unordered_set<int>& set, int k){
    if(root == NULL)return false;
    if(set.count(k - root->val))return true;
    set.insert(root->val);
    return dfs(root->left, set, k) || dfs(root->right, set, k);
}
```

Method 2.
The idea is to use a sorted array to save the values of the nodes in the BST by using an inorder traversal. Then, we use two pointers which begins from the start and end of the array to find if there is a sum `k`.

Time Complexity: `O(n)`, Space Complexity: `O(n)`.

Java version:

```java
    public boolean findTarget(TreeNode root, int k) {
        List<Integer> nums = new ArrayList<>();
        inorder(root, nums);
        for(int i = 0, j = nums.size()-1; i<j;){
            if(nums.get(i) + nums.get(j) == k)return true;
            if(nums.get(i) + nums.get(j) < k)i++;
            else j--;
        }
        return false;
    }

    public void inorder(TreeNode root, List<Integer> nums){
        if(root == null)return;
        inorder(root.left, nums);
        nums.add(root.val);
        inorder(root.right, nums);
    }
```

C++ version:

```cpp
    bool findTarget(TreeNode* root, int k) {
        vector<int> nums;
        inorder(root, nums);
        for(int i = 0, j = nums.size()-1; i<j;){
            if(nums[i] + nums[j] == k)return true;
            (nums[i] + nums[j] < k)? i++ : j--;
        }
        return false;
    }

    void inorder(TreeNode* root, vector<int>& nums){
        if(root == NULL)return;
        inorder(root->left, nums);
        nums.push_back(root->val);
        inorder(root->right, nums);
    }
```

Method 3.
The idea is to use binary search method. For each node, we check if `k - node.val` exists in this BST.

Time Complexity: `O(nlogn)`, Space Complexity: `O(h)`. `h` is the height of the tree, which is `logn` at best case, and `n` at worst case.

Java version:

```
    public boolean findTarget(TreeNode root, int k) {
        return dfs(root, root,  k);
    }

    public boolean dfs(TreeNode root,  TreeNode cur, int k){
        if(cur == null)return false;
        return search(root, cur, k - cur.val) || dfs(root, cur.left, k) || dfs(root,
cur.right, k);
    }

    public boolean search(TreeNode root, TreeNode cur, int value){
        if(root == null)return false;
        return (root.val == value) && (root != cur)
            || (root.val < value) && search(root.right, cur, value)
                || (root.val > value) && search(root.left, cur, value);
    }
```

C++ version:

```
    bool findTarget(TreeNode* root, int k) {
        return dfs(root, root,  k);
    }

    bool dfs(TreeNode* root,  TreeNode* cur, int k){
        if(cur == NULL)return false;
        return search(root, cur, k - cur->val) || dfs(root, cur->left, k) || dfs(roo
t, cur->right, k);
    }

    bool search(TreeNode* root, TreeNode *cur, int value){
        if(root == NULL)return false;
        return (root->val == value) && (root != cur)
            || (root->val < value) && search(root->right, cur, value)
                || (root->val > value) && search(root->left, cur, value);
    }
```

written by Hao Cai original link here

Solution 2

O(n) time O(n) space - Inorder Vector

```
class Solution {
public:
    bool findTarget(TreeNode* root, int k) {
        vector<int> nums;
        inorder(root, nums);
        return findTargetInSortedArray(nums, k);
    }

private:
    void inorder(TreeNode* node, vector<int>& nums) {
        if (!node) return;
        inorder(node->left, nums);
        nums.push_back(node->val);
        inorder(node->right, nums);
    }

    bool findTargetInSortedArray(vector<int> a, int target) {
        for (int i = 0, j = a.size() - 1; i < j;) {
            int sum = a[i] + a[j];
            if (sum == target) {
                return true;
            }
            else if (sum < target) {
                i++;
            }
            else {
                j--;
            }
        }

        return false;
    }
};
```

O(n) time O(lg n) space - BinaryTree Iterator

```
class BSTIterator {
    stack<TreeNode*> s;
    TreeNode* node;
    bool forward;
public:
    BSTIterator(TreeNode *root, bool forward) : node(root), forward(forward) {};
    bool hasNext() {
        return node != nullptr || !s.empty();
    }
    int next() {
        while (node || !s.empty()) {
            if (node) {
                s.push(node);
                node = forward ? node->left : node->right;
            }
            else {
                node = s.top();
                s.pop();
                int nextVal = node->val;
                node = forward ? node->right : node->left;
                return nextVal;
            }
        }

        return -1;  // never reach & not necessary
    }
};
class Solution {
public:
    bool findTarget(TreeNode* root, int k) {
        if (!root) return false;
        BSTIterator l(root, true);
        BSTIterator r(root, false);
        for (int i = l.next(), j = r.next(); i < j;) {
            int sum = i + j;
            if (sum == k) {
                return true;
            }
            else if (sum < k) {
                i = l.next();
            }
            else {
                j = r.next();
            }
        }
        return false;
    }
};
```

Simplified using operator overloading

```
class BSTIterator {
    stack<TreeNode*> s;
    TreeNode* node;
    bool forward;
    int cur;
public:
    BSTIterator(TreeNode *root, bool forward) : node(root), forward(forward) { (*thi
s)++; };
    int operator*() { return cur; }
    void operator++(int) {
        while (node || !s.empty()) {
            if (node) {
                s.push(node);
                node = forward ? node->left : node->right;
            }
            else {
                node = s.top();
                s.pop();
                cur = node->val;
                node = forward ? node->right : node->left;
                break;
            }
        }
    }
};
class Solution {
public:
    bool findTarget(TreeNode* root, int k) {
        if (!root) return false;
        BSTIterator l(root, true);
        BSTIterator r(root, false);
        while (*l < *r) {
            if (*l + *r == k) return true;
            else if (*l + *r < k) l++;
            else r++;
        }
        return false;
    }
};
```

written by alexander original link here

Solution 3

In my solution, I check the target value during the search, whick makes the solution conciser.

C++:

```
class Solution {
    set<int> s;
    bool dfs(TreeNode *cur, int k) {
        if (!cur) return false;
        if (s.count(k - cur->val)) return true;
        s.insert(cur->val);
        return dfs(cur->left, k) || dfs(cur->right, k);
    }
public:
    bool findTarget(TreeNode* root, int k) {
        s.clear();
        return dfs(root, k);
    }
};
```

Python:

```
def findTarget(self, root, k):
        if not root: return False
        bfs, s = [root], set()
        for i in bfs:
            if k - i.val in s: return True
            s.add(i.val)
            if i.left: bfs.append(i.left)
            if i.right: bfs.append(i.right)
        return False
```

written by lee215 original link here