Monday, March 14, 2022 11:52 AM

Brothers From Different Roots
Easy Accuracy: 50.91% Submissions: 17574 Points: 2

Given two BSTs containing N**1** and N**2** distinct nodes respectively and given a value **x**. Your task is to complete the function **countPairs()**, that returns the count of all pairs from both the BSTs whose sum is equal to **x**.

Example 1:

Input:
BST1:
```
       5
     /   \
    3     7
   / \   / \
  2   4 6   8
```
BST2:
```
       10
     /    \
    6      15
   / \    /  \
  3   8  11   18
```
x = 16
Output:
3
Explanation:
The pairs are: **(5, 11), (6, 10)** and **(8, 8)**

Example 2:

Input:
BST1:
```
  1
   \
    3
   /
  2
```
BST2:

```
   3
  / \
 2   4
/
1
```
x = 4
Output:
3
Explanation:
The pairs are: **(2, 2), (3, 1)** and **(1, 3)**

Your Task:

You don't need to read input or print anything. Your task is to complete the function **countPairs()**, which takes 2 BST's as parameter in form of **root1** and **root2** and the integer **x**, that returns the count of all pairs from both the BSTs whose sum is equal to **x**.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

1 ≤ Number of nodes ≤ 10^5

1 ≤ Data of a node ≤ 10^6

From <https://practice.geeksforgeeks.org/problems/brothers-from-different-root/1>

HINT 1

Traverse BST 1 from smallest value to node to largest. This can be achieved with the help of iterative inorder traversal. Traverse BST 2 from largest value node to smallest

From <https://practice.geeksforgeeks.org/problems/brothers-from-different-root/1>

Count pairs from two BSTs whose sum is equal to a given value x

- Difficulty Level : Medium

- Last Updated : 14 Feb, 2022

Given two BSTs containing **n1** and **n2** distinct nodes respectively. Given a value **x**. The problem is to count all pairs from both the BSTs whose sum is equal to **x**.

Examples:

```
Input : BST 1:    5
                /   \
               3     7
              / \   / \
             2  4  6   8
BST 2:    10
                /   \
               6     15
              / \   /  \
             3  8  11  18
        x = 16

Output : 3
The pairs are:
(5, 11), (6, 10) and (8, 8)
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Method 1: For each node value **a** in BST 1, search the value **(x – a)** in BST 2. If value found then increment the **count**. For searching a value in BST, refer this post.

Time complexity: O(n1 * h2), here n1 is number of nodes in first BST and h2 is height of second BST.

Method 2: Traverse BST 1 from smallest value to node to largest. This can be achieved with the help of iterative inorder traversal. Traverse BST 2 from largest value node to smallest. This can be achieved with the help of reverse inorder traversal. Perform these two traversals simultaneously. Sum up the corresponding node's value from both the BSTs at a particular instance of traversals. If sum == x, then increment **count**. If x > sum, then move to the inorder successor of the current node of BST 1, else move to the inorder predecessor of the current node of BST 2. Perform these operations until either of the two traversals gets completed.

From <https://www.geeksforgeeks.org/count-pairs-from-two-bsts-whose-sum-is-equal-to-a-given-value-x/>

```
// Java implementation to count pairs from two
// BSTs whose sum is equal to a given  value x
import java.util.Stack;
public class GFG {

    // structure of a node of BST
    static class Node {
        int data;
        Node left, right;
        // constructor
        public Node(int data) {
            this.data = data;
            left = null;
            right = null;
        }
    }

    static Node root1;
    static Node root2;
    // function to count pairs from two BSTs
    // whose sum is equal to a given value x
    static int countPairs(Node root1, Node root2,
                                           int x)
    {
        // if either of the tree is empty
        if (root1 == null || root2 == null)
            return 0;

        // stack 'st1' used for the inorder
        // traversal of BST 1
        // stack 'st2' used for the reverse
        // inorder traversal of BST 2
        //stack<Node*> st1, st2;
        Stack<Node> st1 = new Stack<>();
        Stack<Node> st2 = new Stack<>();
        Node top1, top2;

        int count = 0;

        // the loop will break when either of two
        // traversals gets completed
        while (true) {

            // to find next node in inorder
            // traversal of BST 1
            while (root1 != null) {
                st1.push(root1);
                root1 = root1.left;
            }

            // to find next node in reverse
            // inorder traversal of BST 2
            while (root2 != null) {
                st2.push(root2);
                root2 = root2.right;
            }

            // if either gets empty then corresponding
            // tree traversal is completed
            if (st1.empty() || st2.empty())
                break;

            top1 = st1.peek();
            top2 = st2.peek();
```

```
            // if the sum of the node's is equal to 'x'
            if ((top1.data + top2.data) == x) {
                // increment count
                count++;

                // pop nodes from the respective stacks
                st1.pop();
                st2.pop();

                // insert next possible node in the
                // respective stacks
                root1 = top1.right;
                root2 = top2.left;
            }

            // move to next possible node in the
            // inorder traversal of BST 1
            else if ((top1.data + top2.data) < x) {
                st1.pop();
                root1 = top1.right;
            }

            // move to next possible node in the
            // reverse inorder traversal of BST 2
            else {
                st2.pop();
                root2 = top2.left;
            }
        }

        // required count of pairs
        return count;
    }

    // Driver program to test above
    public static void main(String args[])
    {
        // formation of BST 1
        root1 = new Node(5);       /*             5        */
        root1.left = new Node(3); /*           /   \      */
        root1.right = new Node(7); /*         3     7     */
        root1.left.left = new Node(2); /*    / \   / \    */
        root1.left.right = new Node(4); /*  2   4 6   8    */
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        // formation of BST 2
        root2 = new Node(10);        /*           10         */
        root2.left = new Node(6); /*           /   \        */
        root2.right = new Node(15); /*        6     15      */
        root2.left.left = new Node(3); /*    / \   /  \     */
        root2.left.right = new Node(8); /*  3  8  11  18    */
        root2.right.left = new Node(11);
        root2.right.right = new Node(18);
        int x = 16;
        System.out.println("Pairs = "
             + countPairs(root1, root2, x));
    }
}
// This code is contributed by Sumit Ghosh
```

Output

```
Pairs = 3
```

Time Complexity: O(n1 + n2)

Auxiliary Space: O(h1 + h2) Where h1 is height of first tree and h2 is height of second tree

Method 3 :

1. Recursive approach to solving this question.
2. Traverse the BST1 and for each node find the diff i.e. (x – root1.data) in BST2 and increment the count.

- Java
- C#
- Javascript

```
// Java implementation to count pairs from two
// BSTs whose sum is equal to a given  value x
import java.util.Stack;
public class GFG {

    // structure of a node of BST
    static class Node {
        int data;
        Node left, right;

        // constructor
        public Node(int data)
        {
            this.data = data;
            left = null;
            right = null;
        }
    }

    static Node root1;
    static Node root2;

    // function to count pairs from two BSTs
    // whose sum is equal to a given value x
    public static int pairCount = 0;
    public static void traverseTree(Node root1, Node root2,
                                    int sum)
    {
        if (root1 == null || root2 == null) {
            return;
        }
        traverseTree(root1.left, root2, sum);
```

```java
        traverseTree(root1.right, root2, sum);
        int diff = sum - root1.data;
        findPairs(root2, diff);
    }

    private static void findPairs(Node root2, int diff)
    {
        if (root2 == null) {
            return;
        }
        if (diff > root2.data) {
            findPairs(root2.right, diff);
        }
        else {
            findPairs(root2.left, diff);
        }
        if (root2.data == diff) {
            pairCount++;
        }
    }

    public static int countPairs(Node root1, Node root2,
                                 int sum)
    {
        traverseTree(root1, root2, sum);
        return pairCount;
    }

    // Driver program to test above
    public static void main(String args[])
    {
        // formation of BST 1
        root1 = new Node(5); /*             5        */
        root1.left = new Node(3); /*           /   \      */
        root1.right = new Node(7); /*         3     7     */
        root1.left.left = new Node(2); /*    / \   / \    */
        root1.left.right = new Node(4); /*  2   4 6   8 */
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        // formation of BST 2
        root2 = new Node(10); /*           10         */
        root2.left = new Node(6); /*           /   \ */
        root2.right = new Node(15); /*        6     15 */
        root2.left.left = new Node(3); /*    / \   /  \ */
        root2.left.right
            = new Node(8); /*  3  8  11  18    */
        root2.right.left = new Node(11);
        root2.right.right = new Node(18);

        int x = 16;
        System.out.println("Pairs = "
                           + countPairs(root1, root2, x));
    }
}
// This code is contributed by Sujit Panda
```

Output
```
Pairs = 3
```

From <https://www.geeksforgeeks.org/count-pairs-from-two-bsts-whose-sum-is-equal-to-a-given-value-x/>

Method 4 : Using BinarySearch Tree Iterator (A more general way of doing this)

Create two class one as *BSTIterator1* and other as *BSTIterator2*. These two class corresponds to *inOrder* and reverse *inOrder* traversal respectively.

Each class will have three methods –

- **hasNext** : will return true when traversal is not yet completed
- **next** : will move the pointer to the next node
- **peek** : will return current node in the traversal
  After creating two such classes, simple run the iterator while both have next node and find the sum. If sum == x, increment the next pointer of iterator1 and iterator2 and if sum > x ,increment the next pointer of iterator2 else increment the next pointer of iterator1 i.e when sum < x.

Below is the code in java.

- Java
- C#

```
import java.util.*;
class Node{
    int data;
      Node left, right;
       public Node(int data){
        this.data = data;
          this.left = null;
          this.right = null;
    }
}
// inorder successor iterator
class BSTIterator1{
    Stack<Node> s1 = new Stack<>();
    Node root1;
    boolean hasPeeked = false;
    public BSTIterator1(Node root){
        this.root1 = root;
    }
    public boolean hasNext(){
        if(!s1.isEmpty() || root1!=null)
            return true;
        return false;
    }
    public Node peek(){
        if(!hasNext())
            return null;
        while(root1!=null){
            s1.push(root1);
            root1 = root1.left;
            hasPeeked = true;
        }
        return s1.peek();
    }
    public int next(){
        if(!hasNext())
            return -1;
        if(!hasPeeked)
            peek();
        hasPeeked = false;
        root1 = s1.pop();
        Node temp = root1;
        root1 = root1.right;
        return temp.data;
    }
}
// inorder predecessor iterator
class BSTIterator2{
    Stack<Node> s1 = new Stack<>();
    Node root1;
    boolean hasPeeked = false;
    public BSTIterator2(Node root){
        this.root1 = root;
    }
    public boolean hasNext(){
        if(!s1.isEmpty() || root1!=null)
            return true;
        return false;
    }
    public Node peek(){
        if(!hasNext())
            return null;
        while(root1!=null){
            s1.push(root1);
            root1 = root1.right;
            hasPeeked = true;
        }
        return s1.peek();
    }
    public int next(){
        if(!hasNext())
            return -1;
        if(!hasPeeked)
            peek();
        hasPeeked = false;
        root1 = s1.pop();
        Node temp = root1;
        root1 = root1.left;
        return temp.data;
    }
}
class GfG
{
    public static int countPairs(Node r1, Node r2, int x)
    {

        BSTIterator1 it1 = new BSTIterator1(r1);
        BSTIterator2 it2 = new BSTIterator2(r2);
        int count = 0;
        while(it1.hasNext() && it2.hasNext()){
            Node n1 = it1.peek();
            Node n2 = it2.peek();
            int sum = n1.data+n2.data;
            if(sum == x){
                count++;
                it1.next();
                it2.next();
            }
            else if(sum > x){
                it2.next();
            }else{
                it1.next();
            }
        }
        return count;

    }
   // Driver program to test above
    public static void main(String args[])
    {
        Node root1, root2;
          // formation of BST 1
        root1 = new Node(5); /*                     5         */
        root1.left = new Node(3); /*             /   \       */
        root1.right = new Node(7); /*          3     7      */
        root1.left.left = new Node(2); /*     / \   / \     */
        root1.left.right = new Node(4); /*  2   4 6   8    */
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        // formation of BST 2
        root2 = new Node(10); /*                   10       */
        root2.left = new Node(6); /*             /   \     */
        root2.right = new Node(15); /*         6     15   */
        root2.left.left = new Node(3); /*     / \   /   \ */
        root2.left.right
            = new Node(8); /*  3  8  11  18     */
        root2.right.left = new Node(11);
        root2.right.right = new Node(18);

        int x = 16;
        System.out.println("Pairs = "
                           + countPairs(root1, root2, x));
    }
}
```
Output

```
Pairs = 3
```

Time Complexity: O(n1 + n2)

Auxiliary Space: O(h1 + h2) Where h1 is height of first tree and h2 is height of second tree

From <https://www.geeksforgeeks.org/count-pairs-from-two-bsts-whose-sum-is-equal-to-a-given-value-x/>