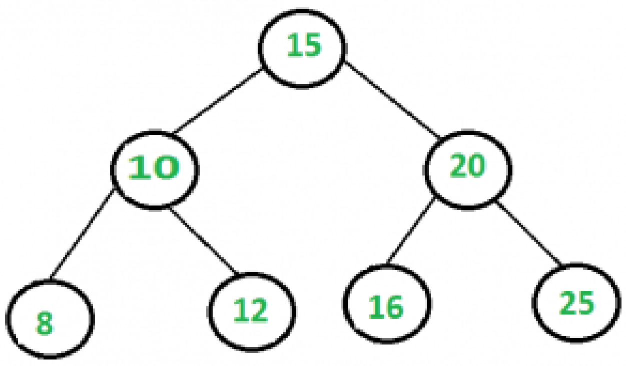
Find a pair with given sum in a Balanced BST

Given a Balanced Binary Search Tree and a target sum, write a function that returns true if there is a pair with sum equals to target sum, otherwise return false. Expected time complexity is O(n) and only O(Logn) extra space can be used. Any modification to Binary Search Tree is not allowed. Note that height of a Balanced BST is always O(Logn).



This problem is mainly extension of the previous post. Here we are not allowed to modify the BST.

The **Brute Force Solution** is to consider each pair in BST and check whether the sum equals to X. The time complexity of this solution will be O(n^2).

A **Better Solution** is to create an auxiliary array and store Inorder traversal of BST in the array. The array will be sorted as Inorder traversal of BST always produces sorted data. Once we have the Inorder traversal, we can pair in O(n) time (See this for details). This solution works in O(n) time, but requires O(n) auxiliary space.

A **space optimized solution** is discussed in previous post. The idea was to first in-place convert BST to Doubly Linked List (DLL), then find pair in sorted DLL in O(n) time. This solution takes O(n) time and O(Logn) extra space, but it modifies the given BST.

The **solution discussed below takes O(n) time, O(Logn) space and doesn't modify BST**. The idea is same as finding the pair in sorted array (See method 1 of this for details). We traverse BST in Normal Inorder and Reverse Inorder simultaneously. In reverse inorder, we start from the rightmost node which is the maximum value node. In normal inorder, we start from the left most node which is minimum value node. We add sum of current nodes in both traversals and compare this sum with given target sum. If the sum is same as target sum, we return true. If the sum is more than target sum, we move to next node in reverse inorder traversal, otherwise we move to next node in normal inorder traversal. If any of the traversals is finished without finding a pair, we return false. Following is C++ implementation of this approach.

```
/* In a balanced binary search tree isPairPresent two element which sums to
   a given value time O(n) space O(logn) */
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

// A BST node
struct node
{
    int val;
    struct node *left, *right;
};

// Stack type
struct Stack
{
    int size;
    int top;
    struct node* *array;
};

// A utility function to create a stack of given size
struct Stack* createStack(int size)
{
    struct Stack* stack =
        (struct Stack*) malloc(sizeof(struct Stack));
    stack->size = size;
    stack->top = -1;
    stack->array =
        (struct node**) malloc(stack->size * sizeof(struct node*));
    return stack;
}

// BASIC OPERATIONS OF STACK
int isFull(struct Stack* stack)
{   return stack->top - 1 == stack->size;  }

int isEmpty(struct Stack* stack)
{   return stack->top == -1;   }

void push(struct Stack* stack, struct node* node)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = node;
}

struct node* pop(struct Stack* stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top--];
}

// Returns true if a pair with target sum exists in BST, otherwise false
bool isPairPresent(struct node *root, int target)
{
    // Create two stacks. s1 is used for normal inorder traversal
    // and s2 is used for reverse inorder traversal
    struct Stack* s1 = createStack(MAX_SIZE);
    struct Stack* s2 = createStack(MAX_SIZE);

    // Note the sizes of stacks is MAX_SIZE, we can find the tree size and
    // fix stack size as O(Logn) for balanced trees like AVL and Red Black
    // tree. We have used MAX_SIZE to keep the code simple
    // done1, val1 and curr1 are used for normal inorder traversal using s1
    // done2, val2 and curr2 are used for reverse inorder traversal using s2
    bool done1 = false, done2 = false;
    int val1 = 0, val2 = 0;
    struct node *curr1 = root, *curr2 = root;

    // The loop will break when we either find a pair or one of the two
    // traversals is complete
    while (1)
    {
        // Find next node in normal Inorder traversal. See following post
        // http://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion
        while (done1 == false)
        {
            if (curr1 != NULL)
            {
                push(s1, curr1);
                curr1 = curr1->left;
            }
            else
            {
                if (isEmpty(s1))
                    done1 = 1;
                else
                {
                    curr1 = pop(s1);
                    val1 = curr1->val;
                    curr1 = curr1->right;
                    done1 = 1;
                }
            }
        }

        // Find next node in REVERSE Inorder traversal. The only
        // difference between above and below loop is, in below loop
        // right subtree is traversed before left subtree
        while (done2 == false)
        {
            if (curr2 != NULL)
            {
                push(s2, curr2);
                curr2 = curr2->right;
            }
            else
            {
                if (isEmpty(s2))
                    done2 = 1;
                else
                {
                    curr2 = pop(s2);
                    val2 = curr2->val;
                    curr2 = curr2->left;
                    done2 = 1;
                }
            }
        }

        // If we find a pair, then print the pair and return. The first
        // condition makes sure that two same values are not added
        if ((val1 != val2) && (val1 + val2) == target)
        {
            printf("\n Pair Found: %d + %d = %d\n", val1, val2, target);
            return true;
        }

        // If sum of current values is smaller, then move to next node in
        // normal inorder traversal
        else if ((val1 + val2) < target)
            done1 = false;
        // If sum of current values is greater, then move to next node in
        // reverse inorder traversal
        else if ((val1 + val2) > target)
            done2 = false;

        // If any of the inorder traversals is over, then there is no pair
        // so return false
        if (val1 >= val2)
            return false;
    }
}
// A utility function to create BST node
struct node * NewNode(int val)
{
    struct node *tmp = (struct node *)malloc(sizeof(struct node));
    tmp->val = val;
    tmp->right = tmp->left =NULL;
    return tmp;
}

// Driver program to test above functions
int main()
{
    /*
                   15
                /     \
              10      20
             / \     /  \
            8  12   16  25    */
    struct node *root =  NewNode(15);
    root->left = NewNode(10);
    root->right = NewNode(20);
    root->left->left = NewNode(8);
    root->left->right = NewNode(12);
    root->right->left = NewNode(16);
    root->right->right = NewNode(25);

    int target = 33;
    if (isPairPresent(root, target) == false)
        printf("\n No such values are found\n");

    getchar();
    return 0;
}
```

Run on IDE

Output:

```
Pair Found: 8 + 25 = 33
```

This article is compiled by Kumar and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

116 Comments Category: Binary Search Tree

Related Posts:

- Replace every element with the least greater element on its right
- In-place Convert BST into a Min-Heap
- Count inversions in an array | Set 2 (Using Self-Balancing BST)
- Print Common Nodes in Two Binary Search Trees
- Construct all possible BSTs for keys 1 to N
- K'th smallest element in BST using O(1) Extra Space
- Count BST subtrees that lie in given range
- Count BST nodes that lie in a given range

(Login to Rate and Mark)

4.2 Average Difficulty : **4.2/5.0**
Based on **50** vote(s)

🟩 Add to TODO List
🟩 Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

116 Comments GeeksforGeeks 1 Login

Recommend 7 Share Sort by Newest

Join the discussion…

Vivek Agrawal · 2 days ago
stack will contain contain n no of elements in both inorder and reverse inorder traversal .

how space complexity will come to O(log n)..

it will come to O(n)
Reply · Share ›

Vardaan Sangar · 2 days ago
O(n) Solution Space O(2logn)
http://code.geeksforgeeks.org/...
Reply · Share ›

Apar Madan · 5 days ago
O(n) solution using recursion and unordered_map.
(Iterative function commented)
https://ideone.com/GiY9Pj
Reply · Share ›

wilson · 5 days ago
great

Reply · Share ›

Jack Reacher · 8 days ago

what if the tree is not balanced ?

Reply · Share ›

Rohit Singh · 11 days ago

Can anyone explain me these lines in terms of memory representation. How memory will be given to stack and array under stack.

```
struct Stack
{
int size;
int top;
struct node* *array;
};

// A utility function to create a stack of given size
struct Stack* createStack(int size)
{
struct Stack* stack =
(struct Stack*) malloc(sizeof(struct Stack));
stack->size = size;
stack->top = -1;
stack->array =
(struct node**) malloc(stack->size * sizeof(struct node*));
return stack;
}
```

Reply · Share ›

Sunil Kumar · 15 days ago

http://code.geeksforgeeks.org/...

Reply · Share ›

Ayushi Grover · 20 days ago

Why does this take log(n) extra space ?

Function Call Stack ?

Reply · Share ›

Tailung ➦ Ayushi Grover · 16 days ago

The height of a balanced binary search tree is log n. In this iterative implementation the stacks will contain log n elements.

Reply · Share ›

Anmol Varshney · a month ago

Clean Code:

http://ideone.com/zq1bdr
3 ^ | v • Reply • Share ›

Shreya Kataria ➦ Anmol Varshney • 9 days ago
Thank you. :)
^ | v • Reply • Share ›

Ankur Lathiya ➦ Anmol Varshney • 25 days ago
thanks
^ | v • Reply • Share ›

Ankit Joshi • a month ago
Easy one
1 ^ | v • Reply • Share ›

Ratnesh Singh • a month ago
it's giving wrong answer,can anyone debug this plz...........

http://ideone.com/TYxe8r
^ | v • Reply • Share ›

kartik bhatnagar ➦ Ratnesh Singh • 25 days ago
In the if(!root) condition it should be if(root) instead. That's it!
^ | v • Reply • Share ›

Ratnesh Singh ➦ kartik bhatnagar • 25 days ago
thanqu
^ | v • Reply • Share ›

Shubham Chaudhary • a month ago
The trick for this one should be we should maintain a hash.As we know BST inorder traversal is sorted, for each node we will check the difference between required number and node->data whether it is already present in map or not . If yes return 1 else store m[root->data] as 1.If no such pair found, function will return 0. Time complexity : O(n) Space Complexity: O(log h)(approx)
P.S.:for better searching in map and c++ 11 is there , use unordered_map
Code:
https://ideone.com/GBqdNF
1 ^ | v • Reply • Share ›

devil399 ➦ Shubham Chaudhary • a month ago
How is space complexity O(log h). If no such pair found wont the map be storing a boolean value for almost all the nodes???
Please correct me if i m wrong
^ | v • Reply • Share ›

ankit verma ➦ Shubham Chaudhary · a month ago
BOOM BOOM BOOM... YOOO.. DRAUNZER GiGS.. :P
· Reply · Share ›

Shubham Chaudhary ➦ ankit verma · a month ago
aur bhai badhia :p
· Reply · Share ›

ankit verma ➦ Shubham Chaudhary · 20 days ago
mast bhai.. apne haalchaal bataao.. :P
· Reply · Share ›

Shubham Chaudhary ➦ ankit verma · 20 days ago
m badhia aur kb aa rha h clg ki clg m hi h?
· Reply · Share ›

ankit verma ➦ Shubham Chaudhary · 19 days ago
clg me hi hoon bhai... aur tum bataao... kya chal raha ..?? :P
· Reply · Share ›

Kumar Gaurav · 3 months ago
what we can also do is for each element of a BST calculate
x=abs(data given - current node data) and then search for x in a BST which will take log n time and hence a total time of nlogn with no extra space
· Reply · Share ›

Shruti · 3 months ago
Isn't this solution consumes lot of space by using two stacks?
find in order traversal and then use the sliding window approach find the pair .this will make use of only one array .
· Reply · Share ›

Aman Garg ➦ Shruti · 3 months ago
Inorder traversal will itself consume O(n) space as you have to store all the nodes in an array. This defeats the very purpose of reducing the O(n) space complexity. Using stacks is a much better option.
· Reply · Share ›

Darsh Gupta · 4 months ago
Much cleaner code without the need of exclusive use of stack
https://ideone.com/bs5fvE
3 · Reply · Share ›

Himanshu Dhiman ➦ Darsh Gupta · 4 months ago
15

/ \

10 20

/ \ / \

8 12 16 25

For sum of 27, your solution gives no as output. Whereas we have pair of nodes resulting 27.

1 ^ | ˅ • Reply • Share ›

Rahul Sarkar • 5 months ago

the simple approach is to convert it into DLL and then solve it

^ | ˅ • Reply • Share ›

bhavik gujarati ➦ Rahul Sarkar • a month ago

You are not allowed to modify the tree.

1 ^ | ˅ • Reply • Share ›

vibhu • 7 months ago

how can the above code be used to find triplet that adds to zero??

^ | ˅ • Reply • Share ›

Bala Murali Krishna ➦ vibhu • 23 days ago

We need three elements a,b,c which sum up to 0. We fix one element using inorder traversal if it is negative, say -val. Then we use above method to find two nodes which sum upto val. We have to make sure the nodes considered in the two traversals are diff from first node fixed using inorder traversal.

We have to repeat this process after fixing a -ve value. I think this would be O(n^2) approach :)

^ | ˅ • Reply • Share ›

Vinit Sinha • 9 months ago

Java code for the solution

package DATASTRUCTURE;

import java.util.*;

public class Tree {

public TreeNode head = null;

public Tree()

{

```
head = null;

}

public void add(int data)

{
```

see more

Reply • Share ›

TulsiRam • a year ago

Nice approach :)

Reply • Share ›

Karandeep Singh • a year ago

You can save more time if the root is having data >=the target sum.
Keep on traversing left, until you find a node having value < target sum.
Then call this function from that particular node..
Can save time and space, if all the nodes are having a value greater than the target sum.

1 • Reply • Share ›

rohit jain • a year ago

Better Solution in O(n) WO extra space
http://ideone.com/Om4MHu

1 • Reply • Share ›

AllergicToBitches → rohit jain • 2 months ago

Beauty!!
Here - http://ideone.com/VVAGuX
added functions for find min and max as well

2 • Reply • Share ›

Arpit gupta → AllergicToBitches • a month ago

ur code doesn't work for 8+25==33

1 • Reply • Share ›

AllergicToBitches → Arpit gupta • a month ago

replaced < with <=, works fine now
http://ideone.com/jFfoiW

Reply • Share ›

paramvir → rohit jain • 6 months ago

Implicitly you are also using logn space, that is the internal stack.

Reply • Share ›

Kshitij • a year ago

I think in function isFull(), the condition should be return stack->top + 1 == stack->size;

• Reply • Share ›

Kshitij • a year ago

How the third solution space optimized one takes O(Logn) extra space?

• Reply • Share ›

Brahma Reddy Chilakala → Kshitij • a year ago

The size of the call stack

• Reply • Share ›

Narendra • a year ago

cleaner code in c++

http://ideone.com/RlZBnn

• Reply • Share ›

Guest → Narendra • a year ago

You're not popping anything.

• Reply • Share ›

Narendra → Guest • a year ago

You are right forgot to add pop. Just added pop

http://ideone.com/AAQlx1

Thanks you for the feedback.

1 • Reply • Share ›

gamer guy → Narendra • 8 months ago

Can you please explain your code?

• Reply • Share ›

Narendra → gamer guy • 8 months ago

there is similar problem on sorted array. Take first and last numbers get the sum if sum is more than Target sum then dec last

else if less incre start else if equal return sum. I did the same on BST iterative inorder and reverse inroder to get the first and last

• Reply • Share ›

gyemeleth • a year ago

What if we were do an in-order traversal, find if (num-current_number) is in the tree with a (log n) search? Obviously this is an (n log n) solution but worth mentioning

• Reply • Share ›

Aashish Karki gyemeleth • a year ago