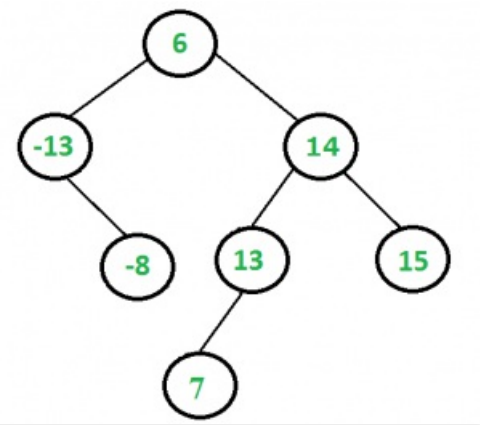
Find if there is a triplet in a Balanced BST that adds to zero

Given a Balanced Binary Search Tree (BST), write a function isTripletPresent() that returns true if there is a triplet in given BST with sum equals to 0, otherwise returns false. Expected time complexity is O(n^2) and only O(Logn) extra space can be used. You can modify given Binary Search Tree. Note that height of a Balanced BST is always O(Logn)

For example, isTripletPresent() should return true for following BST because there is a triplet with sum 0, the triplet is {-13, 6, 7}.



The Brute Force Solution is to consider each triplet in BST and check whether the sum adds upto zero. The time complexity of this solution will be O(n^3).

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 use method 2 of this post to find the triplet with sum equals to 0. This solution works in O(n^2) time, but requires O(n) auxiliary space.

Following is the solution that works in O(n^2) time and uses O(Logn) extra space:

1) Convert given BST to Doubly Linked List (DLL)

2) Now iterate through every node of DLL and if the key of node is negative, then find a pair in DLL with sum equal to key of current node multiplied by -1. To find the pair, we can use the approach used in hasArrayTwoCandidates() in method 1 of this post.

```
// A C++ program to check if there is a triplet with sum equal to 0 in
// a given BST
#include<stdio.h>

// A BST node has key, and left and right pointers
struct node
{
    int key;
    struct node *left;
    struct node *right;
};

// A function to convert given BST to Doubly Linked List. left pointer is used
// as previous pointer and right pointer is used as next pointer. The function
// sets *head to point to first and *tail to point to last node of converted DLL
void convertBSTtoDLL(node* root, node** head, node** tail)
{
    // Base case
    if (root == NULL)
        return;

    // First convert the left subtree
    if (root->left)
        convertBSTtoDLL(root->left, head, tail);

    // Then change left of current root as last node of left subtree
    root->left = *tail;

    // If tail is not NULL, then set right of tail as root, else current
```

```
    // node is head
    if (*tail)
        (*tail)->right = root;
    else
        *head = root;

    // Update tail
    *tail = root;

    // Finally, convert right subtree
    if (root->right)
        convertBSTtoDLL(root->right, head, tail);
}

// This function returns true if there is pair in DLL with sum equal
// to given sum. The algorithm is similar to hasArrayTwoCandidates()
// in method 1 of http://tinyurl.com/dy6palr
bool isPresentInDLL(node* head, node* tail, int sum)
{
    while (head != tail)
    {
        int curr = head->key + tail->key;
        if (curr == sum)
            return true;
        else if (curr > sum)
            tail = tail->left;
        else
            head = head->right;
    }
    return false;
}

// The main function that returns true if there is a 0 sum triplet in
// BST otherwise returns false
bool isTripletPresent(node *root)
{
    // Check if the given  BST is empty
    if (root == NULL)
       return false;

    // Convert given BST to doubly linked list.  head and tail store the
    // pointers to first and last nodes in DLLL
    node* head = NULL;
    node* tail = NULL;
    convertBSTtoDLL(root, &head, &tail);

    // Now iterate through every node and find if there is a pair with sum
    // equal to -1 * heaf->key where head is current node
    while ((head->right != tail) && (head->key < 0))
    {
        // If there is a pair with sum equal to  -1*head->key, then return
        // true else move forward
        if (isPresentInDLL(head->right, tail, -1*head->key))
            return true;
        else
            head = head->right;
    }

    // If we reach here, then there was no 0 sum triplet
    return false;
}

// A utility function to create a new BST node with key as given num
node* newNode(int num)
{
    node* temp = new node;
    temp->key = num;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to insert a given key to BST
node* insert(node* root, int key)
```

```
{
    if (root == NULL)
       return newNode(key);
    if (root->key > key)
       root->left = insert(root->left, key);
    else
       root->right = insert(root->right, key);
    return root;
}

// Driver program to test above functions
int main()
{
    node* root = NULL;
    root = insert(root, 6);
    root = insert(root, -13);
    root = insert(root, 14);
    root = insert(root, -8);
    root = insert(root, 15);
    root = insert(root, 13);
    root = insert(root, 7);
    if (isTripletPresent(root))
        printf("Present");
    else
        printf("Not Present");
    return 0;
}
```

Output:

```
Present
```

Note that the above solution modifies given BST.

Time Complexity: Time taken to convert BST to DLL is O(n) and time taken to find triplet in DLL is O(n^2).

Auxiliary Space: The auxiliary space is needed only for function call stack in recursive function convertBSTtoDLL(). Since given tree is balanced (height is O(Logn)), the number of functions in call stack will never be more than O(Logn).

We can also find triplet in same time and extra space without modifying the tree. See next post. The code discussed there can be used to find triplet also.