Largest BST in a Binary Tree | Set 2

Given a Binary Tree, write a function that returns the size of the largest subtree which is also a Binary Search Tree (BST). If the complete Binary Tree is BST, then return the size of the whole tree.

Examples:

```
Input:
      5
    /  \
   2    4
 /  \
1    3
Output: 3
The following subtree is the
maximum size BST subtree
   2
 /  \
1    3
Input:
       50
     /    \
  30       60
 /  \     /  \
5   20   45    70
              /  \
            65    80
Output: 5
The following subtree is the
maximum size BST subtree
      60
     /  \
   45    70
        /  \
      65    80
```

[Find the largest BST subtree in a given Binary Tree | Set 1](#)

In this post, a different O(n) solution is discussed. This solution is simpler than the solutions discussed above and works in O(n) time.

The idea is based on method 3 of [check if a binary tree is BST article](#).

A Tree is BST if following is true for every node x.

1. The largest value in left subtree (of x) is smaller than value of x.

2. The smallest value in right subtree (of x) is greater than value of x. We traverse tree in bottom up manner. For every traversed node, we return maximum and minimum values in subtree rooted with it. If any node follows above properties and size of

```
// C++ program to find largest BST in a

// Binary Tree.

#include <bits/stdc++.h>

using namespace std;


/* A binary tree node has data,

pointer to left child and a

pointer to right child */

struct Node {

	int data;

	struct Node* left;

	struct Node* right;

};


/* Helper function that allocates a new

node with the given data and NULL left

and right pointers. */

struct Node* newNode(int data)

{

	struct Node* node = new Node;

	node->data = data;

	node->left = node->right = NULL;


	return (node);

}
```

```
// Information to be returned by every
// node in bottom up traversal.
struct Info {
        // Size of subtree
        int sz;
        // Min value in subtree
        int max;
        // Max value in subtree
        int min;
        // Size of largest BST which
        // is subtree of current node
        int ans;
        // If subtree is BST
        bool isBST;
};

// Returns Information about subtree. The
// Information also includes size of largest
// subtree which is a BST.
Info largestBSTBT(Node* root)
{
        // Base cases : When tree is empty or it has
        // one child.
        if (root == NULL)
                return { 0, INT_MIN, INT_MAX, 0, true };
        if (root->left == NULL && root->right == NULL)
                return { 1, root->data, root->data, 1, true };

        // Recur for left subtree and right subtrees
```

```
Info l = largestBSTBT(root->left);
Info r = largestBSTBT(root->right);

// Create a return variable and initialize its
// size.
Info ret;
ret.sz = (1 + l.sz + r.sz);

// If whole tree rooted under current root is
// BST.
if (l.isBST && r.isBST && l.max < root->data
        && r.min > root->data) {
        ret.min = min(l.min, root->data);
        ret.max = max(r.max, root->data);

        // Update answer for tree rooted under
        // current 'root'
        ret.ans = l.ans + r.ans + 1;
        ret.isBST = true;

        return ret;
}

// If whole tree is not BST, return maximum
// of left and right subtrees
ret.ans = max(l.ans, r.ans);
ret.isBST = false;

return ret;
```

```
}

/* Driver program to test above functions*/
int main()
{
        /* Let us construct the following Tree
                60
        / \
        65 70
        /
        50 */

        struct Node* root = newNode(60);
        root->left = newNode(65);
        root->right = newNode(70);
        root->left->left = newNode(50);
        printf(" Size of the largest BST is %d\n",
                largestBSTBT(root).ans);
        return 0;
}
```

Output

```
 Size of the largest BST is 2
```

Time Complexity : O(n)
Space complexity: O(n) For call stack since using recursion