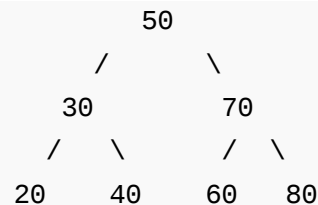
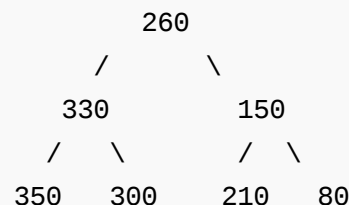


Add all greater values to every node in a given BST

Given a **Binary Search Tree** (BST), modify it so that all greater values in the given BST are added to every node. For example, consider the following BST.



The above tree should be modified to following



We strongly recommend you to minimize the browser and try this yourself first.

A **simple method** for solving this is to find sum of all greater values for every node. This method would take $O(n^2)$ time.

We can do it **using a single traversal**. The idea is to use following BST property. If we do reverse Inorder traversal of BST, we get all nodes in decreasing order. We do reverse Inorder traversal and keep track of the sum of all nodes visited so far, we add this sum to every node.

```
// C program to add all greater values in every node of BST
#include<stdio.h>
```

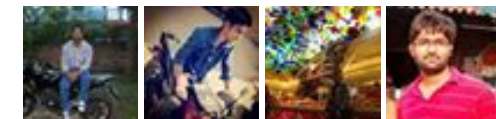
Google™ Custom Search



GeeksforGeeks



52,731 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

```
#include<stdlib.h>

struct node
{
    int data;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to add all greater values in every node
void modifyBSTUtil(struct node *root, int *sum)
{
    // Base Case
    if (root == NULL) return;

    // Recur for right subtree
    modifyBSTUtil(root->right, sum);

    // Now *sum has sum of nodes in right subtree, add
    // root->data to sum and update root->data
    *sum = *sum + root->data;
    root->data = *sum;

    // Recur for left subtree
    modifyBSTUtil(root->left, sum);
}

// A wrapper over modifyBSTUtil()
void modifyBST(struct node *root)
{
    int sum = 0;
    modifyBSTUtil(root, &sum);
}


// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
    if (root != NULL)
    {
```



**Deploy Early.
Deploy Often.**

DevOps from Rackspace:
Automation

[FIND OUT HOW ►](#)

 **rackspace**
the open cloud company

Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding “extern” keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

```

        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data <= node->data)
        node->left = insert(node->left, data);
    else
        node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
        50
       /  \
      30   70
     /  \  /  \
    20  40 60  80 */
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    modifyBST(root);

    // print inorder traversal of the modified BST
    inorder(root);

    return 0;
}

```

```
    return 0;  
}
```

Output

```
350 330 300 260 210 150 80
```

Time Complexity: $O(n)$ where n is number of nodes in the given BST.

As a side note, we can also use reverse Inorder traversal to find k th largest element in a BST.

This article is contributed by **Chandra Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Related Tpoics:

- [Print a Binary Tree in Vertical Order | Set 2 \(Hashmap based Method\)](#)
- [Print Right View of a Binary Tree](#)

695



Subscribe

Recent Comments

[affizerv](#) Your example has two 4s on row 3, that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 27 minutes ago

[RVM](#) Can someone please elaborate this Qs from above...

[Flipkart Interview | Set 6](#) · 47 minutes ago

[Vishal Gupta](#) I talked about as an Interviewer in general,...

[Software Engineering Lab, Samsung Interview | Set 2](#) · 47 minutes ago

[@meya](#) Working solution for question 2 of 4f2f round....

[Amazon Interview | Set 53 \(For SDE-1\)](#) · 1 hour ago

[sandeep void](#) rearrange(struct node *head) {...

[Given a linked list, reverse alternate nodes and append at the end](#) · 2 hours ago

[Neha](#) I think that is what it should return as, in...

[Find depth of the deepest odd level leaf node](#) · 2 hours ago

- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree



24



Tweet

3



1

Writing code in comment? Please use ideone.com and share the link here.

20 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



trojansmith · 5 months ago

How about this...

```
public class Tree {
    int data;
    Tree left;
    Tree right;
    Tree(int data, Tree left, Tree right){
        this.data = data;
        this.left = left;
        this.right = right;
    }
    public void inorder(Tree node){
        if(node.left!=null)inorder(node.left);
        System.out.println(node.data);
        if(node.right!=null)inorder(node.right);
    }
}
```

AdChoices ▶

▶ [Java Tree](#)

▶ [Binary Tree](#)

▶ [Java to C++](#)

AdChoices ▶

▶ [Node](#)

▶ [Java Array](#)

▶ [Tree Root](#)

AdChoices ▶

▶ [Java Null](#)

▶ [Java Replace](#)

▶ [Tree Swing](#)

```
}  
public int addSum(Tree node,int sum){  
    if(node.right!=null)sum=addSum(node.right,sum);
```

[see more](#)

1 ^ | v • Reply • Share ›



Guest • 6 months ago

Wow Solution

^ | v • Reply • Share ›



sriharsha • 6 months ago

please correct me if am wrong;

in the order traversal place the data in an array and then sort in $O(n \log n)$ time traversal of node..which takes $O(n)$..total $O(n \log n) + O(n) = O(n \log n)$ time

^ | v • Reply • Share ›



Roxanne → sriharsha • 5 months ago

this is not efficient. Could be modified by-
inorder traversal = $O(n)$. Store inorder nodes in a orderedhashmap.
Iterate over it and find sum of numbers greater than x eg(tree with 3 nodes)
OrderedHM(
key-2 value- 10 (8 +2)
key- 3 value - 8 (5+3)
key 5 - value-5 (right most in inorder)
) -> $O(n)$

Traverse tree again and find sum in this hashmap with corresponding |

^ | v • Reply • Share ›



Sayali • 6 months ago

Sorry for the last post.. I meant don't need to pass sum as a pointer in modifyE

^ | v • Reply • Share ›



Sayali • 6 months ago

Don't need to pass sum as a pointer in modifyBSTUtil

^ | v • Reply • Share ›



pavansrinivas • 7 months ago

Code in JAVA using reverse In-Order
Validate me

```
void addAllgreatValuesBST(){
    Stack< Node> s = new Stack<>();
    Node temp = root;
    int sum = 0;
    while(true){
        while(temp!=null){
            s.push(temp);
            temp = temp.right;
        }
        if(s.isEmpty()){
            break;
        }
        temp = s.pop();
        sum = sum+temp.i;
        temp.i = sum;
        System.out.print(temp.i+" ");
        temp = temp.left;
    }
}
```

^ | v • Reply • Share ›



Venkatesh • 7 months ago

//using Inorder traversal Right-Root-Left T(n) : O(n)

```
// typedef struct tree mytree
```

```
static int sum=0;
```

```
mytree* addAllGreater(mytree *root)
```

```
{
```

```
if(root)
```

```
{
```

```
if(root->right)
```

```
addAllGreater(root->right);
```

```
root->data+=sum;
```

```
sum+=root->data;
```

```
if(root->left)
```

```
addAllGreater(root->left);
```

```
}
```

```
}
```

^ | v • Reply • Share ›



hary → Venkatesh • 7 months ago

This will give incorrect result. Correct me If I am wrong in understanding (take a closer look)

^ | v • Reply • Share ›



Venkatesh → hary • 7 months ago

@hary : Yess , you are right sum is getting added twice

Correct steps are :

```
root->data+=sum;
```

```
sum=root->data;
```

here for each node first sum will be added to it & then resulted sum will be placed into sum.

Please correct me if i am wrong :)

Please correct me, if I am wrong .)

^ | v • Reply • Share ›



Guest • 7 months ago

// using Inorder traversal Right-Root-Left $T(n) : O(n)$

```
// typedef struct tree mytree
```

```
static int sum=0;
```

```
mytree* addAllGreater(mytree *root)
```

```
{
```

```
if(root)
```

```
{
```

```
if(root->right)
```

```
addAllGreater(root->right);
```

```
root->data+=sum;
```

```
sum+=root->data;
```

```
if(root->left)
```

```
addAllGreater(root->left);
```

```
}
```

```
}
```

^ | v • Reply • Share ›



Guest • 7 months ago



// using Inorder traversal T(n) : O(n)

// typedef struct tree mytree

static int sum=0;

mytree* temp[n];

mytree* AddAllGreater(mytree *root,int n)

{

mytree *t=root;

int i=1,l

//step1 : 1st take inorder traversal of BST

Inorder(root);

//step2 : now add all greater values to every node

[see more](#)

^ | v • Reply • Share ›



Coder011 • 8 months ago

```
void rev_inorder(struct node *root,int *key)
```

```
{
```

```
    if(root)
```

```
    {
```

```
        rev_inorder(root->right, key);
```

```
        int temp=(root->data);
```

```
        (root->data)+=( *key);
```

```
        ( *key)+=temp;
```

```

        rev_inorder(root->left, key);
    }
}

```

1 ^ | v • Reply • Share ›



Coder011 • 8 months ago

```

void rev_inorder(struct node *root,int *key)
{
    if(root)
    {
        rev_inorder(root->right,key);
        int temp=(root->data);
        (root->data)+=(*key);
        (*key)+=temp;
        rev_inorder(root->left,key);
    }
}

```

1 ^ | v • Reply • Share ›



Kuldeep Kumar • 8 months ago

```

int temp = 0;
int addAll ( node *t ) {
    if ( t ){
        addAll ( t->right ) ;
        temp += t->data ;
        t->data = temp ;
        addAll ( t->left ) ;
    }
}

```

1 ^ | v • Reply • Share ›



Guest • 8 months ago



Can also be done like this....

```
int temp=0;

int addAll ( node *t ){

    if ( t ) {
        addAll ( t->right ) ;
        temp += t->data ;
        t->data = temp ;
        addAll ( t->left ) ;
    }
}
```

1 ^ | v • Reply • Share ›



rahul • 8 months ago

@Geeksforgeeks.

It is a duplicate post.Same has already been discussed at

<http://www.geeksforgeeks.org/c...>

Rahul

4 ^ | v • Reply • Share ›



GeeksforGeeks → rahul • 8 months ago

rahul, thanks for pointing this out. We will take care of this going forward

^ | v • Reply • Share ›

Random • 8 months ago



random · 8 months ago

Yep, pretty simple Depth first search but go to the right child first instead of left

Keeping track of the sum of your right nodes.

^ | v · Reply · Share ›



dinesh.expertjobs · 8 months ago

We can do the same thing by traversing the tree in reverse-in-order and keep greater than the current node.

Initially we call the function with root, 0 as parameters.

when we are calling the function on a left subtree, we add the current nodes value

```
int modifyBst(node *root, int psum){
    if(root == NULL)
        return 0;
    int rs = modifyBst(root->right, psum);
    int data = root->data;
    root->data += rs+psum;
    int ls = modifyBst(root->left, root->data);
    return (rs+ls+data);
}
```

3 ^ | v · Reply · Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team