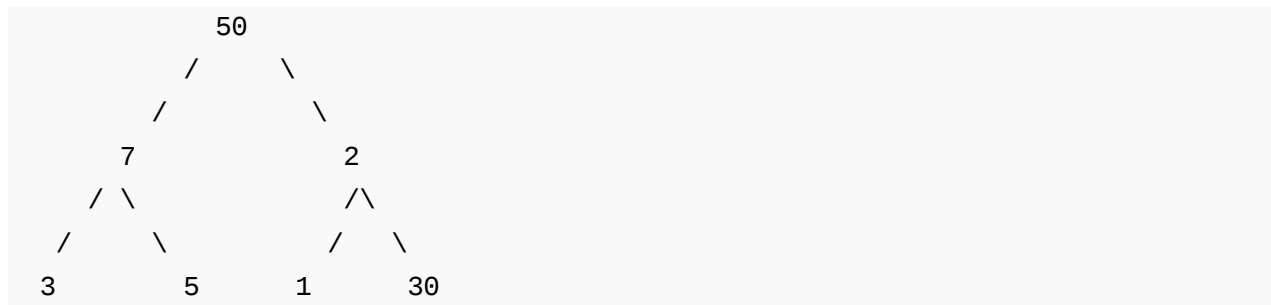


Convert an arbitrary Binary Tree to a tree that holds Children Sum Property

Question: Given an arbitrary binary tree, convert it to a binary tree that holds **Children Sum Property**. You can only increment data values in any node (You cannot change structure of tree and cannot decrement value of any node).

For example, the below tree doesn't hold the children sum property, convert it to a tree that holds the property.



Algorithm:

Traverse given tree in post order to convert it, i.e., first change left and right children to hold the children sum property then change the parent node.

Let difference between node's data and children sum be diff.

$$\text{diff} = \text{node's children sum} - \text{node's data}$$

If diff is 0 then nothing needs to be done.

If diff > 0 (node's data is smaller than node's children sum) increment the node's data by diff.

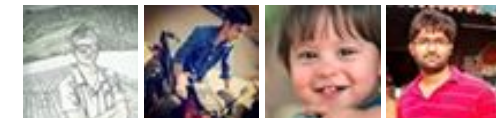
Google™ Custom Search



GeeksforGeeks



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



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

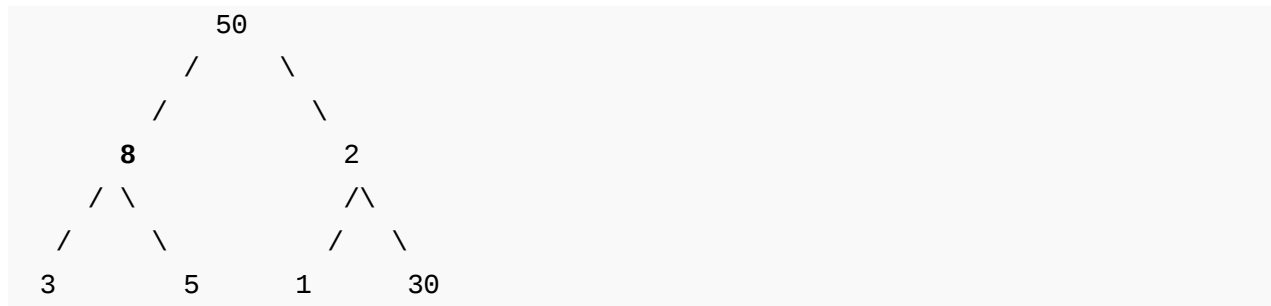
[Recursion](#)

[Geometric Algorithms](#)

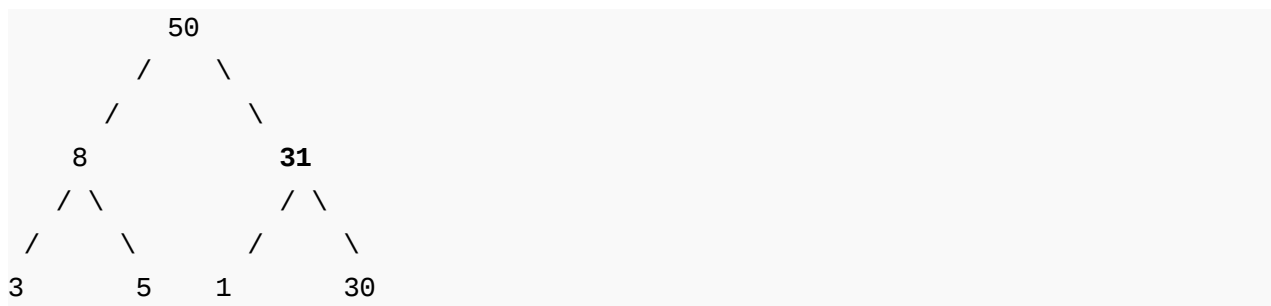
If $\text{diff} < 0$ (node's data is greater than the node's children sum) then increment one child's data. We can choose to increment either left or right child if they both are not NULL. Let us always first increment the left child. Incrementing a child changes the subtree's children sum property so we need to change left subtree also. So we recursively increment the left child. If left child is empty then we recursively call `increment()` for right child.

Let us run the algorithm for the given example.

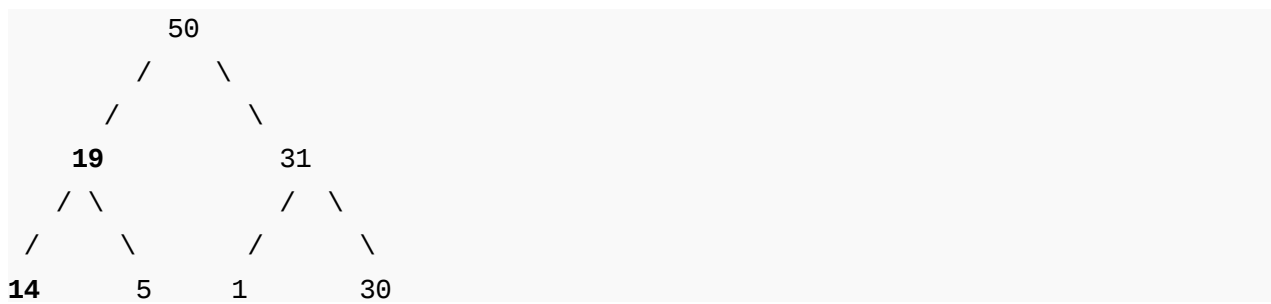
First convert the left subtree (increment 7 to 8).



Then convert the right subtree (increment 2 to 31)



Now convert the root, we have to increment left subtree for converting the root.



ITT Tech - Official Site

itt-tech.edu

Tech-Oriented Degree Programs.
Education for the Future.



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](#)

Please note the last step – we have incremented 8 to 19, and to fix the subtree we have incremented 3 to 14.

Implementation:

```
/* Program to convert an arbitrary binary tree to
   a tree that holds children sum property */

#include <stdio.h>
#include <stdlib.h>

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

/* This function is used to increment left subtree */
void increment(struct node* node, int diff);

/* Helper function that allocates a new node
   with the given data and NULL left and right
   pointers. */
struct node* newNode(int data);

/* This function changes a tree to hold children sum
   property */
void convertTree(struct node* node)
{
    int left_data = 0, right_data = 0, diff;

    /* If tree is empty or it's a leaf node then
       return true */
    if (node == NULL ||
        (node->left == NULL && node->right == NULL))
        return;
    else
    {
        /* convert left and right subtrees */
        convertTree(node->left);
        convertTree(node->right);

        /* If left child is not present then 0 is used
           as data of left child */
    }
}
```

Shouldn't
you expect
a cloud with:

**ONE-CLICK
DEPLOYMENTS**

Experience the
Managed Cloud
Difference

TRY TODAY ►

 **rackspace**
the open cloud company

Recent Comments

karthik it should have been max_wrap=

max_wrap -...

[Maximum circular subarray sum](#) · 1 minute ago

affiszerv Your example has two 4s on row 3, that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 45 minutes ago

RVM Can someone please elaborate this Qs from above...

[Flipkart Interview | Set 6](#) · 1 hour ago

Vishal Gupta I talked about as an Interviewer in general,...

[Software Engineering Lab, Samsung Interview | Set 2](#) · 1 hour 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 · 3 hours ago

AdChoices 

[▶ Binary Tree](#)

[▶ XML Tree Viewer](#)

[▶ Tree in Java](#)

AdChoices 

```

if (node->left != NULL)
    left_data = node->left->data;

/* If right child is not present then 0 is used
as data of right child */
if (node->right != NULL)
    right_data = node->right->data;

/* get the diff of node's data and children sum */
diff = left_data + right_data - node->data;

/* If node's children sum is greater than the node's data */
if (diff > 0)
    node->data = node->data + diff;

/* THIS IS TRICKY --> If node's data is greater than children sum,
then increment subtree by diff */
if (diff < 0)
    increment(node, -diff); // -diff is used to make diff positive
}

/* This function is used to increment subtree by diff */
void increment(struct node* node, int diff)
{
    /* IF left child is not NULL then increment it */
    if (node->left != NULL)
    {
        node->left->data = node->left->data + diff;

        // Recursively call to fix the descendants of node->left
        increment(node->left, diff);
    }
    else if (node->right != NULL) // Else increment right child
    {
        node->right->data = node->right->data + diff;

        // Recursively call to fix the descendants of node->right
        increment(node->right, diff);
    }
}

/* Given a binary tree, printInorder() prints out its
inorder traversal*/
void printInorder(struct node* node)
{
    if (node == NULL)

```

```

    return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(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 =
        (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return(node);
}

/* Driver program to test above functions */
int main()
{
    struct node *root = newNode(50);
    root->left = newNode(7);
    root->right = newNode(2);
    root->left->left = newNode(3);
    root->left->right = newNode(5);
    root->right->left = newNode(1);
    root->right->right = newNode(30);

    printf("\n Inorder traversal before conversion ");
    printInorder(root);

    convertTree(root);

    printf("\n Inorder traversal after conversion ");
    printInorder(root);


    getchar();
    return 0;
}

```

► [Red Black Tree](#)

► [JavaScript Tree](#)

► [Tree Structure](#)

AdChoices 

► [Tree Root](#)

► [Convert SUM](#)

► [Tree Trees](#)

Time Complexity: $O(n^2)$, Worst case complexity is for a skewed tree such that nodes are in decreasing order from root to leaf.

Please write comments if you find any bug in the above algorithm or a better way to solve the same problem.



Related Topics:

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



2



Tweet

0



0

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

Sort by Newest ▼



Join the discussion...

**Kunal Arora** • a month ago

A $O(n)$ solution is possible :-

Algorithm..

We take the Top-Bottom approach

Step-1:- Check the parents value and Sum of child's value

take, $\text{diff} = (\text{parent value} - \text{sum of child's value})$.

If $\text{diff} > 0$,

then, arbitrary choose either left and right child and add the diff to the child value

Else,

arbitrary choose either left and right child and subtract the diff to the child value

Step-2:- Recursively Call for left and right child.

Please comment and correct me if I am making some mistake or if my concept is wrong

^ | v • Reply • Share ›

**Ameet Chhatwal** • 2 months ago

A clean $O(n)$ Solution:

1. Keep adding the root value to its child value and keep traversing down the tree until you reach leaf node. (This will make it positive * (-1))
2. once a node's right and left child are visited update the node's value to left child's value + root value (This will always increase since you have already added value of root to child)

Any feedback is appreciated

^ | v • Reply • Share ›



Sumit Poddar · 6 months ago

Kindly check my JAVA implementation for the same. I believe the solution will I groups to review it and let me know in case of any issues..

```
public class TreeToCSPTree {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        Node n1 = new Node(5, new Node(2, null, null), new Node(5, null, null));  
        Node n2 = new Node(3, null, null);  
        Node n3 = new Node(7, n2, n1);  
        Node n4 = new Node(1, null, null);  
        Node n5 = new Node(50, n4, new Node(30, null, null));  
        Node n = new Node(50, n3, n5);  
        Node partn = convert(n, null, new ChangeAttr());  
        System.out.println(partn);  
    }  
}
```

[see more](#)

^ | v · [Reply](#) · [Share](#) ›



vaibhav · 8 months ago

my code if problem arises ill be thankful to critic :)

```
#include<iostream>
```

```
#include<cstdlib>
```

```
#include<limits.h>
```

```
using namespace std;
```



```
void add(int);

typedef struct node { int value;

struct node *left;

struct node *right; }mynode;

int child(mynode*root)

{if(!root)
```

[see more](#)

^ | v • Reply • Share ›



Rohan • 8 months ago

How about this approach :

```
int makeChildSumTree(struct node* node)
{
if(!node) return 0;
int left = makeChildSumTree(node->left);
int right = makeChildSumTree(node->right);
if(node->data < left + right)
node->data = left + right;
else if(node->left && node->right)
node->left->data = node->data - node->right->data;
else{
if(node->left)
node->left->data = node->data;
else if(node->right)
node->right->data = node->data;
}
```

```
return node->data,  
}
```

^ | v • Reply • Share ›



vinit • 9 months ago

@GeekforGeeks i think there can be $O(n)$ solution possible for this problem. p

Algorithm:-

step 1. bottom up part(if sum of childs weight >parent weight then give this to traversal

Note:- step 1 will ensure that parents will have value more than or equal to sur

step2.top down part (if sum of childs weight >parents weight distribute it to one

/* Paste your code here (You may **delete** these lines **if not** writing co

```
void make_Sum_Tree_Bottomup(struct Bt_Node *root){  
if(root!=NULL){  
make_Sum_Tree_Bottomup(root->left);  
make_Sum_Tree_Bottomup(root->right);  
int l=0;  
int r=0;  
// ...  
}
```

[see more](#)

^ | v • Reply • Share ›



Sanjith Sakthivel • 9 months ago

```
#include<stdlib.h>  
#include<stdio.h>  
#include<conio.h>  
struct node
```

```

\
int data;.
struct node *left,*right;.
};
struct node* insert(struct node* root, int data).
{
struct node* newnode=(struct node*)malloc(sizeof(struct node));.
newnode->data=data;.
if(root==NULL).
{.
newnode->left=NULL;.
newnode->right=NULL;.
return newnode;.
}.

```

[see more](#)

^ | v • Reply • Share ›



denial • 10 months ago

@geeksforgeeks :

Please correct the algorithm above. In condition where 'diff < 0' it is written "no children sum" , it should be "node's children sum is smaller than the node's data"

Correct if I'm wrong.

^ | v • Reply • Share ›



GeeksforGeeks → denial • 10 months ago

Please take a closer look, it is greater, not smaller.

"If node's data is *greater* than children sum, then increment subtree b

^ | v • Reply • Share ›



denial · [GeeksforGeeks](#) · 10 months ago

@geeksforgeeks :

I'm talking about algorithm not code.

diff = node's children sum - node's data

If diff is 0 then nothing needs to be done.

If diff > 0 (node's data is smaller than node's children sum) inc

If diff < 0 (node's data is smaller than the node's children sum)

^ | v · Reply · Share ›



GeeksforGeeks → denial · 10 months ago

@denial: Thanks for pointing this out. We have correcte

^ | v · Reply · Share ›



4m7u1 · a year ago

smart :D !

^ | v · Reply · Share ›



abhishek08aug · a year ago

C++ code:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

class tree_node {
private:
    int data;
```

```

        tree_node * left;
        tree_node * right;
public:
    tree_node() {
        left=NULL;
        right=NULL;
    }
    void set_data(int data) {
        this->data=data;
    }

```

[see more](#)

^ | v • Reply • Share ›



Durga Guntoju • a year ago

```

int makeltChildSumTree(struct node *root)
{
    if(root)
    {
        int l_value=makeltChildSumTree(root->left);
        int r_value=makeltChildSumTree(root->right);
        if((root->left)||(root->right))
        if(root->data!=l_value+r_value)
        root->data=l_value+r_value;
        return root->data;
    }
    else return 0;
}

```

^ | v • Reply • Share ›



Hanish • a year ago

We can optimise the code to work in $O(n)$ time as:
Logic:

If a node's data is smaller than the sum of its children, it won't ever become greater.
If a node's data is greater than the sum of its children, we will increment the left (or convert_tree(node->left)). Now when we will return to this node in postfix order, constant or increased i.e. either the node's data is equal to or less than the sum, so we call the increment function.

Thus, the code works in $O(n)$ time.

Here is the optimised function :

```
void convertTree(struct node* node)
{
    int left_data = 0, right_data = 0, diff;
    if(node == NULL ||
        (node->left == NULL && node->right == NULL))
        return;
    else
```

[see more](#)

^ | v • Reply • Share ›



Aditya • a year ago

Can it be done something like this in one function itself ?

The adjustment in the case of $\text{diff} < 0$ will be done in the deepest node first and will be adjusted accordingly in the parent.

```
[sourcecode language="C++"]
int convertTree(Node *n, int delta = 0) {
    if (n == NULL) {
        return 0;
    }
    if (n->left == NULL && n->right == NULL) {
        n->data += delta; // Fix in the deepest node.
```

```

return n->data;
}
int sum = convertTree(n->left, delta);
sum += convertTree(n->right, delta);
int diff = sum - n->data;
if (diff == 0) { return n->data; }
else if (diff > 0) {

```

[see more](#)

^ | v • Reply • Share ›



Nikin • a year ago

```

void convertTree(node *sr)
{
    if(sr == NULL)
        return;
    int lData = rData = 0;
    else
    {
        convertTree(sr->left);
        convertTree(sr->right);

        if(sr->left)
            lData = sr->left->data;
        if(sr->right)
            rData = sr->right->data;

        int diff = lData + rData - sr->data;
    }
}

```

[see more](#)



TC · a year ago
@GeekforGeeks,

I think proposed solution may fail, if root value is less than sum of its children.

Why can't we visit left and right child of a root, after we did some processing on root before.

That is I am suggesting for Top-down approach rather than bottom - up as proposed.

/* Paste your code here (You may **delete** these lines **if not** writing code)

^ | v · Reply · Share ›



vaibhavbright · 2 years ago

better answer - takes $O(n)$ time for all cases :)

Algorithm -- do a DFS traversal (mixture of pre and post) and is children_sum is reverse add this to current_index of array to_adjust.

Now we are maintaining an array to_adjust to store the amount by which node value is also using a static variable called index. So, the nodes (only the internal nodes) are visited in reverse fashion, i.e., when we reach that node we increment the index value to be given to to_adjust[index] is set after we have traversed left and right subtree of course.

The second function/stage is another DFS when I am updating the children values in reverse order type traversal.

Both traversals take $O(n)$ time. So, time complexity = $O(n)$

Code is given below.

```
int increment_node_data(node *current_node, int *to_adjust) {
    static int index = 0;
```



```
if((current_node->left == NULL) && (current_node->right == NULL))
```

[see more](#)

^ | v • Reply • Share ›



a2 • 2 years ago

Is there any problem with the following recursive code ?

```
#include<stdlib.h>

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

node* newNode(int data)
{
    node* temp = (node*)malloc(sizeof(node));
    temp->data=data;
    temp->left=temp->right=NULL;
    return temp;
}
```

[see more](#)

^ | v • Reply • Share ›



a2 → a2 • 2 years ago

Sorry , there has been some type errors in the the print function and to

^ | v • Reply • Share ›



Nitin · 2 years ago

Please tell me if I am wrong but the question doesn't say that we need to make a case then we can simply calculate the maximum value once ($O(n)$), then we can calculate the maximum value and propagate the sum of left, right child upwards (recursively).

^ | v · Reply · Share ›



GeeksforGeeks · 2 years ago

@All: We have fixed the increment function. If left subtree is not NULL, then it increments the left subtree. If right subtree is NULL, then it increments the right subtree. It recursively calls itself. The code now works for following type of trees.

```

    13
   /\
  4  6
   \
    4
  
```

^ | v · Reply · Share ›



Abhishek · 2 years ago

We can do the following recursively (Please tell me if I am wrong)

```

void convert(tree_node *node)
> convert(node->left)
> convert(node->right)
> node->data = sum(left, right)
> return;
  
```

Code given below (We can write the code for sum() where we can check if on other's data, else we return the sum of the data of both)

```

void convert(tree_node *node) {
    if(node == NULL) {
        return;
    }
  
```

```

    }
    /* Take care if its a leaf node...just return */
    if((node->left == NULL) && (node->right == NULL)) {
        return;
    }

```

[see more](#)

^ | v • Reply • Share ›



Abhishek → Abhishek • 2 years ago

Aaaaah!! got it! sorry for the wrong post.

^ | v • Reply • Share ›



Vijay • 2 years ago

I think the following need be added to increment function at the end of while loop
diff=3

```

    13
    /\
    4  6
    \
    4

```

Without this line, result after applying increment function would be:

```

    13
    /\
    7  6
    \
    4

```

```

if(node->right != NULL)
{

```

```
node->left = newNode(diff);  
return;  
}
```

^ | v • Reply • Share ›



amitp49 • 2 years ago

Does it necessary to have unique tree after this conversion?

i mean if somebody increment right child and resultant tree satisfy the Children as right answer?

^ | v • Reply • Share ›



ada • 2 years ago

Cant you just **do** a Preorder, check **if parent** satisfy child sum pro **i**:

^ | v • Reply • Share ›



Vikram • 2 years ago

/* Paste your code here (You may **delete** these lines **if not** writing c
I think it has a very simple **and** elegant solution:

```
Convert(node *head){  
    if(head is not leaf){  
        head->data = Convert(head->left) + Convert(head->right);  
    }  
    return head->data;  
}
```

^ | v · Reply · Share ›



Rahim → Vikram · a year ago

You cannot decrement value of any node, remember? This does not g

^ | v · Reply · Share ›



hari6988 · 3 years ago

consider a tree like this

7
/
3
/
2
/
1

In this case, the diff is 1. So, u should just increment the value of root's left node. In your code, u will increment both 3 and 2, which will not give u a sum tree

^ | v · Reply · Share ›



hari6988 → hari6988 · 3 years ago

sorry, it was a stupid comment... i didn't understand correctly

^ | v · Reply · Share ›



Adam → hari6988 · 3 years ago

when node data is greater than the sum of two of its children we rather increase its left child...is there is some algorithm for

^ | v · Reply · Share ›



jagannath · 3 years ago

```
#include<iostream>
using namespace std;
```

```

struct Node
{
    int data;
    Node * lchild;
    Node * rchild;
};

typedef Node * Nodeptr;
Node * create_node(int value)
{
    Node * temp = new Node();
    temp->data = value;
    temp->lchild = NULL;
    temp->rchild = NULL;
    return temp;

```

[see more](#)

^ | v • Reply • Share ›



jagannath → jagannath • 3 years ago

```

void update_node_values_with_child_sum_from_leaf_nodes(Node *r
{
    if(*root == NULL)
        return ;

    update_node_values_with_child_sum_from_leaf_nodes(&(*root)·
    update_node_values_with_child_sum_from_leaf_nodes(&(*root)·

    int root_value = (*root)->data;
    int lchild_value = ((*root)->lchild?(*root)->lchild->data:(
    int rchild_value = ((*root)->rchild?(*root)->rchild->data:(

```

```

        if(root_value data = (lchild_value + rchild_value);
    }
}

void update_node_values_with_child_sum_from_root_nodes(Node *r
{

```

[see more](#)

^ | v • Reply • Share ›



jagannath → jagannath • 3 years ago

```

void update_node_values_with_child_sum_from_leaf_nodes(
{
    if(*root == NULL)
        return ;

    update_node_values_with_child_sum_from_leaf_node
    (&(*root)->lchild);

    update_node_values_with_child_sum_from_leaf_node
    (&(*root)->rchild);

    int root_value = (*root)->data;
    int lchild_value = ((*root)->lchild?(*root)->lchild->data:0);
    int rchild_value = ((*root)->rchild?(*root)->rchild->data:0);

    if(root_value < (lchild_value + rchild_value))
    {
        (*root)->data = (lchild_value + rchild_value)
    }
}

```

[see more](#)

| • Reply • Share ›



devendra088 · 3 years ago

there is a typo at line : if diff < 0 (node's data is smaller than the node's children sum)
(node's data is larger than the node's children sum)

^ | v · Reply · Share ›



Shri · 3 years ago

In the increment function I think you should create the node with the diff after fi.
function should be like...

```
void increment(struct node* node, int diff)
{
    /* Go in depth, and fix all left children */
    while(node->left != NULL)
    {
        node->left->data = node->left->data + diff;
        node = node->left;
    }

    /* This if is for the case where left child is NULL */
    if(node->left == NULL)
    {
        node->left = newNode(diff);
        return;
    }
}
```

^ | v · Reply · Share ›



Shri → Shri · 3 years ago

Or before creating check if we haven't reached at leaf node(i.e. right ch

^ | v • Reply • Share ›



vibhav3008 • 4 years ago

```
void increment(node * root,int data)
{
    node *current=root;
    while(current!=NULL)
    {
        current->data+=data;
        if(current->left!=NULL)
        {
            current=current->left;
        }
        else
            current=current->right;
    }
    delete current;
}

void binarytochildsum(node *root)
{
```

[see more](#)

^ | v • Reply • Share ›



vibhav3008 → vibhav3008 • 4 years ago

Please comment on this solution. It retains the structure of the tree.

^ | v • Reply • Share ›



Sandeep → vibhav3008 • 4 years ago

@vibhav3008:

Could you please provide code of main() that you used to test y about your approach.

^ | v • Reply • Share ›



Cracker → Sandeep • 3 years ago

@sandeep...i think its complexity is $O(nj)$..why r saying

^ | v • Reply • Share ›



gauravs • 4 years ago

Why not Increment the value of the right node if left node is not present instead

^ | v • Reply • Share ›



coderyogi • 4 years ago

This algorithm does not take into account if different nodes have same values. node to a given tree in order to satisfy the children sum property is practical, ir increment the right subtree if the left one is NULL.

^ | v • Reply • Share ›



Harsh → coderyogi • 4 years ago

Yes, i agree.

Could anyone provide an algorithm where the structure of the tree is not

^ | v • Reply • Share ›



Jyothi • 4 years ago

Sorry, Missed the statement "you can only increment data values in any node"

Thanks

^ | v • Reply • Share ›



GeeksforGeeks • 4 years ago

@Jyothi: Please read the question statement carefully. The question says you

node. So in the 3rd step, we cannot decrement root's value from 30 to $8 + 31$.
incrementing the left subtree in such cases, so we have incremented the left s

Now, the question is - why are we incrementing left subtree not just left child?
incrementing the left child makes left subtree violate children sum property.

The program works for trees with more than two levels. Let us know the tree f

^ | v • Reply • Share ›



Jyothi • 4 years ago

In the 3rd step why are we changing the left subtree values instead of changin
of summing $8+31$ directly why are we changing the values $8 \rightarrow 19, 3 \rightarrow 14$.

And why didnt you follow the same steps for 1 and 2. If the tree level is more th

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

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