GeeksforGeeks
A computer science portal for geeks
IDE Q&A GeeksQuiz

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.

```
             50
           /     \
         /         \
       7             2
     / \             /\
   /     \          /   \
  3        5      1      30
```

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.

```
     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) increment the node's data by diff.

If 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).

```
             50
           /     \
         /         \
       8            2
      / \          /\
     /    \       /   \
   3        5    1     30
```

Then convert the right subtree (increment 2 to 31)

```
             50
           /    \
         /        \
       8            31
      / \          / \
    /     \      /     \
  3        5   1        30
```

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

```
             50
           /    \
         /        \
       19           31
      / \          /  \
    /     \      /      \
  14       5    1        30
```

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 aribitary 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 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 */
    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;
}
```

Run on IDE

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.

148 Comments Category: Trees

Related Posts:

- Find Count of Single Valued Subtrees
- Check if a given array can represent Preorder Traversal of Binary Search Tree

- Mirror of n-ary Tree
- Succinct Encoding of Binary Tree
- Construct Binary Tree from given Parent Array representation
- Symmetric Tree (Mirror Image of itself)
- Find Minimum Depth of a Binary Tree
- Maximum Path Sum in a Binary Tree

0 Average Difficulty : **0/5.0**
No votes yet.

(Login to Rate)

Like Share 5 people like this. Be the first of your friends.

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

148 Comments **GeeksforGeeks** 1 **Login**

Recommend 4 **Share** **Sort by Newest**

Join the discussion…

Saurabh Vats • a month ago

Clean and short solution: http://ideone.com/hsT1FD

1 • Reply • Share ›

Victor • 2 months ago

```
private static void childSumProperty(Node root) {
if(root == null)
return;
childSumProperty(root.left);
childSumProperty(root.right);
if(root.left == null && root.right == null)
return;
int childSum = (root.left != null ? root.left.data : 0) + (root.right != null ? root.right.data : 0);
if(root.data < childSum) {
root.data = childSum;
return;
} else if(root.data > childSum){
if(root.left.data < root.right.data) {
root.left.data = root.data - root.right.data;
childSumProperty(root.left);
} else {
```

```
root.right.data = root.data- root.left.data;
childSumProperty(root.right);
}
}
}
```

^ | ⌄ • Reply • Share ›

Amit • 3 months ago

simple method

```
node* convert(node* root){

if(root==NULL)

return root;

if(root->left==NULL&& root->right==NULL)

return root;

int ll=0,rr=0;

node* l=convert(root->left);

node* r = convert(root->right);

if(l!=NULL)ll=l->data;

if(r!=NULL)rr=r->data;

root->data = ll+rr;

return root;

}
```

^ | ⌄ • Reply • Share ›

Ajcoo • 4 months ago

How is the time complexity coming to be O(n^2) ?

^ | ⌄ • Reply • Share ›

Anurag S ➦ Ajcoo • 3 months ago

Imagine a case where the tree is skewed and the child node is greater than parent at each level.
As per the algorithm, you touch each node at least once. So 1) O(n) and while operating on each node if you have to call increment() function each time, then the running time will shoot up to O(n^2).

• Reply • Share ›

Shashi Jey • 4 months ago

https://ideone.com/ZvrDKV

• Reply • Share ›

atishya jain • 4 months ago

here we can find the maximum value key in a tree and make the leaf nodes equal to that value.we keep on moving upwards merging them

• Reply • Share ›

codelearner • 4 months ago

in case when diff <0 , cannot we just decrease the node's data by diff and not calling increment to keep it simple https://ideone.com/31kVLA
this is my code please have a look at it and tell me if there would be some problems with the approach

• Reply • Share ›

Jyoti Prakash codelearner • 4 months ago

In question it is said that , we can't decrease the value of any node :)

• Reply • Share ›

codelearner Jyoti Prakash • 4 months ago

Oh . ok must have missed that line. Thanks

• Reply • Share ›

siddharth rajpal • 4 months ago

can you tell what is wrong with this approach ->
we have a bottom up approach.
when current->left==NULL and current->right==NULL - do nothing.
and then move up.
when we get to a node we see its two or one child, we add them up. If the current node is greater than the sum then do nothing otherwise change the current data to the sum.
and so on till we reach the root.
now all the parents are at least the sum of their children.
we now see if the parent> sum of it's children. if it is then we increment the left child (if it is present and the right child otherwise) by the difference.
and hence the solution is in O(n)

1 • Reply • Share ›

Rachit Nagdev • 4 months ago

scince convertTree function will always increment the Tree node's value we can use convertTree function inplace of increment function with a little change:

```
-if diff < 0 then
   -if left child exists
        -increment left childs data by -diff
        -call convertTree on left child
-else
   -if right child exists
        -increment left childs data by -diff
        -call convertTree on right child
```

link to working code: ideone

Reply • Share ›

Dman • 4 months ago

different approach. Only one traversal required. No need to again traverse a subtree rooted with a node.
https://ideone.com/ghUZl7

1 • Reply • Share ›

Manohar Tn Dman • 4 months ago

But what do you think is the time complexity in your prog ?

Reply • Share ›

Dipankar Bhardwaj • 5 months ago

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

Reply • Share ›

Lokesh • 5 months ago

In case if decrement of node value is allowed.
http://code.geeksforgeeks.org/...

Reply • Share ›

Ashish Singh • 5 months ago

A much simpler one !!
http://ideone.com/tF8faE

Reply • Share ›

Abhimanyu Mittal Ashish Singh • 4 months ago

In the question it is mentioned that you cannot decrement the value of any node. In your solution, the value of the node will be decremented in case the 'd' turns out to be negative.

Reply • Share ›

dev21 • 5 months ago

what if I find the maximum element and update all leaf nodes to that value. use these values to update the non leaf nodes.

• Reply • Share ›

Vijay Kumar Attri • 5 months ago

Here is another recursive solution:

if the value of root is less than the sum of l and r child, then simple update.

else

call the function to recrsively correct the child with smaller data.(after updating the data of small child.

```
//few corner cases have been handled ie when rl or rr is zero because of
//root->right or root->left being NULL
struct node *csum_con(node *root){
if(root==NULL)
return NULL;
if(root->left==NULL && root->right==NULL)
return root;
root->left=csum_con(root->left);
root->right=csum_con(root->right);
```

see more

• Reply • Share ›

Vijay Kumar Attri ↪ Vijay Kumar Attri • 5 months ago

-=""> is actually ->. Don't know why it changed.

• Reply • Share ›

light • 5 months ago

Does this solution work for negative values in the binary tree ?

• Reply • Share ›

himanshu bindal • 5 months ago

this is O(n) solution . let me know if i'm wrong

```
void convertTree(node *root){

if(!root || ( !root->left && !root->right ) ) return ;

int diff = root->data - ( root->left ? root->left->data : 0 ) - ( root->right ? root->right->data : 0
) ;
```

```
if( diff>0 ){

if(root->left)

root->left->data += diff;

else

root->right->data += diff ;

}
```

see more

1 • Reply • Share ›

madhukar himanshu bindal • 2 months ago

this won't work.
What happens if diff < 0 ?
You should update the root value to the sum of the children which is totally skipped

• Reply • Share ›

Eknoor • 5 months ago

A little less code:

```
int sumTree(node *r)
{
if(!r)
return 0;
if(!r->lchild && !r->rchild)
return r->key;
int s = sumTree(r->lchild) + sumTree(r->rchild);
if(s>=(r->key))
r->key=s;
else
{
node *t = r->lchild?(r->lchild):(r->rchild);
t->key += (r->key)-s;
sumTree2(t);
}
return r->key;
}
```

• Reply • Share ›

Monica Shankar • 5 months ago

shouldn't the above code be O(NlogN) worst case because for each node where parent greater than child , you must effectively propagate the change in a dfs manner till the leaf in that branching.

• Reply • Share ›

Hitesh Saini Monica Shankar • 5 months ago

take an skewed tree as explain in above time complexity explanation you will definitely get your answer

• Reply • Share ›

Sravan Kumar Monica Shankar • 5 months ago

In worst case, the height of a tree would be O(N) not O(logN). U are thinking right "you must effectively propagate the change in a dfs manner till the leaf in that branching" which means for every node you go to the leaf that is at n-1 distance below it. So for all nodes O(n*n).

• Reply • Share ›

Sravan Kumar • 5 months ago

http://ideone.com/TY7osD

• Reply • Share ›

Hrishikesh Goyal • 5 months ago

O(n) solution ideone link http://ideone.com/73qgSk

• Reply • Share ›

Monica Shankar Hrishikesh Goyal • 5 months ago

Nice but it stills uses the same approach that is start from bottom, if parent is greater, propagate that difference to its children.

1 • Reply • Share ›

Hrishikesh Goyal Monica Shankar • 4 months ago

hey, no its different from what was given in the solution. here although the tree going to form will be different from the gfg solution.Actually in this problem more than one trees are possible satisfying children sum property. method takes O(n) time.

• Reply • Share ›

geek_13 • 6 months ago

Solved it in O(n)
First checking if sum < root,
increment left(if exists),else increment right
call the function on left and right subtree
when in reach back at the node,
again check if sum>root

root->data=sum;

Link of running code :: http://ideone.com/JrFQ7W
Please let me know if I am wrong

6 • Reply • Share ›

Siya geek_13 • 5 months ago
You have used very good approach. According to question we can only add so in bottom up don't assign values instead add the difference(line no 74). Please all check this solution as I could not find any wrong test case.
If they will ask minimum addition to convert to tree satisfying children sum property then this solution will not work.

1 • Reply • Share ›

DS+Algo Siya • 5 months ago
Why would it not work in minimum addition cond.?

• Reply • Share ›

Siya DS+Algo • 5 months ago
Every time i see addition my mind see overflow. What if the node value is large then this approach may give overflow. Its just a case. If numbers are small then this approach is very good.

• Reply • Share ›

Mr. Lazy DS+Algo • 5 months ago
What she mean to say is that although the resulting tree will satisfy the csum property but the overall sum of all the nodes of the tree will be greater obtained using this approach, as we before processing left or right subtree, make sure that later after recurring, the root's (parent node) value will always be smaller or equal to the values of child nodes (as we added the difference to left and right child already) so that we just need to add the difference to the root. However, the tree if asked to convert to hold csum by adding minimal difference to the nodes than solution given by G4G excels here. you can see the difference in the resulting trees using these two approaches. I hope you got your point! :)

3 • Reply • Share ›

DS+Algo Mr. Lazy • 5 months ago
I thought she is saying that number of changes made would be more in this approach

• Reply • Share ›

Jerry Goyal • 6 months ago

works fine and more optimised. O(n^2) in case of skewed tree.

```
void convert(struct node* root)
{
if(!root)
return;  // satisfies children sum property
if(!root - > left&&!root - > right)
return;          // satisfies children sum property
convert(root - > left);          // postorder traversal
convert(root - > right);
if(!root - > left){               //  only right child
        if(root - > right - > data < root - > data){     // root is bigger
         root - > right - > data = root - > data;        // set child data
         convert(root);          // also fix the subtree concurrently.
         }
        else root - > data=root - > right - > data;  // else root is smaller or equal
}
else if(!root - > right){                                  // only left child
```

see more

• Reply • Share ›

Jerry Goyal • 7 months ago

here's much efficient code with single function. O(n^2) only in case of decremented skewed tree!!

```
void modify(node* root){
        if(!root) return;
        modify(root->left);
        modify(root->right);

        int lc=0,rc=0;
        if(root->left) lc=root->left->data;
        if(root->right) rc=root->right->data;

        if(root->data < (lc+rc))
        root->data=lc+rc;

        else if(root->data > (lc+rc)){
                if(root->left){
                        root->left->data=root->data-rc;
```

see more

2 • Reply • Share ›

Aryan Parker • 7 months ago

Can someone pls provide a link of an efficient and clean O(N) algorithm for the above problem.

• Reply • Share ›

surbhijain93 • 8 months ago

(Just for my own reference:)

```
void ii(struct node *node)

{

if(node==NULL)

return;

if(node->left==NULL && node->right==NULL)

return;

if(node->left)

{

(node->left->data)+=node->data-(node->left->data+node->right->data);

ii(node->left);
```

see more

• Reply • Share ›

The_Geek • 9 months ago

If we don't need sum of nodes minimum, then we can simply make root->data=root->left->data +root->right_data in post-order, i.e. in bottom-up fashion.
Time complexity: O(n)
Why sum peple in below answers have added root value to child values in top-down manner and then again in bottom-up fashion they followed my approach.
Can anybody explain?

• Reply • Share ›

Jitendra Singh The_Geek • 9 months ago

It will fail when root->data > root->left->data +root->right_data. We cannot decrease value of any node

2 • Reply • Share ›

The_Geek Jitendra Singh • 9 months ago

Thanks. Got it!

Reply • Share ›

chirag • 9 months ago

below is a simple and short O(n) function for this problem. find any bug,if there is so.
void convertToChildSum(node* root)

{

if(!root||(root->left==NULL&&root->right==NULL))

return ;

int child_sum=0;

if(root->left)

child_sum+=root->left->data;

if(root->right)

child_sum+=root->right->data;

if(root->data>child_sum)

see more

1 • Reply • Share ›

Ashish Jaiswal ➦ chirag • 8 months ago

Your code is decreasing the value of root node in case where sum of left and right child value is more than root node. And we can not decrease the value of any node as by problem statement....please have a look at your code..
for Exp: root(8)->left=3 & root(8)->right=4;
let me know if i am wrong...

2 • Reply • Share ›

shantanu ➦ chirag • 9 months ago

your code looks alright but i think you missed the case when there's only one child of a node,right ?

Reply • Share ›

chirag ➦ shantanu • 9 months ago

thx shantanu! i think addition of a single if condition sort out this problem :)

void convertToChildSum(node* root)

{

{

if(!root||(root->left==NULL&&root->right==NULL))

return ;

int child_sum=0;

if(root->left)

child_sum+=root->left->data;

if(root->right)

child_sum+=root->right->data;

if(root->data>child_sum)

see more

1 • Reply • Share ›

neer1304 • 10 months ago

Using PostOrder Traversal
http://ideone.com/N2TUlv

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy

@geeksforgeeks, Some rights reserved Contact Us! About Us! Advertise with us!