How to determine if a binary tree is height-balanced?

A tree where no leaf is much farther away from the root than any other leaf. Different balancing schemes allow different definitions of “much farther” and different amounts of work to keep them balanced.

Consider a height-balancing scheme where following conditions should be checked to determine if a binary tree is balanced.
An empty tree is height-balanced. A non-empty binary tree T is balanced if:
1) Left subtree of T is balanced
2) Right subtree of T is balanced
3) The difference between heights of left subtree and right subtree is not more than 1.

The above height-balancing scheme is used in AVL trees. The diagram below shows two trees, one of them is height-balanced and other is not. The second tree is not height-balanced because height of left subtree is 2 more than height of right subtree.

To check if a tree is height-balanced, get the height of left and right subtrees. Return true if difference between heights is not more than 1 and left and right subtrees are balanced, otherwise return false.

C

```
/* program to check if a tree is height-balanced or not */
#include<stdio.h>
#include<stdlib.h>
#define bool int

/* 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;
};
```

```
/* Returns the height of a binary tree */
int height(struct node* node);

/* Returns true if binary tree with root as root is height-balanced */
bool isBalanced(struct node *root)
{
   int lh; /* for height of left subtree */
   int rh; /* for height of right subtree */

   /* If tree is empty then return true */
   if(root == NULL)
    return 1;

   /* Get the height of left and right sub trees */
   lh = height(root->left);
   rh = height(root->right);

   if( abs(lh-rh) <= 1 &&
       isBalanced(root->left) &&
       isBalanced(root->right))
     return 1;

   /* If we reach here then tree is not height-balanced */
   return 0;
}

/* UTILITY FUNCTIONS TO TEST isBalanced() FUNCTION */

/* returns maximum of two integers */
int max(int a, int b)
{
  return (a >= b)? a: b;
}

/*  The function Compute the "height" of a tree. Height is the
    number of nodes along the longest path from the root node
    down to the farthest leaf node.*/
int height(struct node* node)
{
   /* base case tree is empty */
   if(node == NULL)
       return 0;

   /* If tree is not empty then height = 1 + max of left
      height and right heights */
   return 1 + max(height(node->left), height(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);
}

int main()
{
    struct node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->left->left->left = newNode(8);

    if(isBalanced(root))
      printf("Tree is balanced");
    else
      printf("Tree is not balanced");

    getchar();
    return 0;
}
```

Java

```
// Java program to determine if binary tree is height balanced

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;
```

```
/* Returns true if binary tree with root as root is height-balanced */
boolean isBalanced(Node node) {
    int lh; /* for height of left subtree */

    int rh; /* for height of right subtree */

    /* If tree is empty then return true */
    if (node == null) {
        return true;
    }

    /* Get the height of left and right sub trees */
    lh = height(node.left);
    rh = height(node.right);

    if (Math.abs(lh - rh) <= 1
            && isBalanced(node.left)
            && isBalanced(node.right)) {
        return true;
    }

    /* If we reach here then tree is not height-balanced */
    return false;
}

/* UTILITY FUNCTIONS TO TEST isBalanced() FUNCTION */
/* returns maximum of two integers */
int max(int a, int b) {
    return (a >= b) ? a : b;
}

/*  The function Compute the "height" of a tree. Height is the
 number of nodes along the longest path from the root node
 down to the farthest leaf node.*/
int height(Node node) {

    /* base case tree is empty */
    if (node == null) {
        return 0;
    }

    /* If tree is not empty then height = 1 + max of left
     height and right heights */
    return 1 + max(height(node.left), height(node.right));
}

public static void main(String args[]) {
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
```

```
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.left.left.left = new Node(8);

        if(tree.isBalanced(root))
            System.out.println("Tree is balanced");
        else
            System.out.println("Tree is not balanced");
    }
}

// This code has been contributed by Mayank Jaiswal
```

Time Complexity: O(n^2) Worst case occurs in case of skewed tree.

Optimized implementation: Above implementation can be optimized by calculating the height in the same recursion rather than calling a height() function separately. Thanks to Amar for suggesting this optimized version. This optimization reduces time complexity to O(n).

```
/* program to check if a tree is height-balanced or not */
#include<stdio.h>
#include<stdlib.h>
#define bool int

/* 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;
};

/* The function returns true if root is balanced else false
   The second parameter is to store the height of tree.
   Initially, we need to pass a pointer to a location with value
   as 0. We can also write a wrapper over this function */
bool isBalanced(struct node *root, int* height)
{
  /* lh --> Height of left subtree
     rh --> Height of right subtree */
  int lh = 0, rh = 0;

  /* l will be true if left subtree is balanced
```

```
    and r will be true if right subtree is balanced */
  int l = 0, r = 0;

  if(root == NULL)
  {
    *height = 0;
     return 1;
  }

  /* Get the heights of left and right subtrees in lh and rh
    And store the returned values in l and r */
  l = isBalanced(root->left, &lh);
  r = isBalanced(root->right,&rh);

  /* Height of current node is max of heights of left and
     right subtrees plus 1*/
  *height = (lh > rh? lh: rh) + 1;

  /* If difference between heights of left and right
     subtrees is more than 2 then this node is not balanced
     so return 0 */
  if((lh - rh >= 2) || (rh - lh >= 2))
    return 0;

  /* If this node is balanced and left and right subtrees
    are balanced then return true */
  else return l&&r;
}


/* UTILITY FUNCTIONS TO TEST isBalanced() FUNCTION */

/* 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);
}

int main()
{
  int height = 0;

  /* Constructed binary tree is
```

```
            1
          /   \
         2      3
       /  \    /
      4    5  6
     /
    7
   */
   struct node *root = newNode(1);
   root->left = newNode(2);
   root->right = newNode(3);
   root->left->left = newNode(4);
   root->left->right = newNode(5);
   root->right->left = newNode(6);
   root->left->left->left = newNode(7);

   if(isBalanced(root, &height))
     printf("Tree is balanced");
   else
     printf("Tree is not balanced");

   getchar();
   return 0;
}
```

Time Complexity: O(n)

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.

170 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

Rajat Soni • 5 days ago

Simplified code-

```
int check(struct bstnode* t)

{

if(t==NULL) return 0;

else

{

if(abs(check(t->left)-check(t->right))>1) {cout<<"Unbalanced tree.";exit(0);}

else

{
```

{

return 1+max(check(t->left),check(t->right));

}

}

}

• Reply • Share ›

Bhargav Jhaveri • 23 days ago

How complexity (case - I) in the case of Skewed tree would be O(n^2). I think it is the best case in which, we can terminate the loop/recursion in the first two lines only. In that case, when we find that height of left subtree is 0 and height of right subtree is some N, diff is N. If N is more than 1, it is violated. Condition of height difference would be false. So this would be the case of O(n).

In the case of balanced tree, the complexity should be O(n^2)

• Reply • Share ›

AllergicToBitches • 2 months ago

Can't get any simpler -
http://ideone.com/4Q0nNZ

1 • Reply • Share ›

Sagar Baver • 2 months ago

For the following binary tree the above code gives output as tree is balanced, while it is evident from tree structure that tree is not balanced.

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

Shouldn't we check and compare for both minimum and maximum heights of left and right sub trees at every node?

PS: binary tree image: http://s15.postimg.org/nuo18bx...

1 • Reply • Share ›

Anand Nahar ➦ Sagar Baver • a month ago

This is what I think, may be wrong!!

The tree in the eg is balanced..As per the definition, height of a tree is maximum number of nodes on root to leaf path. So there is no question of considering minimum height. For each root we just have to consider the maximum height of left & right subtree.

• Reply • Share ›

Rohit • 2 months ago

In java:
https://gist.github.com/rohitk...

• Reply • Share ›

Rashmi • 2 months ago

How the time complexity O(n^2) for method 1 for skewed trees?

• Reply • Share ›

karmveer@gupta • 3 months ago

we can stop traversing tree as soon as we find it is not balanced. No need to traverse complete tree once we find it is not balanced.
Here is my idea...

```
bool isBalanced(struct node *root, int* height)
{
int lh , rh,k ;
if(root == NULL)
{
*height = 0;
return 1;
}

k = isBalanced(root->left, &lh);
if(!k)
return 0;
k = isBalanced(root->right,&rh);
if(!k)
return 0;
*height = (lh > rh? lh: rh) + 1;
if((lh - rh >= 2) || (rh - lh >= 2))
return 0;
return 1;
}
```

• Reply • Share ›

Jatin • 4 months ago

www.coder2design.com

Java version:

```
private boolean isBalanced(Node<Integer> node){
                if( node == null){
                        return true;
```

```
                }
                int lHeight = height(node.getLeftChild());
                int rHeight = height(node.getRightChild());
                int diff = lHeight > rHeight ? lHeight-rHeight : rHeight-lHeight;
                if( diff > 1){
                        return false;
                }
                boolean isBal = isBalanced(node.getLeftChild());
                if(isBal){
                        isBal = isBalanced(node.getRightChild());
                }
                return isBal;
        }
```

▲ | ▼ • Reply • Share ›

Saddam • 4 months ago

we can just check the difference between max height and min height. If it is greater than 1, then the tree is not balanced.

▲ | ▼ • Reply • Share ›

Ninja ➦ Saddam • 2 months ago

That difference can be lower or equal 1 and three can still be unbalanced, because in order for tree to be balanced every subtree of it must be balanced.

▲ | ▼ • Reply • Share ›

Sapaka • 4 months ago

I am not getting why the complexity in method2 is O(n), when tree is skewed it will reach to O(n^2). Please explain this.

▲ | ▼ • Reply • Share ›

Sougata • 5 months ago

why can't we have 2 functions:
MaxHeight of root(same as height func here): returns recursively : max(lh, rh)+1
MinHeight: returns recursively : min(lh, rh)+1

Now at root do a abs(maxheight - minheight) and check if it's >1 (then return false, else true)

I mean essentially it's doing the same thing, just easier to understand...no?
Complexity : O[n+n] = O[n]

1 ▲ | ▼ • Reply • Share ›

Siya ➚ Sougata • 4 months ago

No it will not. Check the given tree

1
/ \
2 3
/ \ / \
3 5 4 5
\ / \ / \
4 6 7 8 9
/
10

check this to see tree http://ideone.com/OVS8xn
as here tree structure is not shown clearly.
This is height balanced. Acc to your approach minheight is =2 and max height is 4, and its not balanced acc to given approach.

1 ▲ | ▼ • Reply • Share ›

Saddam ➚ Siya • 4 months ago

here min height is 3

▲ | ▼ • Reply • Share ›

Siya ➚ Saddam • 4 months ago

Path from 1->2->5 make min height 2.

▲ | ▼ • Reply • Share ›

Satish ➚ Sougata • 4 months ago

Yeah, I think this is a good approach.

▲ | ▼ • Reply • Share ›

Ramendu Shandilya • 5 months ago

//Works in O(n), Java implementation ; isHtBal is a global variable

```
public int heightBalance(BSTNode root){
                if(root == null)
                {
                        return 0;
                }
                int lh = heightBalance(root.getLeftChild());
                int rh = heightBalance(root.getRightChild());
                int difference = Math.abs(lh - rh);
                if(difference > 1 && htCount == 0)
                {
                        isHtBal = false;
                        htCount++;
```

```
                }
                else
                {
                        isHtBal = true;
                }
                return (lh > rh ? lh+1 : rh +1);
        }
```

• Reply • Share ›

Shambhavi Shinde • 7 months ago

This is a simpler code with O(n)

int isBalanced(struct node *n)
{

if(!n->left&&!n->right)

return 1;

int l=0,r=0;

if(n->left)

l=isBalanced(n->left);

else

l=0;

if(n->right)

---

see more

• Reply • Share ›

Anonymous • 7 months ago

In the second soln we have -

l = isBalanced(root->left, &lh);
r = isBalanced(root->right,&rh);

I think we should return false as soon as we determine that either left subtree or right subtree is not balanced.

l = isBalanced(root->left, &lh);
if (!l) return false;
r = isBalanced(root->right,&rh);

if (!r) return false;

This is an optimization for cases where the input binary tree is not balanced.

• Reply • Share ›

Praveen Yadav • 7 months ago

Another O(n) solution using reference in c++:
#include "bits/stdc++.h"

using namespace std;

typedef struct node{

int data;

node *left;

node *right;

}node;

node *newNode(int new_data)

{

node *new_node=new node;

see more

• Reply • Share ›

Apoorv Shrivastava • 8 months ago

Why wont simple postorder traversal with left and right count at each node work?

• Reply • Share ›

lucy ➚ Apoorv Shrivastava • 8 months ago

no, in postorder it will return no. of node in left and right side but u require height of left and right of any node

• Reply • Share ›

sasha • 9 months ago

What does abs signifies in first method

• Reply • Share ›

Deepak Sharma ➚ sasha • 9 months ago

abs means absolute value. Consider this for example abs(-10)=10 and abs(5)=5.

1 • Reply • Share ›

sasha ➚ Deepak Sharma • 8 months ago

Thankuuu so much

• Reply • Share ›

sasha ➚ Deepak Sharma • 9 months ago

thankuu so much

• Reply • Share ›

tree_ds • 9 months ago

can anybody plz explain the function calls in isBalance function

• Reply • Share ›

Bibek Luitel • 9 months ago

```
int isbalance( struct tree *root){
if(root==NULL){ return 0;}
int left, right;
left= isbalance( root->left);
right= isbalance( root->right);
if( left==-1 || right==-1){ return -1;}
int diff;
if(left>right){ diff= left- right;}
else { diff = right-left;}
if(diff>1){ return -1;}
else {
return ((left>right)? left:right)+1;
}
}
```

O(n) complexity

is there any bug in these conditions ???

• Reply • Share ›

infinity • 9 months ago

The above illustrated O(n) code is not giving correct result for the below input for the given "Balanced" definition :-

---

```
struct node *root = newNode(10);

root->left = newNode(8);

root->left->left= newNode(7);

root->left->right= newNode(9);
```

```
root->left->left->left= newNode(6);

----------------------
```


nikhilnvj • 9 months ago

```
public boolean isBalanced(BTNode root){

if(root==null) return true;

int lheight=getHeight(root.left);

int rheight=getHeight(root.right);

if(Math.abs(lheight-rheight)>1)

return false;

return isBalanced(root.left) && isBalanced(root.right);

}

public int getHeight(BTNode node){

if(node==null) return 0;

else

return 1+Math.max(getHeight(node.left), getHeight(node.right));

}
```


Pankaj Kushwaha • 10 months ago

we can use global variable also:

```
int diameterlength=0;
void diameter(struct node * tree)

{

/* base case where tree is empty */

if (tree == 0)

{

return;

}
```

/* get the height of left and right sub-trees */

int lheight = height(tree->left);

see more

• Reply • Share ›

pk • 10 months ago

int maxd(node* r){ //Maximum depth

if (r == NULL)
return 0;

return 1 + max(maxd(r->left), maxd(r->right));

}

int mind(node* r){ //Minimum depth

if (r == NULL)
return 0;

return 1 + min(mind(r->left), mind(r->right));

}

int isBalanced(struct node* r){

return (maxd(r) - mind(r)) <= 1;

}

• Reply • Share ›

DS+Algo ➚ pk • 10 months ago

Kyun bhai.. aur waise bhi it's failing
. 1
. / \
. 2 3
. /
. 4

4 is left child of 3

• Reply • Share ›

Anand • a year ago

Why do we need to traverse whole tree ? If we check abs(lh-kh) <= 1; isn't it enough for us to know tree is balanced ? please explain... thanks.

• Reply • Share ›

GOPI GOPINATH ➦ Anand • a year ago

Only if we traverse the tree, we can get the values lh and rh.

• Reply • Share ›

rbchtt ➦ GOPI GOPINATH • 10 months ago

this is wrong.
take this tree:
10
/ \
3 15
/ / \
2 14 16
/ \
1 20

if you check roots' height it will give you 3 on left and 3 on right.
but as you can see, we have a problem at node #3 where left height is 2 and right height is 0 - what give you an illegal AVL tree...
we should traverse the whole nodes for that purpose.

1 • Reply • Share ›

Anand ➦ GOPI GOPINATH • a year ago

we are getting lh and kh from function height.

• Reply • Share ›

GOPI GOPINATH ➦ Anand • a year ago

We need to get lh and rh at every node. lh and rh varies at each node ryt. Thats the reason we traverse the nodes and find lh and rh at each node to check the condition.

Understand that we are finding the first node where the hights are not balanced.

• Reply • Share ›

Anand ➦ GOPI GOPINATH • a year ago

At root, height of right subtree should be either equal or max difference of 1 from height of left subtree. If we check this that's enough. because at any level downward in the tree we are not going to find some different trend. so what is the purpose to check that.

• Reply • Share ›

Siddharth ➚ Anand • a year ago

I don't think that would be enough to only check difference between left and right subtree height at the root of the tree. As, there can be other nodes(parents) in tree as well that are not balanced but at root, it's left and right subtree may seem to be balanced. So, we need to traverse entire tree until we find one unbalanced node(parent).

• Reply • Share ›

GOPI GOPINATH ➚ Anand • a year ago

Thats the second optimized solution in the post!!

• Reply • Share ›

Ashish ➚ GOPI GOPINATH • 7 months ago

@Anand rbchtt has clearly given an example where your logic fails. Thus, we will need to compare the heights at each node

• Reply • Share ›

thevagabond85 • a year ago

the optimized version should be organized like this :

http://ideone.com/xHkQD0

• Reply • Share ›

Aryan Parker • a year ago

In the second method, why are we returning l&&r ? cant we just return 1. or can we change the last two lines of method 2 to these (similar to the one in 1st method)

if((abs(lh-rh)<=1)&&l&&r)return 1;
else return 0;

• Reply • Share ›

Guest • a year ago

Can anyone tell me why in the function is_balanced we need to still do recursion on is_balanced? Why not just compare once if abs(lh-rh) > 1 return 0; else return 1?

• Reply • Share ›

Jerry Goyal • a year ago

shorter is better.

```
bool is(struct node *root)
```

```
{
   if(root==NULL){
        return 1;
   }
   is(root->left);
   is(root->right);
   return (abs(height(root->left)-height(root->right))<2);
}
```

• Reply • Share ›

Guest • a year ago

```
int height(struct node* node){  //height function
        if(node==NULL){
                return 0;
        }
        int a=height(node->left);
        int b=height(node->right);
        return (1+(a>b?a:b));
}
//call the height function for left and right subtree
int isBalanced(struct node* node)
{
        if(abs(height(node->left)-height(node->right))<2){
                return 1;
        }
        else return 0;
}
```

• Reply • Share ›

AllergicToBitches • a year ago

It is a DP problem.

• Reply • Share ›

Batman ➚ AllergicToBitches • a year ago

how ?

• Reply • Share ›

Load more comments