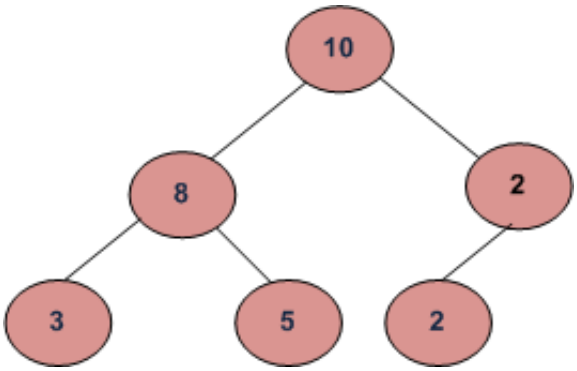
Root to leaf path sum equal to a given number

Given a binary tree and a number, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given number. Return false if no such path can be found.



For example, in the above tree root to leaf paths exist with following sums.

21 –> 10 – 8 – 3

23 –> 10 – 8 – 5

14 –> 10 – 2 – 2

So the returned value should be true only for numbers 21, 23 and 14. For any other number, returned value should be false.

Algorithm:

Recursively check if left or right child has path sum equal to (number – value at current node)

Implementation:

C

```
#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;
};

/*
 Given a tree and a sum, return true if there is a path from the root
 down to a leaf, such that adding up all the values along the path
 equals the given sum.

 Strategy: subtract the node value from the sum when recurring down,
 and check to see if the sum is 0 when you run out of tree.
*/
bool hasPathSum(struct node* node, int sum)
{
  /* return true if we run out of tree and sum==0 */
  if (node == NULL)
  {
     return (sum == 0);
  }

  else
  {
    bool ans = 0;

    /* otherwise check both subtrees */
    int subSum = sum - node->data;

    /* If we reach a leaf node and sum becomes 0 then return true*/
    if ( subSum == 0 && node->left == NULL && node->right == NULL )
      return 1;

    if(node->left)
      ans = ans || hasPathSum(node->left, subSum);
    if(node->right)
      ans = ans || hasPathSum(node->right, subSum);

    return ans;
  }
}

/* UTILITY FUNCTIONS */
/* 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()
{

  int sum = 21;

  /* Constructed binary tree is
            10
          /   \
```

```
        8      2
      /  \    /
     3    5  2
  */
  struct node *root = newnode(10);
  root->left        = newnode(8);
  root->right       = newnode(2);
  root->left->left  = newnode(3);
  root->left->right = newnode(5);
  root->right->left = newnode(2);

  if(hasPathSum(root, sum))
    printf("There is a root-to-leaf path with sum %d", sum);
  else
    printf("There is no root-to-leaf path with sum %d", sum);

  getchar();
  return 0;
}
```

Run on IDE

Java

```
// Java program to print to print root to leaf path sum equal to
// a given number

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;

    /*
     Given a tree and a sum, return true if there is a path from the root
     down to a leaf, such that adding up all the values along the path
     equals the given sum.

     Strategy: subtract the node value from the sum when recurring down,
     and check to see if the sum is 0 when you run out of tree.
     */

    boolean haspathSum(Node node, int sum) {
        if (node == null) {
            return (sum == 0);
        } else {
            boolean ans = false;

            /* otherwise check both subtrees */
            int subsum = sum - node.data;
            if (subsum == 0 && node.left == null && node.right == null) {
                return true;
```

```
            }
            if (node.left != null) {
                ans = ans || haspathSum(node.left, subsum);
            }
            if (node.right != null) {
                ans = ans || haspathSum(node.right, subsum);
            }
            return ans;
        }
    }


    public static void main(String args[]) {
        int sum = 14;
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);

        if (tree.haspathSum(root, sum)) {
            System.out.println("There is a root to leaf path with sum " + sum);
        } else {
            System.out.println("There is no root to leaf path with sum " + sum);
        }
    }
}

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

Run on IDE

Time Complexity: O(n)

References:

http://cslibrary.stanford.edu/110/BinaryTrees.html

Author: Tushar Roy

Please write comments if you find any bug in above code/algorithm, or find other ways to solve the same problem

191 Comments Category: Trees

Related Posts:

- 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?
- Closest leaf to a given node in Binary Tree

Bhargav Jhaveri • 23 days ago

I have modified the hasPathSum as:
Please provide your suggestions.

```
if (node == null) {

return (sum == 0);

} else {

/* otherwise check both subtrees */

int subsum = sum - node.data;

boolean l = haspathSum(node.left,subsum);

boolean r = haspathSum(node.right,subsum);

return l||r;

}
```

• Reply • Share ›

9Codie05 • a month ago

can someone explain how is recursion working here,
in these lines

```
if(node->left)

ans = ans || hasPathSum(node->left, subSum);

if(node->right)

ans = ans || hasPathSum(node->right, subSum);
```

when the control reaches back to node 8 while unwinding, it should go to node 5 and then it should call function hasPathSum again but it does not i tried it in code blocks.

• Reply • Share ›

Saumya Dubey • a month ago

http://ideone.com/WWAibB

c++ code

• Reply • Share ›

Jeff • a month ago

code in python
http://code.geeksforgeeks.org/...
• Reply • Share ›

Deepak • a month ago
Simple logic and implementation:

1. at each stage , keep adding the sum of data of previous all nodes to the current node data.
2. since this sum needs to be preserved , so keep it passing in the recursive function.

```
int root_to_leaf_sum(node *root,int sum)
{
if (NULL == root)
{
if (sum == number)
return 1;
else
return 0;
}
sum = sum + root->data;

return (root_to_leaf_sum(root->left, sum) || root_to_leaf_sum(root->right, sum))
}
```
• Reply • Share ›

raj • 2 months ago
we can simply have in else (hasPathSum(node->left,subSum)||hasPathSum(node->right,subSum))
return 1;
no need of that ans thing.
1 • Reply • Share ›

Fragrance raj • 2 months ago
Well I guess if you try running above code with sum=12 (in main())..it would give a wrong answer with your logic!
• Reply • Share ›

Fragrance raj • 2 months ago
Even I have same doubt! Use of "ans" is redundant here!
• Reply • Share ›

Fragrance Fragrance • 2 months ago
Well I guess if you try running above code with sum=12 (change in main()

)..it would give a wrong answer with your logic!

• Reply • Share ›

Piyush Chaudhary • 2 months ago

Another Efficient code -->

public class rootLeafSum {

static boolean sum(treeNode root,int sum,int total)

{

if(root==null)

return false;

total+=root.data;

if(total==sum && root.left==null && root.right==null)

return true;

else

return sum(root.left,sum,total)||sum(root.right,sum,total);

see more

• Reply • Share ›

yogesh • 4 months ago

what if we also want to print that path ??

• Reply • Share ›

Jatin • 4 months ago

www.coder2design.com

```
private boolean sumFromRootToLeaf( Node<Integer> node, int sum, int expectedSum){
                if( node == null){
                        return false;
                }
                sum = sum + node.getData();
                if( node.getLeftChild() == null && node.getRightChild() == null) {
                        if( sum == expectedSum){
                                return true;
                        }
                }
                boolean valid = sumFromRootToLeaf(node.getLeftChild(), sum, expectedSu
                if( !valid)
```

```
                if( !valid)
                        valid = sumFromRootToLeaf(node.getRightChild(), sum, expectedS
                sum = sum - node.getData();
                return valid;
        }
```

1 • Reply • Share ›

Palash Goel • 4 months ago

int sumPathExists(struct treenode *root, int sum){
if(!root){
return !sum;
}

return sumPathExists(root->left, sum - root->data) || sumPathExists(root->right, sum - root->data);

}

• Reply • Share ›

amrutha • 5 months ago

It works efficiently!! bool haspath(node,sum){
f(!node)
return false;

sum=sum-node->data;
if(sum==0&&!node->lptr&&!node->rptr)
return true;
else{
haspath(node->lptr,sum);
haspath(node->rptr,sum);
}
}
return false;
}

• Reply • Share ›

UK • 6 months ago

Need to return false instead of (sum == 0), if root is null.
sum == 0 fails when tree is empty and sum input is 0.

• Reply • Share ›

pankkum2 • 6 months ago

I think, following is better and also efficient.

```
bool hasPathSum(struct node* node, int sum)
{
if (node == NULL || sum < 0)
{
return false;
}
else
{
sum = sum - node->data;
if (sum == 0 && node->left == NULL && node->right == NULL)
return true;
else
return hasPathSum(node->left, sum) || hasPathSum(node->right, sum);
}
}
```

• Reply • Share ›

Ankit Singh • 6 months ago

http://code.geeksforgeeks.org/

• Reply • Share ›

Holden • 6 months ago

Here there are 3 solutions in Java:
http://codereview.stackexchang...

First one, recursive, 2nd one by DFS, and last one by BFS.

• Reply • Share ›

Flower girl • 7 months ago

y is bool ans = 0; used??

2 • Reply • Share ›

NANDHINI G • 7 months ago

Try this ...it works..
this code will print the path if sum is correct and returns 1 if sum is correct else 0 to main()

```
int root_leaf_sum(struct tree *node,int user_s,int cal_sum)
{
if(node==NULL)
return (user_s==cal_sum);
cal_sum+=node->data;
if(root_leaf_sum( node->left,user_s,cal_sum) || root_leaf_sum(node-
>right,user_s,cal_sum))
{
```

```
printf("%d ",node->data);
return 1;
}
return 0;
}
```

• Reply • Share ›

Abhishek Kumar • 7 months ago

return sum==0
signifies what?

• Reply • Share ›

Anmol Varshney ➦ Abhishek Kumar • 7 months ago

It checks that if root is NULL and if sum is zero then return true otherwise return false.

1 • Reply • Share ›

ankita • 7 months ago

how to write code for max sum root to leaf path?

• Reply • Share ›

rohit • 7 months ago

```
bool hasPathSum(node *root,int sum)

{

if(sum<0 || root==NULL && sum!=0) ///

return false;

if(root==NULL && sum==0)

return true;

return (hasPathSum(root->left,sum-root->data) || hasPathSum(root->right,sum-root->data));

}
```

2 • Reply • Share ›

Holden ➦ rohit • 6 months ago

Your code fails for 12 ....
It returns true, instead of false;

• Reply • Share ›

Rakesh • 8 months ago

To improve in negative cases you should include "sum !=0" before going further deep check

if(node->left && sum !=0)
ans = ans || hasPathSum(node->left, subSum);
if(node->right && sum !=0)
ans = ans || hasPathSum(node->right, subSum);

1 • Reply • Share ›

Chandra Kant Siyag • 8 months ago

//Is this function is correct
bool hasPathSum(struct node* node, int sum)

{

if(node==NULL)

return(sum==0);

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

return(sum==node->data);

else

{

return(hasPathSum(node->left,sum-node->data) || hasPathSum(node->right,sum-node->data));

}

}

• Reply • Share ›

jeff • 8 months ago

A much easier implementation.

http://ideone.com/TWz8gw

• Reply • Share ›

xtreme • 8 months ago

another approach is adding sum till we reach leaf node and then checking the sum-

int has(struct node *root,int sum,int k)

{

```
if(root==NULL)

return 0;

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

{

if(sum==(k+root->data))

return 1;

else

return 0;
```

see more

• Reply • Share ›

Ronit Halder • 8 months ago

We should check for -ve on subSum. That will decrease the execution time

• Reply • Share ›

sasha • 9 months ago

what does return(sum==0);
signifies

• Reply • Share ›

sanjeev sasha • 8 months ago

suppose you have an empty tree,and you want to check for sum=k.then it must return false because empty tree must have sum 0. so if sum 0 in case of node==null , then it will return true, otherwise false.. same as suppose you have visited all the nodes and reach to null but you don't have sum==0.then again it will return false.

• Reply • Share ›

Ankit Gupta sanjeev • 8 months ago

But I think it will never reach NULL while traversing because we are traversing the tree further only if left and right child of the tree are not empty. Although the first case you mentioned is correct.

Correct me if I'm wrong. :)

1 • Reply • Share ›

sasha • 9 months ago

Why using bool in this code?

Why using bool in this code?

• Reply • Share ›

sanjeev ➚ sasha • 8 months ago

because we need to check that, if given tree is holding the path sum property then return true otherwise return false.

• Reply • Share ›

Jude Martin • 9 months ago

The code will break for the condition when sum is 0 and the root is null

Ex:
hasPathSum(root, sum))// root == null and the sum = 0 ;

as per the code it enters the first if condition and return true as sum == 0
but as per the question it should be false; as there is no such path

hence adding this line will pass this case

```
if(sum==0 && root ==null){
return false;
}
```

/////// Java implementation

```
public boolean hasPathSum(TreeNode root, int sum) {
if(sum==0 && root ==null) return false;
if(root == null){
return (sum==0);
```

see more

• Reply • Share ›

Deepak Sharma • 9 months ago

c++ implementtion

http://ideone.com/h7us2f

• Reply • Share ›

Lokesh • 9 months ago

Clean and simple !!
http://code.geeksforgeeks.org/...

1 • Reply • Share ›

Holden ➚ Lokesh • 6 months ago

You said return true, but you return 0!

• Reply • Share ›

Holden ➚ Lokesh • 6 months ago

Look at this line which you have written:

```
/* return true if we run out of tree and sum==0 */
 if (node == NULL)
 return 0;
```

• Reply • Share ›

Mayank yadav • 9 months ago

to save execution time, can we use exit function after the condition if (subSum == 0 && node->left == NULL && node->right == NULL) ?

• Reply • Share ›

Jatin Jain ➚ Mayank yadav • 9 months ago

exit function terminates the process. It won't impact if you are just writing this function and no more functionality/code is to follow. However, if your code consists of more functioning after the call to this function, exit will terminate the process without executing anything else.

1 • Reply • Share ›

Rinni Jain • 9 months ago

Is their any test case this won't pass..

```
bool hasPathSum(struct node* node,int sum)
{
if(node==NULL)
return sum==0;
sum=sum-node->data;
return (pass(node->left,sum) || pass(node->right,sum));

}
```

• Reply • Share ›

jerky ➚ Rinni Jain • 9 months ago

yes there are cases

1
/ \
2 3
/ \ \
4 5 7
/
8

here it will return true for sum = 10 as a path (1-3-7) but it is not root to leaf.

the tree is
root=newNode(1);
root->left=newNode(2);
root->right=newNode(3);
root->left->left=newNode(4);
root->left->right=newNode(5);
root->right->right=newNode(7);
root->right->right->left=newNode(8);

• Reply • Share ›

Eknoor jerky • 9 months ago

1-3-7 is actually a root to leaf path. The code above will return true for sum=4 for the tree you have drawn. it will return 1-3. We have to check for leaf nodes separately, otherwise those nodes with one child only can return a wrong ans

• Reply • Share ›

Rinni Jain Eknoor • 9 months ago

yes you are right, in that case my code wont work.
Thank you Eknoor and Jerkey :)

1 • Reply • Share ›

Eknoor Rinni Jain • 9 months ago

Very compact code :). Here's mine:

```
bool isSum(node *r,int k)
{
if(r==NULL)
return false;
if(r->rchild==NULL && r->lchild==NULL)
return r->key==k;
return isSum(r->lchild,k-(r->key)) || isSum(r->rchild,k-(r->key));
}
```

• Reply • Share ›

Komal Singh • 10 months ago

Much more easier and compact code :)

```
int isHeightBalanced(struct tree *root,int num,int sum){

if(root==NULL)

return 0;

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

sum=sum+root->value;

return ((sum==num && root->left==NULL && root->right==NULL) ||
isHeightBalanced(root->left,num,sum) || isHeightBalanced(root->right,num,sum));

return 0;

}

• Reply • Share ›

Anshu Singh ➚ Komal Singh • 10 months ago

Nice!

• Reply • Share ›

thevagabond85 • a year ago

Alternate way of writing the given solution is : (avoids using that extra boolean ans flag)
http://ideone.com/REhRat

bool hasPathSum2(node* root, int sum)
{

if(root == NULL)
return (sum==0);
int subSum = sum - (root->data);
if(subSum ==0 && root->left==NULL && root->right ==NULL
|| (root->left && hasPathSum(root->left, subSum))
|| (root->right && hasPathSum(root->right, subSum))
)
return true;

return false;
}

• Reply • Share ›

Load more comments