Foldable Binary Trees

Question: Given a binary tree, find out if the tree can be folded or not.

A tree can be folded if left and right subtrees of the tree are structure wise mirror image of each other. An empty tree is considered as foldable.

```
Consider the below trees:
(a) and (b) can be folded.
(c) and (d) cannot be folded.

(a)
       10
     /    \
    7      15
     \    /
      9  11

(b)
        10
       /  \
      7    15
     /      \
    9       11

(c)
        10
       /  \
      7   15
     /    /
    5   11

(d)

         10
       /   \
      7     15
    /  \    /
   9   10  12
```

Method 1 (Change Left subtree to its Mirror and compare it with Right subtree)

Algorithm: isFoldable(root)

```
1) If tree is empty, then return true.
2) Convert the left subtree to its mirror image
    mirror(root->left); /* See this post */
3) Check if the structure of left subtree and right subtree is same
   and store the result.
    res = isStructSame(root->left, root->right); /*isStructSame()
        recursively compares structures of two subtrees and returns
        true if structures are same */
4) Revert the changes made in step (2) to get the original tree.
    mirror(root->left);
5) Return result res stored in step 2.
```

Thanks to ajaym for suggesting this approach.

C

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

/* You would want to remove below 3 lines if your compiler
   supports bool, true and false */
#define bool int
#define true 1
#define false 0

/* 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;
};

/* converts a tree to its mrror image */
void mirror(struct node* node);

/* returns true if structure of two trees a and b is same
   Only structure is considered for comparison, not data! */
bool isStructSame(struct node *a, struct node *b);

/* Returns true if the given tree is foldable */
bool isFoldable(struct node *root)
{
  bool res;
  /* base case */
  if(root == NULL)
    return true;

  /* convert left subtree to its mirror */
  mirror(root->left);

  /* Compare the structures of the right subtree and mirrored
    left subtree */
  res = isStructSame(root->left, root->right);

  /* Get the originial tree back */
  mirror(root->left);

  return res;
}


bool isStructSame(struct node *a, struct node *b)
{
  if (a == NULL && b == NULL)
  {  return true; }
  if ( a != NULL && b != NULL &&
       isStructSame(a->left, b->left) &&
       isStructSame(a->right, b->right)
     )
  {  return true; }

  return false;
}


/* UTILITY FUNCTIONS */
/* Change a tree so that the roles of the  left and
    right pointers are swapped at every node.
    See http://geeksforgeeks.org/?p=662 for details */
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}

/* 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 mirror() */
int main(void)
{
  /* The constructed binary tree is
         1
       /   \
      2     3
      \    /
       4  5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->right->left  = newNode(4);
  root->left->right = newNode(5);

  if(isFoldable(root) == 1)
  { printf("\n tree is foldable"); }
  else
  { printf("\n tree is not foldable"); }

  getchar();
  return 0;
}
```

Java

```
// Java program to check foldable binary tree

// 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 the given tree is foldable */
    boolean isFoldable(Node node) {
        boolean res;

        /* base case */
        if (node == null) {
            return true;
        }

        /* convert left subtree to its mirror */
        mirror(node.left);

        /* Compare the structures of the right subtree and mirrored
         left subtree */
        res = isStructSame(node.left, node.right);

        /* Get the originial tree back */
        mirror(node.left);

        return res;
    }

    boolean isStructSame(Node a, Node b) {
        if (a == null && b == null) {
            return true;
        }
        if (a != null && b != null
                && isStructSame(a.left, b.left)
                && isStructSame(a.right, b.right)) {
            return true;
        }

        return false;
    }

    /* UTILITY FUNCTIONS */
    /* Change a tree so that the roles of the  left and
     right pointers are swapped at every node.
     See http://geeksforgeeks.org/?p=662 for details */
    void mirror(Node node) {
        if (node == null) {
            return;
        } else {
            Node temp;

            /* do the subtrees */
            mirror(node.left);
            mirror(node.right);

            /* swap the pointers in this node */
            temp = node.left;
            node.left = node.right;
            node.right = temp;
        }
    }

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

        if (tree.isFoldable(root)) {
            System.out.println("tree is foldable");
        } else {
            System.out.println("Tree is not foldable");
        }

    }

}

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

Time complexity: O(n)

Method 2 (Check if Left and Right subtrees are Mirror)

There are mainly two functions:

// Checks if tree can be folded or not

```
IsFoldable(root)
```

```
1) If tree is empty then return true
2) Else check if left and right subtrees are structure wise mirrors of
    each other. Use utility function IsFoldableUtil(root->left,
    root->right) for this.
```

// Checks if n1 and n2 are mirror of each other.

```
IsFoldableUtil(n1, n2)
1) If both trees are empty then return true.
2) If one of them is empty and other is not then return false.
3) Return true if following conditions are met
   a) n1->left is mirror of n2->right
   b) n1->right is mirror of n2->left
```

C

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

/* You would want to remove below 3 lines if your compiler
   supports bool, true and false */
#define bool int
#define true 1
#define false 0

/* 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;
};

/* A utility function that checks if trees with roots as n1 and n2
  are mirror of each other */
bool IsFoldableUtil(struct node *n1, struct node *n2);

/* Returns true if the given tree can be folded */
bool IsFoldable(struct node *root)
{
     if (root == NULL)
     {  return true;  }

     return IsFoldableUtil(root->left, root->right);
}

/* A utility function that checks if trees with roots as n1 and n2
  are mirror of each other */
```

```
bool IsFoldableUtil(struct node *n1, struct node *n2)
{
    /* If both left and right subtrees are NULL,
      then return true */
    if (n1 == NULL && n2 == NULL)
    {  return true;  }

    /* If one of the trees is NULL and other is not,
      then return false */
        if (n1 == NULL || n2 == NULL)
    {  return false; }

    /* Otherwise check if left and right subtrees are mirrors of
       their counterparts */
        return IsFoldableUtil(n1->left, n2->right) &&
           IsFoldableUtil(n1->right, n2->left);
}

/*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 mirror() */
int main(void)
{
  /* The constructed binary tree is
         1
       /   \
      2     3
      \    /
       4  5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->right  = newNode(4);
  root->right->left = newNode(5);

  if(IsFoldable(root) == true)
  { printf("\n tree is foldable"); }
  else
  { printf("\n tree is not foldable"); }

  getchar();
  return 0;
}
```

Java

```
// Java program to check foldable binary tree

// 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 the given tree can be folded */
    boolean IsFoldable(Node node) {
        if (node == null) {
            return true;
        }

        return IsFoldableUtil(node.left, node.right);
    }

    /* A utility function that checks if trees with roots as n1 and n2
     are mirror of each other */
    boolean IsFoldableUtil(Node n1, Node n2) {

        /* If both left and right subtrees are NULL,
         then return true */
        if (n1 == null && n2 == null) {
            return true;
        }

        /* If one of the trees is NULL and other is not,
         then return false */
        if (n1 == null || n2 == null) {
            return false;
        }

        /* Otherwise check if left and right subtrees are mirrors of
         their counterparts */
        return IsFoldableUtil(n1.left, n2.right)
                && IsFoldableUtil(n1.right, n2.left);
    }

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

        if (tree.IsFoldable(root) ) {
            System.out.println("tree is foldable");
        } else {
            System.out.println("Tree is not foldable");
        }

    }

}

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

Thanks to Dzmitry Huba for suggesting this approach.

Please write comments if you find the above code/algorithm incorrect, or find other ways to solve the same problem.

Spend less time debugging, more time coding.
Get Visual Studio Code FREE
Microsoft

24 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?

Like Share 5 people like this.

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

63 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Sboy • 2 months ago

#include <iostream>

#include<stdlib.h>

using namespace std;

struct node

{

int data;

node *left,*right;

};

struct node* newNode(int data)

{

struct node* node = (struct node*)

see more

• Reply • Share ›

Satish Kumar • 2 months ago

int twoTreeEq(bnode *t1,bnode* t2)

{

if(t1==NULL&&t2==NULL)

return 1;

if((t1==NULL&&t2!=NULL)||(t1!=NULL&&t2==NULL))

return 0;

return twoTreeEq(t1->lc,t2->rc)&&twoTreeEq(t1->rc,t2->lc);

}

• Reply • Share ›

DIVYA NAGAR • 7 months ago

we can also use level order traversal for getting the structure of the left sub tree and then compare it to the structure of right sub tree. :)

• Reply • Share ›

Anukool shrivastava • 7 months ago

there is no need of calling mirror function u can directly do it by calling isstructsame() and passing opposites nodes that is

bool isStructSame(struct node *a, struct node *b)

{

if (a == NULL && b == NULL)

{ return true; }

if (a != NULL && b != NULL &&

isStructSame(a->left, b->right) && //this is the change

isStructSame(a->right, b->left) //here also

)

{ return true; }

return false;

}

• Reply • Share ›

rahul khanna Anukool shrivastava • 4 months ago

I did the same
there wasn't a need to call mirror
using mirror was absurd

• Reply • Share ›

rs1993 Anukool shrivastava • 5 months ago

Is this approach correct or there exist some case where this fails ?

• Reply • Share ›

Apoorva • 8 months ago

We can find the mirror tree of the original tree. Then store the elements in two separate arrays for both original and mirror trees by making inorder traversal. Then just check whether two arrays are reverse to each other or not.

• Reply • Share ›

Utkarsh Agarwal • 8 months ago

we can do a level order traversal checking if each level is a palindrome . each NULL can be replaced by any other special data.

• Reply • Share ›

Ramendu Shandilya ↪ Utkarsh Agarwal • 5 months ago

The complexity won't be exactly O(n) right ?? Not even O(nexp2), somewhere between both depending upon the length of the temporary array we would be using to check the palindrome of a level.

• Reply • Share ›

sasha • 8 months ago

I don't understand the question..can someone explain me

• Reply • Share ›

vimal ↪ sasha • 8 months ago

i can.. we want to check that if we put a mirror vertically at root node than are the nodes on left side have same structure as that on right side when seen in mirror ?(its not that there data values needs to be same,,... just the structure..)

• Reply • Share ›

Pankaj Kushwaha • 10 months ago

there may be a simple solution:

```
bool isFoldable(struct node *rootleft, struct node *rootright)
{
bool result;
if(rootleft!=NULL && rootright!=NULL )
{
result=isFoldable(rootleft->left,rootright->right);
if (result == false) return result;
result=isFoldable(rootleft->right,rootright->left);
}
else if (rootleft==NULL&&rootright==NULL)
return true;
else
return false;

return result;

}
```

ideone link
https://ideone.com/N3STS7

• Reply • Share ›

OURJAY • 10 months ago

take two nodes temp1 and temp2;

initialize both to root->left and root ->right respectively..

then:->

int isFoldable(node*temp1,node*temp2)

{

if(temp1==temp2==root)

{

return 1;

}

if(temp1==NULL&&temp2==NULL)

return 1;

if((temp1==NULL&&temp2!=NULL)||(temp2==NULL&&temp1!=NULL))

return 0;

return(isFoldable(temp1->left,temp2->right)&&isFoldable(temp1->right,temp2->left));

}

^ | ⌄ • Reply • Share ›

Tanish Bansal • 10 months ago

int folder(node *root){

if(!root)

return 0;

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

return 0;

else if(root->right)

return 1;

else

return -1;

return(folder(root->left)+folder(root->right));

}

^ | ⌄ • Reply • Share ›

NITIN PANCHAL • a year ago

We can also do the code as follows

//to check whether the given tree has a faesible mirror tree or not

//to create a double tree

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

using namespace std ;

#include<queue>

struct tree

{

int data ;

see more

^ | v • Reply • Share ›

Aryan Parker • a year ago

Only 3 lines of code- http://ideone.com/VpXmIE

2 ^ | v • Reply • Share ›

Mr. Lazy • a year ago

Simple & Short! http://ideone.com/jsgyhJ

5 ^ | v • Reply • Share ›

Aryan Parker → Mr. Lazy • a year ago

I have also written similar code. Is the code checked for all boundary cases?

^ | v • Reply • Share ›

Mr. Lazy → Aryan Parker • a year ago

Yes, it covers all the corner cases.

^ | v • Reply • Share ›

Rajendra Prasad • a year ago

//function to check tree is foldable or not
int IsFoldableTree(Tnode *Ltree, Tnode *Rtree)
{

```
if(Ltree==NULL && Rtree==NULL)
return 1;
else if( Ltree!=NULL && Rtree!=NULL
&& IsFoldableTree(Ltree->left,Rtree->right)
&& IsFoldableTree(Ltree->right,Rtree->left))
{
return 1;
}

return 0;
}

/*Main function*/
int main()
{
```

see more

• Reply • Share ›

Guest • a year ago

```
//function to check tree is foldable or not

int IsFoldableTree(Tnode *Ltree, Tnode *Rtree)

{

if(Ltree==NULL && Rtree==NULL)

return 1;

else if( Ltree!=NULL && Rtree!=NULL

&& IsFoldableTree(Ltree->left,Rtree->right)

&& IsFoldableTree(Ltree->right,Rtree->left))

{

return 1;
```

see more

• Reply • Share ›

Abhi • a year ago

GeeksforGeeks

we can simplen write a function.

```
bool TreeFoldable(Node* root1)

{

if(root1==NULL)

return true;

if(root1->left==NULL && root1->right!=NULL)

return false;

if(root1->left!=NULL && root1->right==NULL)

return false;

if(root1->left==NULL && root1->right==NULL)

return true;

return (TreeFoldable(root1->left) && TreeFoldable(root1->right));

}
```

• Reply • Share ›

Vineeth Reddy Abhi • a year ago

Your function is not comparing tree structure on either side of the root. It would fail in following case. Tree has only following elements root, root->left, root->left->left & root->right & root->right->left;

• Reply • Share ›

Aryan Parker Vineeth Reddy • a year ago

@vineeth its giving correct result for your case

• Reply • Share ›

Abhi Vineeth Reddy • a year ago

May be you are right. but please check it once again..

• Reply • Share ›

let_us_c • a year ago

calculate the width of each node and add them together
if ==0
foldable
else

not

^ | ⌄ • Reply • Share ›

scofield • a year ago

can we do like this
1.) maintain an array of size equal to the height of the tree and initialize all the elements of the array to 0.
2.) initialize a variable i=0.
3.) do a preorder traversal of the tree
(a) on going left decrement i
(b) on going right increment i
(c) add this i to the corresponding index=level no., in the array
4) at the end traverse the array
if(all the elements are zero)
then foldable
else
non foldable

^ | ⌄ • Reply • Share ›

sam • a year ago

is this is same for finding tree is symmetric or not??plz help
If any changes what are the changes..???

^ | ⌄ • Reply • Share ›

Anon ➦ sam • a year ago

Its the same : a tree is symmetric with vertical axis can be folded; i.e. the nodes of left and right subtrees can be superimposed.

^ | ⌄ • Reply • Share ›

ramshah • a year ago

1) Traverse Root->left and make one string (s1) which notes movement of Pointer
2) Traverse Root->Right and take another string (s2) which do same thing
3) Now from S2 replace all Left with Right And Right With Left
4) If S1 == S2 then OK else Not Foldable

eg

10
/ \
7 15 D - Down , U - Up , L - LEFT , R - Right
\ /
9 11

Step 1,2 : S1 = DLDRUU and S2 = DRDLUU

Step 3 : S2 = D(R->L)D(L->R)UU
.`. S2 = DLDRUU
Step 4 : S1 == S2 .`. Foldable
2 ^ | v • Reply • Share ›

Praveen Kumar • a year ago
/*A simple program using count variable */
#include<stdio.h>
#include<stdlib.h>

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

struct node *newNode(int data)
{
struct node *new_node=(struct node*)malloc(sizeof(struct node));
new_node->data=data;
new_node->left=NULL;
new_node->right=NULL;
return new_node;

see more

^ | v • Reply • Share ›

Tripathi • 2 years ago
there is no need to write mirror function ..

if (a != NULL && b != NULL &&
isStructSame(a->left, b->right) &&
isStructSame(a->right, b->left)
)
will replace extra code done for mirror in method 1
^ | v • Reply • Share ›

Anjaney Pandey • 2 years ago
In method 1 as we are neither using double pointer nor returning root in mirror method.Won't it create any problem ?
^ | v • Reply • Share ›

nobeita • 2 years ago
I think this can also be done by using level order traversal as well

I used vector as a queue.
Algorithm:
1)At each level except root there should be even number of elements as to allow the tree to be foldable.
2) If first condition is satisfied for a level then for that level take an iterator 'i' pointing to first element of that level and take pointer 'j' pointing to last element of the level.
check that if ith node left exists then jth node right should also exist similarly
if ith node right exist then jth node left should exist.Then do i++ j-- and traverse the list till while i is not equal to half the total no of element in the level.
3)If first and second condition is true keep flag=1 otherwise turn it to 0.
Correct me if i m wrong .

```
bool foldable(node*root)
{
vector<node*> v;
vector<node*>::iterator i,j;
bool flag=1;
```

see more

• Reply • Share ›

Vãîbhåv Joshî • 2 years ago
Java Code For Foldable

```
public boolean isFoldable(){
if(root==null)
return true;
return isFoldable(root.getLeft(),root.getRight());
}

public boolean isFoldable(BSTNode l , BSTNode r){
if(l==null && r==null)
return true;

if(l==null || r==null)
return false;

return isFoldable( l.getLeft(), r.getRight() ) && isFoldable(l.getRight(), r.getLeft());
}
}
```

1 • Reply • Share ›

Rudra Vãîbhåv Joshî • 2 years ago
I came up with the exact same solution myself! Here is the code in C++:

[code]

bool foldable(node *a, node *b) {

if(!a && !b)

return true;

if(!a || !b)

return false;

return foldable(a->left, b->right) && foldable(a->right, b->left);

}

bool isFoldable(node *root) {

if(!root)

return true;

return foldable(root->left, root->right);

}

[\code]

1 • Reply • Share ›

Suvodip Bhattacharya • 2 years ago

#include<iostream>
using namespace std;

struct tree{
int data;
struct tree *rc,*lc;
};

tree* newNode(int n)
{
struct tree *temp=new tree;
temp->lc=temp->rc=NULL;
temp->data=n;
return temp;
}

int foldableUtil(struct tree *root1,struct tree *root2)
{

if(root1==NULL && root2==NULL) return 1;

see more

• Reply • Share ›

AlienOnEarth • 2 years ago

What should be the answer to following tree?

10
/ \
7 15
/ \ \
1 2 3

This can be considered as foldable as 10-15-3 can lie on 10-7-1. If this should return true, than both methods provided would be wrong.

1 • Reply • Share ›

Anon AlienOnEarth • a year ago

I don't think this should be foldable. For a tree to be folding, it should be symmetric at its vertical axis and this is clearly not.
Just my views, please correct me if I am wrong.

• Reply • Share ›

typing.. AlienOnEarth • 2 years ago

this is not foldable, actually the question is asking whether root->left and root->right are mirror image of each other or not..

• Reply • Share ›

Harjit Singh • 2 years ago

bool mirror(struct node*root1,struct node*root2)

{

if(root1==NULL&& root2==NULL)

return true;

if(root1==NULL||root2==NULL)

return false;

return(mirror(root1->left,root2->right)&&mirror(root1->right,root2->left));

}

/* Returns true if the given tree is foldable */

bool isFoldable(struct node *root)

{

if(root==NULL)

return true;

return(mirror(root->left,root->right));

}

• Reply • Share ›

Guest • 2 years ago

bool isStructSame(struct node *a, struct node *b)
{
if (a == NULL && b == NULL)
return true;
if (!(a && b))
return false;
return (true && isStructSame(a->left, b->left) && isStructSame(a->right,b->right));
}

• Reply • Share ›

aspire • 3 years ago

We can solve this problem by putting the left and right trees in queue in preorder fashion. And then comparing their structure while popping them from the queue.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<queue>
using namespace std;
typedef struct tree
{
    int data;
    struct tree *left,*right;
} *Node;

Node newNode(int data)
{
    Node temp = (Node)malloc(sizeof(struct tree));
```

see more

• Reply • Share ›

logic_bomber • 3 years ago

Hope this can be done more easily as follows ... Do drop here if i am wrong

```
bool isFoldableUtil(TreeNode *root1,TreeNode *root2)
{
    if(root1==NULL && root2==NULL)
        return true;
    if(root1==NULL || root2==NULL)
        return false;
    return ((root1->val == root2->val)&&          (isFoldableUtil(root1->left,root
}
bool isFoldable(TreeNode *root) {

    if(root==NULL)
        return true;
    return isFoldableUtil(root->left,root->right);
}
```

• Reply • Share ›

Prashant Agarwal • 3 years ago

We can find the inorder and postorder traversals of the tree and compare both the arrays....
Thats i think easiest way...

• Reply • Share ›

shek8034 → Prashant Agarwal • 3 years ago

This wont work because you have to consider their structure, not their values. If you store the traversals in arrays, then finally you end up with comparing the two arrays, which is wrong since you have to check if the left structure is mirror of right structure or not, not their values. (See the definition of Folded trees).

2 • Reply • Share ›

abhishek08aug • 3 years ago

Intelligent :D

• Reply • Share ›

mrn • 3 years ago

@geeksforgeeks : I think only one of them is sufficient to check.why both?

a) n1->left is mirror of n2->right

a) n1->left is mirror of n2->right
b) n1->right is mirror of n2->left
Here is my code.Plz let me know for what input this code breaks.

```
bool isfoldable(Node *n1,Node *n2)
{
        if(n1==NULL && n2==NULL) return true;
        if(((!n1->l && !n2->r) || (n1->l && n2->r))  &&
           ((!n1->r && !n2->l) || (n1->r && n2->l)) &&
           isfoldable(n1->l,n2->r) && isfoldable(n1->r,n2->l)
           )
        {
                return true;
        }
        return false;
}

In main() : calling isfoldable(root->l,root->r);
```

• Reply • Share ›

Sarvasva Sawhney • 4 years ago
u not using double pointers so there will be no change original tree so no need to correct tree to original tree

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

• Reply • Share ›

seabird • 4 years ago

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

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

int foldable(node *rootl,node *rootr) {
    if(rootl==NULL && rootr==NULL)return 1;
    if(rootl->left==NULL && rootr->right==NULL &&rootr->left==NULL&&rootl->right==NULL
    if(rootl->left==NULL && rootr->right==NULL &&rootr->left!=NULL&&rootl->right!=NULL
    else if(rootl->left!=NULL && rootr->right!=NULL &&rootr->left==NULL&&rootl->right=
    else if(rootl->left!=NULL && rootr->right!=NULL &&rootr->left!=NULL&&rootl->right!
        foldable(rootl->left,rootr->right);
        foldable(rootl->right,rootr->left);
```

see more

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Add Disqus Add Privacy