

Foldable Binary Trees

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

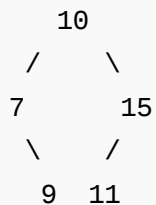
A tree can be folded if left and right subtrees of the tree are structure wise mirror image of each other. An empty tree is considered as foldable.

Consider the below trees:

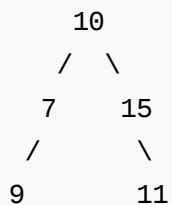
(a) and (b) can be folded.

(c) and (d) cannot be folded.

(a)



(b)



(c)



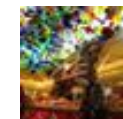
Google™ Custom Search



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

```

      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.

```

#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

```

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](#)

```

/* 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 mirror 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 original 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)
    )

```

Custom market
research at scale.

Get \$75 off

 Google consumer surveys



```

{   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

```

Recent Comments

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

[Backtracking | Set 7 \(Sudoku\)](#) · 43 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

Neha I think that is what it should return as, in...

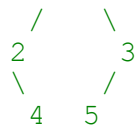
Find depth of the deepest odd level leaf node · 3 hours ago

AdChoices 

[► Binary Tree](#)

[► Java Tree](#)

[► Tree Trees](#)



```

*/
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;
}

```

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

AdChoices

[▶ Tree Structure](#)

[▶ Red Black Tree](#)

[▶ Root Tree](#)

AdChoices

[▶ In Memory Tree](#)

[▶ The Mirror](#)

[▶ Mirror 4](#)

```

#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;
}

```

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.



Explore New Frontiers in Data.

HPCC Systems is Big Data Processing and Analytics
Open Source. Proven. Trusted.

 LexisNexis® [Learn More](#) 

Related Tpoics:

- [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](#)



0

 Tweet

0



0

Writing code in comment? Please use [ideone.com](https://www.ideone.com) and share the link here.

24 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



AlienOnEarth • 4 days 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 provided would be wrong.

1 ^ | v • Reply • Share ›



Harjit Singh • 4 days 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));

}

^ | v · Reply · Share ›
```



Guest · 2 months 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->rig
}

^ | v · Reply · Share ›
```



aspire · 9 months ago

We can solve this problem by putting the left and right trees in queue in preord structure while popping them from the queue.

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

```
typedef struct tree
{
    int data;
    struct tree *left,*right;
} *Node;

Node newNode(int data)
{
    Node temp = (Node)malloc(sizeof(struct tree));
    temp->left=temp->right=NULL;
```

[see more](#)

^ | v • Reply • Share ›



logic_bomber • 11 months 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(
}

bool isFoldable(TreeNode *root) {

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

^ | v • Reply • Share ›



Prashant Agarwal • 11 months ago

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

^ | v • Reply • Share ›



shek8034 ➔ Prashant Agarwal • 11 months ago

This wont work because you have to consider their structure, not their arrays, then finally you end up with comparing the two arrays, which is left structure is mirror of right structure or not, not their values. (See the

1 ^ | v • Reply • Share ›



abhishek08aug • a year ago

Intelligent :D

^ | v • Reply • Share ›



mrn • a year ago

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

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 false;
}

```

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

^ | v • Reply • Share ›



Sarvasva Sawhney • 2 years ago

u not using double pointers so there will be no change original tree so no need

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

^ | v • Reply • Share ›



seabird • 2 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&&r
    if(rootl->left==NULL && rootr->right==NULL &&rootr->left!=NULL&&r
    else if(rootl->left!=NULL && rootr->right!=NULL &&rootr->left==NUI

```

```
foldable(rootl->left,rootr->right);
foldable(rootl->right,rootr->left);
```

[see more](#)

^ | v • Reply • Share ›



Sourabh Goyal • 2 years ago

I think in the first method, in the function isstructsame() there should be

```
if ( a != NULL && b != NULL &&
    isStructSame(a->left, b->right) &&
    isStructSame(a->right, b->left)
    ) /*b->right and b->left are interchanged*/
```

^ | v • Reply • Share ›



hemant • 2 years ago

check if right subtree and left subtree calls are equal in number

```
void foldable(struct node* root, int L, int R )
{
    if(root == NULL)
        return;

    foldable(root->left,L+1,R);
    foldable_left+= L;
    foldable_right+= R;
    foldable(root->right,L,R+1);
}
```

void main()

```
-
foldable(root,0,0);

if(foldable_left == foldable_right)
printf("BST is a foldable BST\n");
else
printf("BST is not a foldable BST\n");

return;
}
```

^ | v • Reply • Share ›



hemant → hemant • 2 years ago

in the code above:

foldable_left and foldable_right are global variables initialised to 0

^ | v • Reply • Share ›



lekho00 • 4 years ago

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
int isfoldable(struct node *,struct node*);
struct node* newNode(int );
struct node
{int info;
struct node* left;
struct node* right;
};
void main()
{
struct node * root;
root=NULL;
root = newNode(1);
```

```
root->left      = newNode(2);
root->right      = newNode(3);
root->right->left = newNode(4);
```

[see more](#)

^ | v • Reply • Share ›



donbosio → lekho00 • 4 years ago

good..!!

^ | v • Reply • Share ›



Shekhu • 4 years ago

Your approach would print "Not foldable" for below tree, but it is foldable.

```
      10
     /  \
    7    15
   \    /
  9   11
```

When the pointer comes to node with value 7, it sees that right is not NULL and is the case with node of value 5. Correct me if I am wrong.

^ | v • Reply • Share ›



dev → Shekhu • 4 years ago

I think we can eliminate the steps of mirroring the left subtree and then condition like this inside the isStructureSame():

```
if ( a != NULL && b != NULL &&
    isStructSame(a->left, b->right) &&
    isStructSame(a->right, b->left)
    )
```




smilinglyqing → Shekhu · 4 years ago

as long as change the following part:

```
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; }
```

to be

```
if (n1 == NULL && n2 == NULL)
    { return true; }

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

^ | v · Reply · Share ›



kartik → smilinglyqing · 4 years ago

@smilinglyqing,

They both are same. Presence of "return" statement inside "if"

^ | v • Reply • Share ›



Dzmitry Huba → Shekhu • 4 years ago

Nope, it correctly detects that it is a foldable tree as it won't work with 7 time but rather 7th left and 15th right.

^ | v • Reply • Share ›



GeeksforGeeks → Dzmitry Huba • 4 years ago

@Dzmitry Huba: Thanks for suggesting a mew method. We ha

^ | v • Reply • Share ›



geek4u → Dzmitry Huba • 4 years ago

I concur with @Dzmitry Huba, his solution works fine, and is ef

^ | v • Reply • Share ›



Dzmitry Huba • 4 years ago

We can avoid tree modification costs by making the following recursive algorit
N1, R1) and (L2, N2, R2) where L and R denote left and right subtrees respec
if they have mirror structure we need to recursively check whether pairs (L1, F
find code in C# below:

```
[sourcecode language="CSharp"]
static bool IsFoldable<T>(TreeNode<T> root)
where T: IComparable<T>
{
    if (root == null)
        return true;
    return IsFoldable(root.Left, root.Right);
}

static bool IsFoldable<T>(TreeNode<T> n1, TreeNode<T> n2)
where T : IComparable<T>
{
```

```
if (n1 == null && n2 == null)
```

```
return true;
```

```
if (n1 == null || n2 == null)
```

```
return false;
```

```
return IsFoldable(n1.Left, n2.Right) && IsFoldable(n1.Right, n2.Left);
```

```
}
```

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site