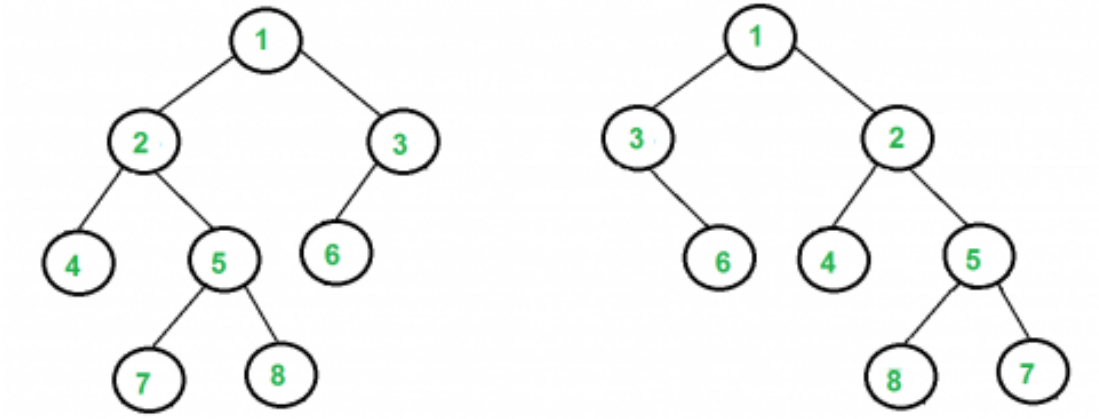
Tree Isomorphism Problem

Write a function to detect if two trees are isomorphic. Two trees are called isomorphic if one of them can be obtained from other by a series of flips, i.e. by swapping left and right children of a number of nodes. Any number of nodes at any level can have their children swapped. Two empty trees are isomorphic.

For example, following two trees are isomorphic with following sub-trees flipped: 2 and 3, NULL and 6, 7 and 8.

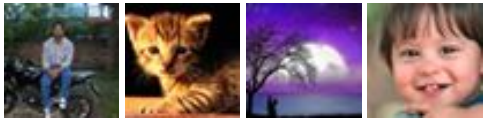


We simultaneously traverse both trees. Let the current internal nodes of two trees being traversed be **n1** and **n2** respectively. There are following two conditions for subtrees rooted with n1 and n2 to be isomorphic.

1) Data of n1 and n2 is same.

2) One of the following two is true for children of n1 and n2

……**a)** Left child of n1 is isomorphic to left child of n2 and right child of n1 is isomorphic to right child of n2.



……**b)** Left child of n1 is isomorphic to right child of n2 and right child of n1 is isomorphic to left child of n2.

```
// A C++ program to check if two given trees are isomorphic
#include <iostream>
using namespace std;

/* A binary tree node has data, pointer to left and right children */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Given a binary tree, print its nodes in reverse level order */
bool isIsomorphic(node* n1, node *n2)
{
 // Both roots are NULL, trees isomorphic by definition
 if (n1 == NULL && n2 == NULL)
    return true;

 // Exactly one of the n1 and n2 is NULL, trees not isomorphic
 if (n1 == NULL || n2 == NULL)
    return false;

 if (n1->data != n2->data)
    return false;

 // There are two possible cases for n1 and n2 to be isomorphic
 // Case 1: The subtrees rooted at these nodes have NOT been "Flipped"
 // Both of these subtrees have to be isomorphic, hence the &&
 // Case 2: The subtrees rooted at these nodes have been "Flipped"
 return
 (isIsomorphic(n1->left,n2->left) && isIsomorphic(n1->right,n2->right)
 (isIsomorphic(n1->left,n2->right) && isIsomorphic(n1->right,n2->left)
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
node* newNode(int data)
{
    node* temp = new node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
```


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

```
    return (temp);
}

/* Driver program to test above functions*/
int main()
{
    // Let us create trees shown in above diagram
    struct node *n1 = newNode(1);
    n1->left        = newNode(2);
    n1->right       = newNode(3);
    n1->left->left  = newNode(4);
    n1->left->right = newNode(5);
    n1->right->left  = newNode(6);
    n1->left->right->left = newNode(7);
    n1->left->right->right = newNode(8);

    struct node *n2 = newNode(1);
    n2->left         = newNode(3);
    n2->right        = newNode(2);
    n2->right->left   = newNode(4);
    n2->right->right   = newNode(5);
    n2->left->right   = newNode(6);
    n2->right->right->left = newNode(8);
    n2->right->right->right = newNode(7);

    if (isIsomorphic(n1, n2) == true)
       cout << "Yes";
    else
      cout << "No";

    return 0;
}
```

Output:

```
Yes
```

Time Complexity: The above solution does a traversal of both trees. So time complexity is O(m + n) where m and n are number of nodes in given trees.

This article is contributed by **Ciphe**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks

main page and help other Geeks.

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

Writing code in comment? Please use **ideone.com** and share the link here.

21 Comments GeeksforGeeks

Sort by Newest

Recent Comments

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

Backtracking | Set 7 (Sudoku) · 31 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 51 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 51 minutes 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 · 2 hours ago

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

Find depth of the deepest odd level leaf node · 2 hours ago

Join the discussion…

curiousCoder • a month ago

@GeeksforGeeks i am not sure abt its complexity but , how abt

1) A level order traversal of both the trees followed by
2)sorting of each level's nodes, for both the trees,
if yields same result implies, trees are isomorphic..

In shaa Allah..

and dis wud take O(N) time for level order, followed by O(h*xlogx) for each lev
no. of nodes in any level)

it will use auxiliary space of O(m+n)

Reply • Share ›

sumit • 5 months ago

how the time complexity of above solution is O(m+n) in worst case... ? we trav
combination

Reply • Share ›

sumit sumit • 5 months ago

plz someone explain about its time complexity ... thanks...

Reply • Share ›

GeometryMonkey • 6 months ago

I need to check this, but I think the algorithm is O(n^2), not O(n+m). Let T(n) be
complexity on a tree with n nodes. Assume the trees are complete and both ha
require O(1) time and the return statement calls isIsomorphic four times, each
T(n/2) + O(1). This recurrence is equal to Sum_{i=1}^{log n} 4^i, which is rougl

thus T(n) = O(n^2).

Reply • Share ›

annonymoe ➦ GeometryMonkey • 6 months ago

or if taking into the account the number of comparisons, each node wil
n!=m return false
else it will be 2^(logn-1) 2^(log n - 1) comparisons in the worst case the
approximately O(n^2)

Reply • Share ›

prakash ➦ GeometryMonkey • 6 months ago

well said.this exactly what i think.

Reply • Share ›

smrite • 8 months ago

complexity of the code is o(m+n) .. can you tell me a case where two tress ar
elements ?

Reply • Share ›

GeometryMonkey ➦ smrite • 5 months ago

Two trees cannot be isomorphic with a different number of elements.

Reply • Share ›

Amit Bgl • 9 months ago

wow code :D

Reply • Share ›

santosh • 9 months ago

cant we do it using traversals algo!

i mean, call traversal for each node(recursive), it will start from leaf node, then
excluding the root

i.e.

call on 5, then store 7,8 in array instead of 7,5,8 and do same for other tree, if

Reply • Share ›

Dijil • 9 months ago

This print codes "YES" even if the tree is not isomorphic

Reply • Share ›

GeometryMonkey Dijil • 5 months ago

Example?

Reply • Share ›

Manisha Barnwal • 10 months ago

Good One!

Reply • Share ›

Name • 10 months ago

Looks like good solution. :)

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

Reply • Share ›

rohit • 11 months ago

There is an another efficient way of doing it. I solved this problem by array repr
different arrays (following the rules). Then start searching from back of the arr
i---> /* Iterator of array1 */
j---> /* iterator of array2 */
First check if array2[j]==array1[i] anywhere theres a hit! then check if array1[i/
Basic idea is that root node has to be equal.

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

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

Reply • Share ›

kk • 11 months ago

i don't think it is O(n+m)...
it is O(2^(m+n))
because at each node you have to choices (l-l ,r-r) or (l-r, r-l)
l-l means in both the tree we go to left.

Reply • Share ›

Rahul kk • 11 months ago

@kk

But the code covers this case of (l-l && r-r) || (l-r && r-l) in a single pas

Look carefully at this line :
return
(isIsomorphic(n1->left,n2->left) && isIsomorphic(n1->right,n2->right))||
(isIsomorphic(n1->left,n2->right) && isIsomorphic(n1->right,n2->left));

its a single return covering the two possibilities

Reply • Share ›

Maddy Rahul • 8 months ago

it still calls the recursion twice .. even i think it'll be O(2^(m+n))
whether to compare its child with the same orientation or flip it.
this one runs in exponential

Reply • Share ›

abhishek08aug • 11 months ago

Intelligent :D

• Reply • Share ›

C Programming Techies • a year ago

what is the practical application of isomorphism of trees?

2 • Reply • Share ›

GeometryMonkey C Programming Techies • 5 months ago

Apparently one of the most important applications is natural language p
is parsed into a tree structure, and tree isomorphism is used to figure
parts of speech its words are. Ref: http://stackoverflow.com/quest...

• Reply • Share ›

Subscribe Add Disqus to your site