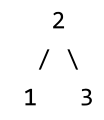
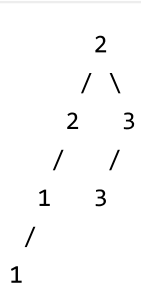
Double Tree

Write a program that converts a given tree to its Double tree. To create Double tree of the given tree, create a new duplicate for each node, and insert the duplicate as the left child of the original node.

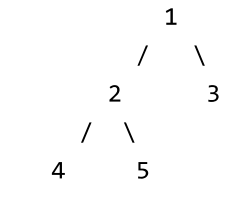
So the tree…



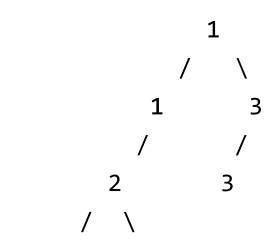
is changed to…



And the tree



is changed to



```
      2     5
     /     /
    4    5
   /
  4
```

Algorithm:

Recursively convert the tree to double tree in postorder fashion. For each node, first convert the left subtree of the node, then right subtree, finally create a duplicate node of the node and fix the left child of the node and left child of left child.

Implementation:

C

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

/* 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;
};

/* function to create a new node of tree and returns pointer */
struct node* newNode(int data);

/* Function to convert a tree to double tree */
void doubleTree(struct node* node)
{
  struct node* oldLeft;

  if (node==NULL) return;

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

  /* duplicate this node to its left */
  oldLeft = node->left;
  node->left = newNode(node->data);
  node->left->left = oldLeft;
}



/* UTILITY FUNCTIONS TO TEST doubleTree() 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);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
  if (node == NULL)
    return;
  printInorder(node->left);
  printf("%d ", node->data);
  printInorder(node->right);
}


/* Driver program to test above functions*/
int main()
{

  /* Constructed binary tree is
            1
          /   \
        2      3
      /  \
    4     5
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);

  printf("Inorder traversal of the original tree is \n");
  printInorder(root);

  doubleTree(root);

  printf("\n Inorder traversal of the double tree is \n");
  printInorder(root);

  getchar();
  return 0;
}
```

Run on IDE

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;


    /* Function to convert a tree to double tree */
    void doubleTree(Node node) {
        Node oldleft;

        if (node == null) {
            return;
        }

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

        /* duplicate this node to its left */
        oldleft = node.left;
        node.left = new Node(node.data);
        node.left.left = oldleft;
    }

    /* Given a binary tree, print its nodes in inorder*/
    void printInorder(Node node) {
        if (node == null) {
            return;
        }
        printInorder(node.left);
        System.out.print(node.data + " ");
        printInorder(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);

        System.out.println("Original tree is : ");
        tree.printInorder(root);
        tree.doubleTree(root);
        System.out.println("");
        System.out.println("Inorder traversal of double tree is : ");
        tree.printInorder(root);

    }
}
```

Run on IDE

Time Complexity: O(n) where n is the number of nodes in the tree.

References:

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

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

79 Comments Category: Trees

Related Posts:

- Locking and Unlocking of Resources arranged in the form of n-ary Tree
- Find all possible binary trees with given Inorder Traversal
- Find LCA in Binary Tree using RMQ
- Find multiplication of sums of data of leaves at same levels
- Find Count of Single Valued Subtrees
- Check if a given array can represent Preorder Traversal of Binary Search Tree
- Mirror of n-ary Tree
- Succinct Encoding of Binary Tree

(Login to Rate and Mark)

2 Average Difficulty : **2/5.0**
Based on **3** vote(s)

Add to TODO List
Mark as DONE

Like Share One person likes this. Be the first of your friends.

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

79 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Gourav Goswami · a month ago

```
void doubleTree(struct node* root)

{

struct node* newnode;

if(!root) return;

doubleTree(root->left);

doubleTree(root->right);
```

```
newnode=newNode(root->data);

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

{

root->left=newnode;

}
```

see more

• Reply • Share ›

Gourav Goswami • a month ago

```
#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node* left;

struct node* right;

};

struct node* newNode(int data);

void doubleTree(struct node* root)

{
```

see more

• Reply • Share ›

Jatin • 2 months ago

www.coder2design.com

```
private void recursiveDoubleTree(Node<Integer> node){
                if( node == null){
                        return;
                }
                recursiveDoubleTree(node.getLeftChild());
                // New node with same data as node and append to to the right
```

```
                Node<Integer> newNode = new Node<Integer>();
                newNode.setData(node.getData());
                Node<Integer> lNode = node.getLeftChild();
                newNode.setLeftChild(lNode);
                node.setLeftChild(newNode);
                recursiveDoubleTree(node.getRightChild());
        }
```

• Reply • Share ›

Praveen Yadav • 5 months ago

```
#include <iostream>

using namespace std;

typedef struct node{

int data;

node *left;

node *right;

}node;

node *newNode(int new_data)

{

node *new_node=new node;

new_node->data=new_data;
```

see more

• Reply • Share ›

sam • 6 months ago

guys is this solution correct ?

```
#include<bits stdc++.h="">
using namespace std;
#define sp printf(" ");
#define nl printf("\n");

typedef struct node1
{
int data;
struct node1* left;
struct node1* right;
```

} node;

node * newNode(int ele)
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->data=ele;

see more

1 ^ v • Reply • Share ›

Karan Kapoor ➚ sam • 6 months ago

I thought exactly the same..I feel its correct :)

1 ^ v • Reply • Share ›

swapnil • 7 months ago

would pre order be wrong?

algo -

double (node* root)

{if(root==NULL)

return ;

n= newnode(root->data);

n->left=root->left;

root->left=n;

double(root->left->left);

double(root->right);

}

3 ^ v • Reply • Share ›

Billionaire ➚ swapnil • 5 months ago

Both works

1 ^ v • Reply • Share ›

Abhijeet Kaur ➚ swapnil • 6 months ago

same doubt

^ v • Reply • Share ›

Ashish Singh · 7 months ago

http://ideone.com/3o4J93

 · Reply · Share ›

Gaurav · 7 months ago

Inorder will also work fine. Below is the code

```
void doubleTree(Node* root)

{

Node* temp;

if(root==NULL)return;

doubleTree(root->leftChild);

temp=malloc(sizeof(Node));

temp->data=root->data;

temp->leftChild=root->leftChild;

temp->rightChild=NULL;

root->leftChild=temp;

doubleTree(root->rightChild);

}
```

1 · Reply · Share ›

blank space · 7 months ago

https://ideone.com/Naw0UG

 · Reply · Share ›

Avata This comment was deleted.

DS+Algo Guest · 7 months ago

sahi hai khan 25 lac ka package sure hai tera

 · Reply · Share ›

xtreme · 7 months ago

giving wrong output with preorder-

void doublet(struct node *p)

void doublet(struct node *p)

{

if(p==NULL)

return ;

struct node *temp=p->left;

p->left=newNode(p->data);

p=p->left;

p->left=temp;

doublet(p->left);

doublet(p->right);

}

the leaf nodes are not getting doubled.. not working with inorder also..
possibly beacause at leaf the null condition prevents it from getting doubled. is this correct explanation ?

• Reply • Share ›

Ajcoo ➚ xtreme • 7 months ago

ur code is not working because its wrong change it like this

void doublet(struct node *p)

{

if(p==NULL)

return ;

struct node *temp=p->left;

p->left=newNode(p->data);

p->left->left=temp;

doubleTree(temp);

doubleTree(p->right);

}

1 • Reply • Share ›

codex ➚ Ajcoo • 2 months ago

Thanks **@Ajcoo**

^ | v • Reply • Share ›

xtreme ➚ Ajcoo • 7 months ago

got it..thanks mate..

^ | v • Reply • Share ›

Saurabh • 8 months ago

It can be also possible by doing a level order traversal

^ | v • Reply • Share ›

Anup Rai ➚ Saurabh • 7 months ago

Well it can be done in any way where you can traverse all the nodes of a tree. The point being if u can tell which node are original and make there clone and which are new and not make there clone.

In level order the problem may arise to tell the difference. For every node the left one will be duplicate if you are going level by level but for every odd level it will be original.

So you will have to take care of that.

1 ^ | v • Reply • Share ›

Chinmay Prabhakar • 9 months ago

I have a very basic question. Why do we need to use

/* do the subtrees */

doubleTree(node->left);

doubleTree(node->right);

at the beginning itself. Can't it be delayed till the end of function (ie do all the processing and then call the subcases recursively)

^ | v • Reply • Share ›

Krishana [illegible]

Yes, U can do in this way also : https://ideone.com/3M1JCl

^ | v • Reply • Share ›

xtreme ➚ Krishana • 7 months ago

its giving wrong answer..

this is the code..

```
void doublet(struct node *p)

{

if(p==NULL)
```

```
if(p==NULL)

return ;

struct node *temp=p->left;

p->left=newNode(p->data);

p=p->left;

p->left=temp;

doublet(p->left);

doublet(p->right);

}
```


Vineeth Reddy • 9 months ago

This could also be achieved using inorder doubling of tree.

use this function instead of the one given above, you'd get the same tree:

```
void doubleTree(struct node* node)
{
struct node* oldLeft;
if (node==NULL) return;
/* do the subtrees */
/* duplicate this node to its left */
doubleTree(node->left);
oldLeft = node->left;
node->left = newNode(node->data);
node->left->left = oldLeft;
doubleTree(node->right);
}
```

Double tree can be built using Inorder and Postorder doubling of each node. But not with pre-order, it would be an overflow.

xtreme [illegible] y •

its only working in postorder..tried in both ways wrong output

Ajcoo [illegible] •

i think it works in preorder also...

```
void doubleTree(node* root){

if(root==NULL) return ;

node* temp = newNode(root->data);

temp->left= root->left;

root->left= temp;

doubleTree(temp->left);

doubleTree(root->right);

}
```

• Reply • Share ›

xtreme ➚ Ajcoo • 7 months ago

not working in preorder

• Reply • Share ›

Ajcoo ➚ xtreme • 7 months ago

working check again using above function

• Reply • Share ›

xtreme ➚ Ajcoo • 7 months ago

yes your function gives correct output..

but whats going wrong with this one..-

```
void doublet(struct node *p)

{

if(p==NULL)

return ;

struct node *temp=p->left;

p->left=newNode(p->data);

p=p->left;

p->left=temp;

doublet(p->left);
```

doublet(p->right);

}

• Reply • Share ›

Jerry Goyal • 10 months ago

```
void doublet(struct node* root)
{
        if(!root) return;
        doublet(root->left);
        doublet(root->right);

        node* dnode = (node*)malloc(sizeof(node));
        dnode->data = root->data;
        dnode->left=root->left;
        root->left=dnode;
        dnode->right=NULL;
}
```

• Reply • Share ›

surbhijain93 • 10 months ago

void g(struct node *node)

{

struct node *temp;

if (node==NULL || (node->left==NULL && node->right!=NULL))

return;

temp=node->left;

struct node* new=(struct node *)malloc(sizeof(struct node));

new->data=node->data;

node->left=new;

new->left=temp;

new->right=NULL;

g(node->left->left);

```
g(node->right);

}
```

• Reply • Share ›

Aryan Parker •

Your code will not work if there is only one node in tree. It also needs to be duplicated but your code will not do so

1 • Reply • Share ›

surbhijain93 •

Hey, you were right ...it doesnt work when root just right child/children with no left child...
i dont knw why i put this line
(node->left==NULL && node->right!=NULL)
i just remove it, right?

• Reply • Share ›

Abhi • a year ago

@geeksforgeeks we can do this following way...

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
using namespace std;

struct Node{

int data;
Node* left;
Node* right;
};

Node* newNode(int data1)
{

Node* new_node=(Node*)malloc(sizeof(Node));
new_node->data=data1;
new_node->left=NULL;
```

see more

1 • Reply • Share ›

Narendra • a year ago

Using Preorder:

```
int doubleTree(struct node *t)

{

if(!t) return 0;

struct node *l=t->left;

t->left = newNode(t->data);

t->left->left=l;

doubleTree(l);

doubleTree(t->right);

}
```


The_Geek · a year ago

I solved it like this.
t=newNode(node->data);
oldLeft = node->left;
node->left=t;
t->left =doubleTree(oldLeft);
doubleTree(node->right);
return node;

1 · Reply · Share ›

irresolute · a year ago

http://ideone.com/fork/N1eLzU

· Reply · Share ›

rihansh · a year ago

http://ideone.com/6I7kdg

· Reply · Share ›

guesT · a year ago

is it fine?
oldleft =newnode(node->data)
oldleft->left=node->left
node->left=oldleft

· Reply · Share ›

riahsnh guesT · a year ago

yeah it is totally fine then call the same procedure for the oldleft->left and node-

>right. Look at my approach it is same

• Reply • Share ›

sanki • a year ago

can be do this question in preorder fashion ?

• Reply • Share ›

irresolute ➚ sanki • a year ago

yupp

• Reply • Share ›

Pooja Arora • a year ago

I think, method void doubleTree(struct node* node) should have use call by reference ie. struct node ** node. As changes made by recurssive calls will not update our tree.

• Reply • Share ›

rihansh ➚ Pooja Arora • a year ago

it is not neccessary because each time we can gurantee that the root of the new tree / double tree will not change or the updated like will be reflected in the tree as we are changing content. i hope you got the point

• Reply • Share ›

Jun • 2 years ago

I think we can use preorder or inorder also....no need to do it specificallly through postorder???isn't it???

1 • Reply • Share ›

DS+Algo ➚ Jun • 2 years ago

How can we do this in preorder, I think we need to change children first....Explain ur logic

• Reply • Share ›

Jun ➚ DS+Algo • 2 years ago

All the 3 traversals work equally good,,,check the codes

preorder

http://ideone.com/Is8BYI

Inorder

http://ideone.com/EAPxpy

2 • Reply • Share ›

Kaushal Varshney Jun • 7 months ago

yes all traversals ll work

• Reply • Share ›

irresolute Jun • a year ago

i haven't checked the inorder but preorder works better as compared to postorder

• Reply • Share ›

<HoldOnLife!#> Jun • 2 years ago

yes but post order is preferred as space complexity is more with preorder and inorder I think .. **@GeeksforGeeks** Please clear it out

• Reply • Share ›

Load more comments