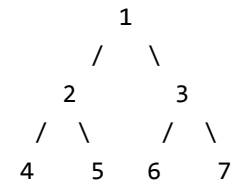
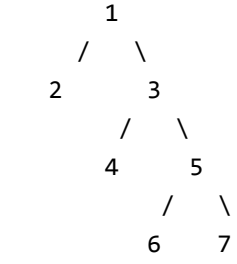
Construct Full Binary Tree from given preorder and postorder traversals

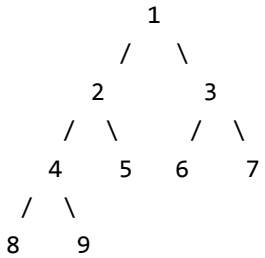
Given two arrays that represent preorder and postorder traversals of a full binary tree, construct the binary tree.

A **Full Binary Tree** is a binary tree where every node has either 0 or 2 children

Following are examples of Full Trees.







It is not possible to construct a general Binary Tree from preorder and postorder traversals (See this). But if know that the Binary Tree is Full, we can construct the tree without ambiguity. Let us understand this with the

help of following example.

Let us consider the two given arrays as pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7} and post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
In pre[], the leftmost element is root of tree. Since the tree is full and array size is more than 1. The value next to 1 in pre[], must be left child of root. So we know 1 is root and 2 is left child. How to find the all nodes in left subtree? We know 2 is root of all nodes in left subtree. All nodes before 2 in post[] must be in left subtree. Now we know 1 is root, elements {8, 9, 4, 5, 2} are in left subtree, and the elements {6, 7, 3} are in right subtree.

```
                1
              /   \
             /     \
    {8, 9, 4, 5, 2}    {6, 7, 3}
```

We recursively follow the above approach and get the following tree.

```
          1
        /   \
       2     3
     /  \   /  \
    4    5 6    7
   / \
  8   9
```

```c
/* program for construction of full binary tree */
#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;
};

// A utility function to create a node
struct node* newNode (int data)
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[] and post[].
// preIndex is used to keep track of index in pre[].
// l is low index and h is high index for the current subarray in post[]
struct node* constructTreeUtil (int pre[], int post[], int* preIndex,
                                int l, int h, int size)
{
```

```
    // Base case
    if (*preIndex >= size || l > h)
        return NULL;

    // The first node in preorder traversal is root. So take the node at
    // preIndex from preorder and make it root, and increment preIndex
    struct node* root = newNode ( pre[*preIndex] );
    ++*preIndex;

    // If the current subarry has only one element, no need to recur
    if (l == h)
        return root;

    // Search the next element of pre[] in post[]
    int i;
    for (i = l; i <= h; ++i)
        if (pre[*preIndex] == post[i])
            break;

    // Use the index of element found in postorder to divide postorder array in
    // two parts. Left subtree and right subtree
    if (i <= h)
    {
        root->left = constructTreeUtil (pre, post, preIndex, l, i, size);
        root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);
    }

    return root;
}

// The main function to construct Full Binary Tree from given preorder and
// postorder traversals. This function mainly uses constructTreeUtil()
struct node *constructTree (int pre[], int post[], int size)
{
    int preIndex = 0;
    return constructTreeUtil (pre, post, &preIndex, 0, size - 1, size);
}

// A utility function to print inorder traversal of a Binary Tree
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 ()
{
    int pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7};
    int post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
    int size = sizeof( pre ) / sizeof( pre[0] );

    struct node *root = constructTree(pre, post, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

Run on IDE

Output:

```
Inorder traversal of the constructed tree:
8 4 9 2 5 1 6 3 7
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

59 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?

(Login to Rate and Mark)

4 Average Difficulty : **4/5.0**
Based on **9** vote(s)

Add to TODO List

Mark as DONE

Like Share 19 people like this.

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

59 Comments GeeksforGeeks 1 Login

Recommend 6 Share Sort by Newest

Join the discussion…

Victor • 4 months ago

I think in problem statement it should be strict binary tree not full binary tree.
Since second example 1,2,4,8,9,5,3,6,7 is strict binary tree but not full binary tree.

http://web.cecs.pdx.edu/~shear...

Please correct me.

• Reply • Share ›

Huzaifa Mahmood Victor • 3 months ago

It is a full binary tree:
Every parent has either two children or none.

• Reply • Share ›

Aritra Ghosh • 4 months ago

The solution given above does not work if there are duplicate elements.

int pre[] = {1, 2, 2, 3,4,5, 6};

int post[] = {2 , 3, 2, 5, 6, 4, 1};

The result for above case is Inorder traversal of the constructed tree:

2 1 3 2 5 4 6

whereas the post order of constructed tree is not same as the given tree.
The inorder traversal should have been 2 2 3 1 5 4 6.

• Reply • Share ›

mars Aritra Ghosh • 3 months ago

please note that actually trees are used for indexing and the values in nodes represent a "key" of index. And obviously index contains unique keys. Thus no question of duplicate values.

• Reply • Share ›

QILI • 8 months ago

root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);

Should from i+i to h-1

Correct me if I wrong

• Reply • Share ›

TulsiRam • 8 months ago

root->right = constructTreeUtil (pre, post, preIndex, i + 1, :h", size);

In this line it is not needed to pass "H", we should pass "H-1" because postOrder[H] we have already used as root of the current tree.

3 • Reply • Share ›

TulsiRam ➚ TulsiRam • 8 months ago

Correct me I wrong please

1 • Reply • Share ›

cc4 • 8 months ago

Hey Guys!! here is the o(n) approach. I have modified the Construct util function such that it works in o(n). Do reply on this thread if you find anything wrong.

http://ideone.com/9o5JHh

• Reply • Share ›

Rachit Nagdev • 9 months ago

i think l > h condition will never execute, if pre and post orders are correct.
Correct me if i am wrong

• Reply • Share ›

user007 ➚ Rachit Nagdev • 8 months ago

Yea, even I think it won't be used anytime!

• Reply • Share ›

Guest • 9 months ago

// Search the next element of pre[] in post[]
int i;
for (i = l; i <= h; ++i)
if (pre[*preIndex] == post[i])

In Post Order List, why are you not starting the search from 0th element? You are always starting from 1st element. Wha happens ifI have a tree with tree only two node (e.g., 1 is the root and 2 is the left child)?

• Reply • Share ›

TulsiRam ➚ Guest • 8 months ago
Bhai it is not "i=1" it is "i=L" L for love
^ | v • Reply • Share ›

Vinay Shivanna • a year ago
I have written a simple recursive function in JAVA, let me know if it works fine for you guys as well.

```
//initial call
//preArracy: array for preorder representation
//postArray: array for postorder representation
public static Node constructFullBinaryTree(int[] preArray, int[] postArray){
Node root = constructFullBinaryTree(preArray, postArray, 0);
return root;
}

//prelow : index of preArray array.
private static Node constructFullBinaryTree(int[] preArray, int[] postArray, preLow) {
if (preLow > preArray.length-1 ) return null;
else
{
//find the position of the specific element in postArray
int indexPost = find(postArray, preArray[preLow]);
```

see more

^ | v • Reply • Share ›

guest • a year ago
root->right should be constructTreeUtil (pre, post, preIndex, i + 1, h-1, size)
not constructTreeUtil (pre, post, preIndex, i + 1, h, size)
4 ^ | v • Reply • Share ›

Gaurav Arora ➚ guest • 8 months ago
yes technically it should be.

But every new node is extracted from the pre array, so extra nodes in post array aren't making any difference.
if (*preIndex >= size)
return NULL;
2 ^ | v • Reply • Share ›

DS+Algo ➚ guest • 10 months ago
But both are working.

• Reply • Share ›

zxcve • a year ago

How it is suppose to work if we have duplicates ?
Is searching from right to left in postorder array right condition to take care for duplicates as the left and right nodes will be more closer to root which is always the last element of the postorder subarray passed ?

1 • Reply • Share ›

zxcve zxcve • a year ago

Code for taking care of repeated values. Kindly review.

```
node *create_tree(int *pre_arr, int *post_arr, int &pre_index, int l, int r, int size)

{

int i = 0;

node *tmp = new node;

tmp->data = pre_arr[pre_index++];

tmp->left = NULL;

tmp->right = NULL;

if (pre_index < size)

{

for (i = r; i >= l; i--)
```

see more

• Reply • Share ›

zxcve zxcve • a year ago

Ok it seems if left and right node are of same value then my code has a problem. I need to check that condition too.

• Reply • Share ›

Neha Nigam • a year ago

```
struct node *constructTree (int pre[], int post[], int size)
{
int preIndex = 0;
return constructTreeUtil (pre, post, &preIndex, 0, size - 2, size);
}

int h = size - 2 in starting 2
```

isn't n = size -2 in starting ?

sree_ec • a year ago

```
tree_int* ConvertPrePostArraysToTree(int pre[],int post[],int poststrt,int postlen,int* preind)
{

int temp;
tree_int* mytree=NULL;
int i=0;

if(poststrt > postlen )
{
return NULL;

}

temp = pre[*preind];
mytree = (tree_int*)newNode(temp);
temp = pre[(*preind)+1];
(*preind)++;
for(i=poststrt;i<postlen;i++) {="" if(temp="=" post[i]){="" mytree-="">left =
ConvertPrePostArraysToTree(pre,post,poststrt,i,preind);
mytree->right = ConvertPrePostArraysToTree(pre,post,i+1,postlen,preind);
return mytree;
}
}

return mytree;

}
```


ryan • 2 years ago

@GeeksforGeeks
there must be
h=h-1;
before for loop

Avata This comment was deleted.

DS+Algo Guest • 9 months ago

reason bhi likh, samjha

Rajeev • 2 years ago

//utility function to get index for the left and right subtree
int searchIndex(int post[],int pre[],int k,int size,int start,int end)

{

int i;

for(i=0;i<size;i++) if(post[i]="=k)" break;="" int="" val="post[i-1];" for(i="start;i<end;i++)" if(pre[i]="=val)" return="" i;="" }="" to="" create="" full="" binary="" tree="" tree="" *constructtree(int="" pre[],int="" post[],int="" start,int="" end,int="" size)="" {="" if(start="">=end)

return NULL;

tree *root = newNode(pre[start]);

if(start==end-1)

return root;

int index = searchIndex(post,pre,pre[start],size,start,end);

root->left = constructTree(pre,post,start+1,index,size);

root->right = constructTree(pre,post,index,end,size);

return root;

}

  • Reply • Share ›

prashant jha • 2 years ago

since u have to make cmplete binary tree so u must have to open two recursive calls for each nodes...the left child will obviously be the next index of preorder and right child index can be found by getting the key that occur just before the node in postorder and find its index in the preorder

#include<iostream>
using namespace std;
struct tnode
{
tnode* lchild;
int data;
tnode* rchild;
tnode(int d)

```
{
lchild=NULL;
data=d;
rchild=NULL;
}
```

see more

• Reply • Share ›

Tapan Avasthi • 2 years ago

```
struct node * buildCompletePrePostTree(int *Pe, int sPe, int ePe, int *Po, int sPo, int ePo){

if(sPe>ePe || sPo>ePo || Pe==NULL || Po==NULL)

return NULL;

struct node *root=newNode(Pe[sPe]);

if(sPe==ePe || sPo==ePo) //size is 1

return root;//subtree doesn't have a child node

root->left=buildCompletePrePostTree(Pe,sPe+1,searchIndex(Pe,sPe,ePe,Po[ePo-
1])-1,Po,sPo,searchIndex(Po,sPo,ePo,Pe[sPe+1]));

root->right=buildCompletePrePostTree(Pe, searchIndex(Pe,sPe,ePe,Po[ePo-1]),ePe, Po,
searchIndex(Po,sPo,ePo,Pe[sPe+1])+1,ePo-1);

return root;
```

see more

• Reply • Share ›

Sriharsha g.r.v • 2 years ago

we hav used array "pre" and searched the corresponding value in "pos"
and then solved the problem.the other is way is
possible and its simple to analyse with above logic,
i.e use "pos" and searched the corresponding value in "pre"

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

/* A binary tree node has data, pointer to left child
and a pointer to right child */
int preIndex;
struct node
```

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

struct node* newNode (int data)
```

see more

• Reply • Share ›

Guest • 2 years ago

here we hav used array "pre" and searched the corresponding value in "pos" and then solved the problem.can any one help if the other is way is possible.

i actually made this change
initialising preorder with n-1 and

```
if (i <= h && i>=0)
{
root->right = constructTreeUtil (pre, post, i , h, size);
root->left = constructTreeUtil (pre, post, l, i-1, size);

}
```

explanation with code is highly appreciated.thanq

• Reply • Share ›

Guest • 2 years ago

here we hav used array pre and searched the corresponding value in pos and then solved the problem.can any one help if the other is way is possible.
explanation with code is highly appreciated.thanq

• Reply • Share ›

prakash • 3 years ago

root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);
in this statement upper bound arg for post array must be (h-1) not h. hecause u must exclude the root value in post order array which is already created

• Reply • Share ›

guest ➚ prakash • a year ago

I agree.

• Reply • Share ›

Nilesh Agrawal • 3 years ago

Great Example

• Reply • Share ›

Sarthak Mall 'shanky' • 3 years ago

okk..got it .. I was seeing it as perfect binary tree by mistake... :)

1 • Reply • Share ›

GeeksforGeeks • 3 years ago

The definition in fact seems to be matching with Wikipedia. The wiki page says "A full binary tree (sometimes proper binary tree or 2-tree or strictly binary tree) is a tree in which every node other than the leaves has two children. Or, perhaps more clearly, every node in a binary tree has exactly (strictly) 0 or 2 children. Sometimes a full tree is ambiguously defined as a perfect tree (see next). Physicists define a binary tree to mean a full binary tree"

• Reply • Share ›

Sarthak Mall 'shanky' • 3 years ago

First of all your def of full binary tree is incomplete and last 2 examples are not full binary tree....plz refer http://en.wikipedia.org/wiki/B....

• Reply • Share ›

Emie Al-Ansi • 3 years ago

thank you so0o0 much^^

• Reply • Share ›

jayant • 3 years ago

```
int pree[9] = {1, 2, 4, 8, 9, 5, 3, 6, 7};
  int post[9] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
  int curr=0,pos=0,len=9;



  node* fullbt()
  {
      int i;
      node *root=(node*)malloc(sizeof(node));
      root->data=pree[pos];
      root->left=NULL;
      root->right=NULL;

      for(i=0;i<len;i++)
      if(post[i]==pree[pos])break;

      pos++;
```

see more

• Reply • Share ›

sap • 3 years ago

```
node constructfullbinary(int pre[],int post[],int i,int j,int p,int q){
        int t;
        if(i>j||p>q)
        return NULL;
        node temp=getnode(pre[i]);
        if(i==j)
        return temp;
        for(t=p;;t++)
        {
                if(pre[i+1]==post[t])
                break;
        }

        temp->left=constructfullbinary(pre,post,i+1,i+1+t-p,p,t);
        temp->right=constructfullbinary(pre,post,i+1+t-p+1,j,t+1,q-1);


return temp;
}
```

• Reply • Share ›

4m7u1 • 3 years ago

if pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7} and post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
when we do the firt recurison,
root => 1
root->left={8,9,4,5,2}
and using the below function
root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);
root->right should be 6,7,3,1 right? as h= size-1 which points to post[8]=0.... can anyone clarify this for me??

• Reply • Share ›

abhishek08aug • 3 years ago

Intelligent :D

• Reply • Share ›

Deepa Kumari • 3 years ago

nice.....

• Reply • Share ›

Viky • 3 years ago

I think we can get rid of if(l<h) condition as we are already comparing it in the beginning ..

• Reply • Share ›

Sreenivas Doosa ➦ Viky • 3 years ago

Hey Viky,

Please have a closer look. In the second condition, it is not 'l' it is 'i'

• Reply • Share ›

ravik • 3 years ago

Please check this and correct me if i am wrong.

```
 /* Paste your code here (You may delete these lines if not writing code) */
struct node *construct(int pre[], int post[], int start, int end, int index)
{
        int i;
        struct node *temp;
        if(start>end)
        {
                index--;
                return NULL;
        }
        temp = newNode(post[end]);
        for( i = end - 1; i >= start; i--)
                if(pre[index] == post[i])
                        break;
        temp->left      = construct(pre, post, start, i, ++index);
        temp->right     = construct(pre, post, i+1, end-1, ++index);
        return temp;
}
```

• Reply • Share ›

Priyank • 3 years ago

For populating the right child of a node, the endIndex for post[] should be h -1 instead of h

```
root->right = constructTreeUtil (pre, post, preIndex, i + 1, h -1, size);
```

The last element in post[] will always be the root itself, hence we need to exclude it while populating its own right child.

• Reply • Share ›

Nitin • 3 years ago

I think if you see here:
root->left = constructTreeUtil (pre, post, preIndex, l, i, size);
root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);
For first time, preIndex will have the value '0' which will be updated to '1'.
Here, you are passing index '1' for both left and right child.
So, in their respective recursion, root will be '1' and both of children will have '2' as value.

//struct node* root = newNode (pre[*preIndex]);

I think, we need to do something so that for left child, index '1'(value = 2) will be passed and for right child, index '6'(value = 3) will be passed.

Please, correct me if I'm wrong.
I don't know much about c++ but Java.
So, I could have missed something here.

• Reply • Share ›

Abhishek ➚ Nitin • 3 years ago

By the time, preIndex would be passed to the second call, preindex would have got incremented from 2.
Remember, the first call is a recursion call. It will make other constructTreeUtil calls thus incrementing preIndex, covering all the nodes under the left subtree.

root->left = constructTreeUtil (pre, post, preIndex, l, i, size);
root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);

• Reply • Share ›

Palash • 3 years ago

Time complexity seems to be O(n^2), worst case.

1 • Reply • Share ›

neha • 3 years ago

i should start from 0;
for (i = 0; i <= h; ++i)

• Reply • Share ›

Load more comments