GeeksforGeeks
A computer science portal for geeks

GeeksQuiz

Login

Home Algorithms DS GATE Interview Corner Q&A C C++ Java Books Contribute Ask a Q About

Array Bit Magic C/C++ Articles GFacts Linked List MCQ Misc Output String Tree Graph

Check whether a given Binary Tree is Complete or not

Given a Binary Tree, write a function to check whether the given Binary Tree is Complete Binary Tree or not.

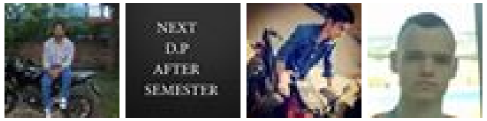
A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. See following examples.

```
The following trees are examples of Complete Binary Trees
    1
  /   \
 2     3

       1
    /    \
   2       3
  /
 4


       1
    /    \
   2      3
  /  \    /
 4    5  6
```

```
The following trees are examples of Non-Complete Binary Trees
    1
      \
```



Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

```
      3

      1
   /    \
  2       3
   \     /  \
    4   5    6

      1
   /    \
  2      3
        /  \
       4    5
```

Source: Write an algorithm to check if a tree is complete binary tree or not

The method 2 of level order traversal post can be easily modified to check whether a tree is Complete or not. To understand the approach, let us first define a term ‘Full Node’. A node is ‘Full Node’ if both left and right children are not empty (or not NULL).
The approach is to do a level order traversal starting from root. In the traversal, once a node is found which is NOT a Full Node, all the following nodes must be leaf nodes.
Also, one more thing needs to be checked to handle the below case: If a node has empty left child, then the right child must be empty.

```
   1
 /   \
2     3
 \
  4
```

Thanks to Guddu Sharma for suggesting this simple and efficient approach.

```
// A program to check if a given binary tree is complete or not
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_Q_SIZE 500
```

Geometric Algorithms

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;
};

/* frunction prototypes for functions needed for Queue data
   structure. A queue is needed for level order tarversal */
struct node** createQueue(int *, int *);
void enQueue(struct node **, int *, struct node *);
struct node *deQueue(struct node **, int *);
bool isQueueEmpty(int *front, int *rear);
```

```
/* Given a binary tree, return true if the tree is complete
   else false */
bool isCompleteBT(struct node* root)
{
  // Base Case: An empty tree is complete Binary Tree
  if (root == NULL)
    return true;

  // Create an empty queue
  int rear, front;
  struct node **queue = createQueue(&front, &rear);

  // Create a flag variable which will be set true
  // when a non full node is seen
  bool flag = false;

  // Do level order traversal using queue.
  enQueue(queue, &rear, root);
  while(!isQueueEmpty(&front, &rear))
  {
    struct node *temp_node = deQueue(queue, &front);

    /* Ceck if left child is present*/
    if(temp_node->left)
    {
       // If we have seen a non full node, and we see a node
       // with non-empty left child, then the given tree is not
       // a complete Binary Tree
       if (flag == true)
         return false;
```

```
       enQueue(queue, &rear, temp_node->left);  // Enqueue Left Child
    }
    else // If this a non-full node, set the flag as true
       flag = true;

    /* Ceck if right child is present*/
    if(temp_node->right)
    {
       // If we have seen a non full node, and we see a node
       // with non-empty left child, then the given tree is not
       // a complete Binary Tree
       if(flag == true)
         return false;

       enQueue(queue, &rear, temp_node->right);  // Enqueue Right Chil
    }
    else // If this a non-full node, set the flag as true
       flag = true;
  }

  // If we reach here, then the tree is complete Bianry Tree
  return true;
}


/*UTILITY FUNCTIONS*/
struct node** createQueue(int *front, int *rear)
{
  struct node **queue =
   (struct node **)malloc(sizeof(struct node*)*MAX_Q_SIZE);

  *front = *rear = 0;
  return queue;
}

void enQueue(struct node **queue, int *rear, struct node *new_node)
{
  queue[*rear] = new_node;
  (*rear)++;
}

struct node *deQueue(struct node **queue, int *front)
{
  (*front)++;
  return queue[*front - 1];
}
```

Recent Comments

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

Backtracking | Set 7 (Sudoku) · 35 minutes ago

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

Flipkart Interview | Set 6 · 55 minutes ago

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

Software Engineering Lab, Samsung Interview | Set 2 · 55 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 · 3 hours ago

```
bool isQueueEmpty(int *front, int *rear)
{
   return (*rear == *front);
}

/* 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 above functions*/
int main()
{
   /* Let us construct the following Binary Tree which
      is not a complete Binary Tree
            1
          /   \
         2     3
        / \     \
       4   5     6
    */

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

  if ( isCompleteBT(root) == true )
      printf ("Complete Binary Tree");
  else
      printf ("NOT Complete Binary Tree");

  return 0;
}
```

Output:

```
NOT Complete Binary Tree
```

Time Complexity: O(n) where n is the number of nodes in given Binary Tree

Auxiliary Space: O(n) for queue.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.

Better Than Hadoop.
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

6 Tweet 0 0

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

37 Comments GeeksforGeeks

Sort by Newest ▾

Join the discussion…

AlienOnEarth • 18 days ago

@Geeksforgeeks:

Easy and recursive solution.

```
int isNotCompleteBT(struct node *root)

{

if(root == NULL)

return 0;

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

{
return 1;

}
return 0;
}
```

^ | ˅ • Reply • Share ›

master ➦ AlienOnEarth • 12 days ago

Lets say a tree has a root and 2 child nodes. These 2 child nodes have
But your code will return true.

^ | ⌄ • Reply • Share ›

AlienOnEarth ➦ master • 10 days ago

@master, thank you for the comment. The updated code:

```
bool isCompleteBT(struct node *root)

{

if(root == NULL)

return true;

if(((root->left != NULL && root->right == NULL) && root->left ->l

/*

1

/

2

/
```

see more

^ | ⌄ • Reply • Share ›

ISha • a month ago

Different approach: We count the number of nodes in the tree and then check
they had to be put in an array, where if root has index i then index of its left chil
be 2*i+2.

Code:

```
#include <iostream>

#include <stdlib.h>

using namespace std;

struct btree {

int data;

struct btree *left;

struct btree *right;

};
```

see more

˄ | ˅ • Reply • Share ›

Mohaan Raja • 2 months ago

```
#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;
};

/* 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;
```

see more

Reply • Share ›

lokesh921 • 4 months ago

Another approach.
Do a reverse in-order traversal. Note the height of the first leaf in this traversal
If ((the height of other leaves is more than the max_height) || (a node has only

else the tree is complete

Reply • Share ›

Vivek • 5 months ago

im

Reply • Share ›

xiaoguangye • 6 months ago

I have one Time O(n), Space O(lgn) solution. Please let me know if it flaws.
A: Get height of tree, can check if (!node->left && node->right) as well.

B: Allocate int * level = malloc(height * sizeof(int)) to count nodes on each leve
Only allow deepest level to have none 2^n number of nodes

C: Use preorder traversal to check the second deepest level nodes. Every one need to be leaf.

If the tree passes above three tests, it is a complete tree.

Reply • Share ›

Vivek • 6 months ago

O(n) solution without using any extra space.

please go through this solution.

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

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

int max(int a, int b)
{
        return a>b?a:b;
}
int height(struct node *root)
{
```

see more

1 ^ | ⌄ • Reply • Share ›

Jayanth ➦ Vivek • 6 months ago

1
/ \
2 3
/ \
4 5

ur code returns true for above tree which isn't a complete tree

Edit : The tree is not clear in the diagram...
the tree desc is
root = 1
1->left = 2;
1->right = 3;

2->left = 2->right = NULL;

3->left = 4;
3->right = 5;

⌃ | ⌄ • Reply • Share ›

Vivek ➦ Jayanth • 5 months ago

my code does check this condition.
if (height (root->left)<height(root->right))
then its not a complete binary tree

⌃ | ⌄ • Reply • Share ›

ISha ➦ Vivek • a month ago

It is giving incorrect results for this tree:

struct node *root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->left->left->left = newNode(6);

root->left->left->right = newNode(7);

root->right->left = newNode(8);

^ | ⌄ • Reply • Share ›

digiter • 6 months ago

How about this one?

```
#include <cmath>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <algorithm>

#include <iostream>

#include <map>

#include <set>
```

see more

^ | ⌄ • Reply • Share ›

Olivier • 7 months ago

Hi,
Just wondering if this code will work too.. I think it should:

```
public boolean isComplete(Node<integer> node) {
       if (node == null) return true;
```

```
        if (node.left == null && node.right != null) return false;
        int leftHeight = 0;
        int rightHeight = 0;
        leftHeight = node.left != null ? tree.height(node.left) : 0;
        rightHeight = node.right != null ? tree.height(node.right) : (
        if( rightHeight > leftHeight)return false; //right height can
        if( leftHeight > 0 && rightHeight > 0){    //leftheight and r:
            if( (leftHeight-rightHeight) > 1)return false;
        }else{
            if( leftHeight>1 || rightHeight > 1 )return false;
        }

        return isComplete(node.left) && isComplete(node.right);

    }
```

^ | ˅ • Reply • Share ›

pavansrinivas • 7 months ago

code in java...

```
boolean isComplete(){

                Node temp = root;
                boolean isFirstLeaf = false;
                 Queue<node> Q = new LinkedList<>();
                    Q.add(root);
                    while (!Q.isEmpty())
                    {
                        temp = Q.remove();
                        if(temp.left==null&&temp.right!=null){
                                return false;
}
                        if(temp.left==null&&temp.right==null){
                                isFirstLeaf = true;
                        }
```

see more

^ | ⌄ • Reply • Share ›

draganwarrior • 7 months ago

can we do as follows

http://ideone.com/SNXRqe

^ | ⌄ • Reply • Share ›

rajeevprasanna • 7 months ago

Why can't we simply check if level before the last level is completely filled or n

1) Calculate height(h) of the tree
2) Count number of node at level h-1
3) Check if node count is equal to 2 power(h-1)
if matches, it is complete tree otherwise not.

Let me know if there are any flaws in this approach.

^ | ⌄ • Reply • Share ›

Sriharsha g.r.v ➦ rajeevprasanna • 6 months ago

u aproach fals for thiscase

1
/ \
2 3
/
4

i mean the child should be towards left..that criteria is missing in ur alg

⌃ | ⌄ • Reply • Share ›

Guest ➦ rajeevprasanna • 7 months ago

need to have a check for following cases..

1
/ \
2 3
\
4

⌃ | ⌄ • Reply • Share ›

xiaoguangye ➦ Guest • 6 months ago

this can be checked by if (!node->left && node->right).
I think you mean:
1
/\
2 3
/ /
4 5

⌃ | ⌄ • Reply • Share ›

xiaoguangye ➦ xiaoguangye • 6 months ago

add:
4). check every node on the second deepest level. ever
to be leaf.

⌃ | ⌄ • Reply • Share ›

Trilok Sharma • 10 months ago

/* c++ version */

```
#include
#include
using namespace std;

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

struct node* newNode(int data)
{
struct node* node = new(struct node);
node->data = data;
node->left = NULL;
node->right = NULL;
```

see more

Reply • Share ›

Saurabh Tamrakar • 10 months ago

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
struct node *leftnext;.
int data;
struct node *rightnext;.
};
int check(struct node *ps);.
```

```
void append(struct node **, int);.
void inorder(struct node *);.
void postorder(struct node *);.
void preorder(struct node *);.
int leaf(struct node *ps);.
int size(struct node *ps); //Number of Elemens in Tree.
int main()
{
```

see more

⌃ | ⌄ • Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

⌃ | ⌄ • Reply • Share ›

FAISAL • 2 years ago

Time complexity :O(n)

⌃ | ⌄ • Reply • Share ›

FAISAL • 2 years ago

```
[sourcecode language="C++"]
/* #include<iostream>
using namespace std;
class Node
{
public:
int data;
Node* left;
Node* right;
Node(int d,Node* l = 0,Node* r = 0)
{
```

```
data = d;
right = r;
left = l;
}
};
bool check(Node* root1,Node* root2)
{
```

see more

^ | ˅ • Reply • Share ›

atul007 • 2 years ago

Time complexity = O(n)
space complexity = O(1)
int flag=0;
ht=height(root);
call : CheckComplete(root,ht-1,&flag)
if return 1 -> Complete Binary Tree
if return 0 -> NOT Complete Binary Tree

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

int height(node *root)
{
int l=0,r=0;

        if(!root)
                return 0;
        l=height(root->left);
        r=height(root->right);
```

see more

Reply · Share ›

mrn ➦ atul007 · 8 months ago

stack implicitly takes O(n) space ..

⌃ | ⌄ · Reply · Share ›

White Tiger · 2 years ago

```
void checkCompleteTree(struct BSTnode* root)
{
int level=0,count=0,flag=0,temp_flag=0;
struct BSTnode* temp;
insert(root);
while(!isQueueEmpty())
{
temp=extract();
if(flag==0)
{
if(temp->left!=NULL && temp->right!=NULL)
{
insert(temp->left);
insert(temp->right);
}
else if(temp->right==NULL)
{
if(temp->left!=NULL)
```

see more

⌃ | ⌄ · Reply · Share ›

lohith · 2 years ago

```
[sourcecode language="C++"]
#include<iostream>
```

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

typedef struct node * Node;

Node newNode(int val)
{
Node temp = new node;
temp->left = NULL;
temp->right = NULL;
temp->value = val;
```

see more

^ | ˅ • Reply • Share ›

Lakshmanan ➦ lohith • 2 years ago

This algorithm fails for the following case... Counting the balance betw
them can't be complete (either if then can be full / only one of them sho
criteria used in ur approach)...

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

```
thunder:7% g++ iscomplete.C
thunder:8% ./a.out
yes
thunder:9% cat iscomplete.C
#include<iostream>
using namespace std;
```

see more

^ | ˅ • Reply • Share ›

BackBencher ➦ Lakshmanan • a year ago

@Lakshmanan and lohit:

Can u please expalain algo, i am bit confused in code logic. Please reply ASAP..

^ | ˅ • Reply • Share ›

lohith ➦ Lakshmanan • 2 years ago

Yeah. I dint notice. Thanks.

^ | ˅ • Reply • Share ›

deep ➦ lohith • 2 years ago

@lohith
i tried for many tries ur program is running well

^ | ˅ • Reply • Share ›

deep ➦ lohith • 2 years ago

@lohith
i tried for many tries ur program is running well

```
 /* Paste your code here (You may delete these lines if not wr:
```

Reply • Share ›

lohith lohith • 2 years ago

[sourcecode language=""]
A simple check, (left sub-tree value - right sub-tree value) <2 &&
(left sub-tree value - right sub-tree value)>=0
at each node will be sufficient to decide if a tree is complete or not.

Reply • Share ›

lohith lohith • 2 years ago

A simple check, (left sub-tree value - right sub-tree value) =0 at each n
is complete or not.

Reply • Share ›

Subscribe Add Disqus to your site