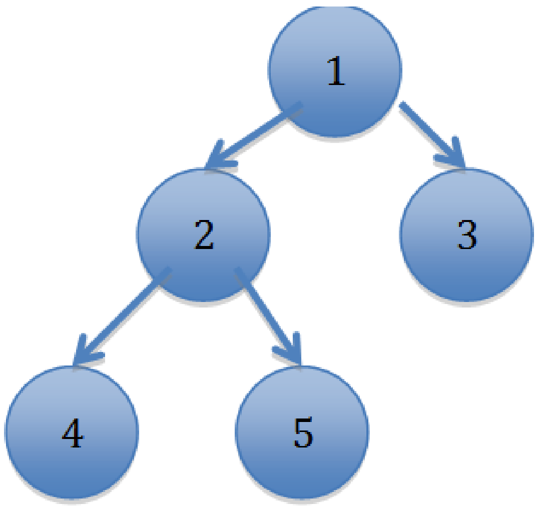
Check whether a binary tree is a complete tree or not | Set 2 (Recursive Solution)

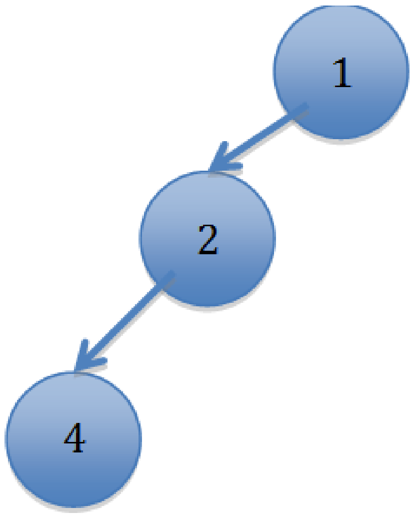
A complete binary tree is a binary tree whose all levels except the last level are completely filled and all the leaves in the last level are all to the left side. More information about complete binary trees can be found here.

For Example:-

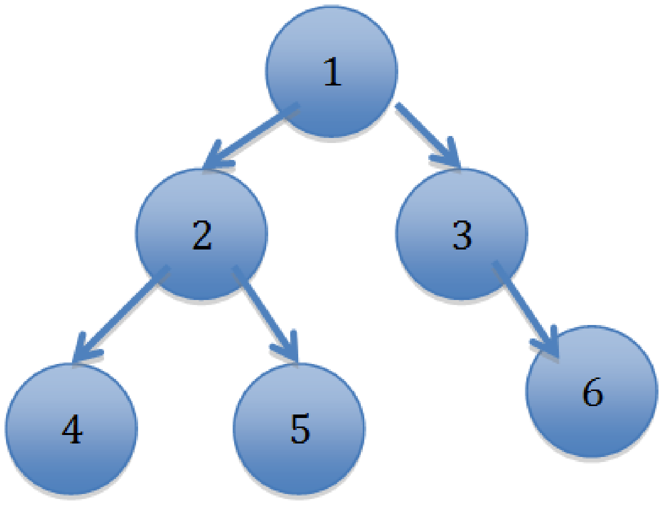
Below tree is a Complete Binary Tree (All nodes till the second last nodes are filled and all leaves are to the left side)



Below tree is not a Complete Binary Tree (The second level is not completely filled)



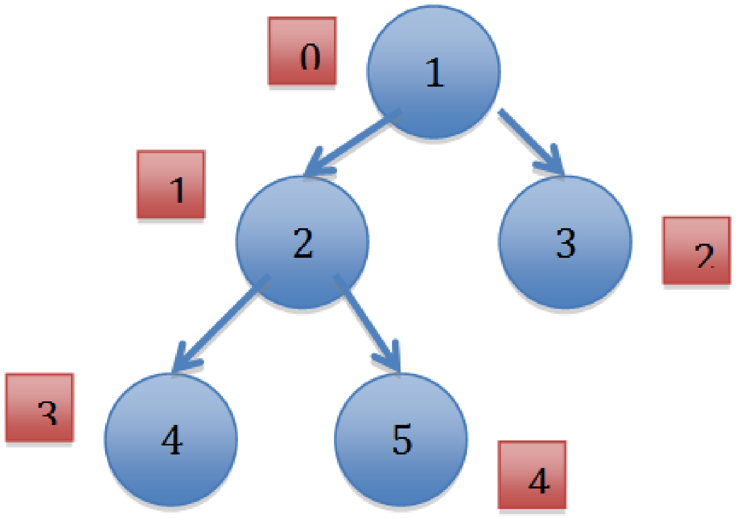
Below tree is not a Complete Binary Tree (All the leaves are not aligned to the left. The left child node of node with data 3 is empty while the right child node is non-empty).



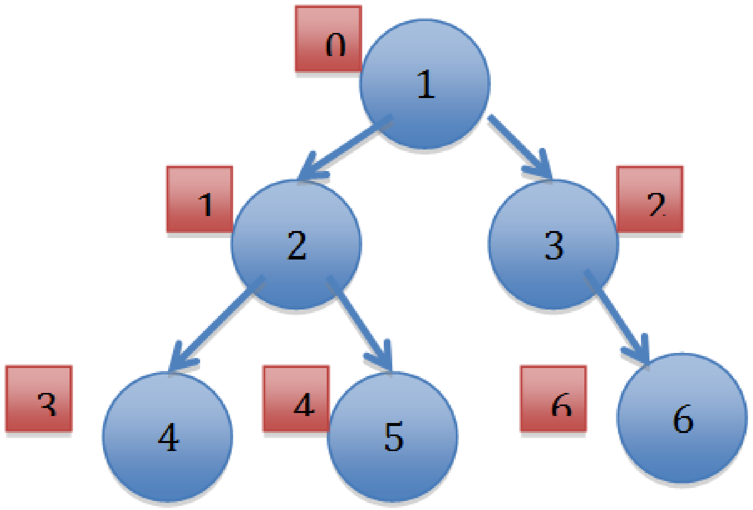
An iterative solution for this problem is discussed in below post.
Check whether a given Binary Tree is Complete or not | Set 1 (Using Level Order Traversal)

In this post a recursive solution is discussed.

In the array representation of a binary tree, if the parent node is assigned an index of ‘i’ and left child gets assigned an index of ‘2*i + 1’ while the right child is assigned an index of ‘2*i + 2’. If we represent the binary tree below as an array with the respective indices assigned to the different nodes of the tree below are shown below:-



As can be seen from the above figure, the assigned indices in case of a complete binary tree will strictly less be than the number of nodes in the complete binary tree. Below is the example of non-complete binary tree with the assigned array indices. As can be seen the assigned indices are equal to the number of nodes in the binary tree. Hence this tree is not a complete binary tree.



Hence we proceed in the following manner in order to check if the binary tree is complete binary tree.

1. Calculate the number of nodes (count) in the binary tree.
2. Start recursion of the binary tree from the root node of the binary tree with index (i) being set as 0 and the number of nodes in the binary (count).
3. If the current node under examination is NULL, then the tree is a complete binary tree. Return true.
4. If index (i) of the current node is greater than or equal to the number of nodes in the binary tree (count) i.e. (i>= count), then the tree is not a complete binary. Return false.
5. Recursively check the left and right sub-trees of the binary tree for same condition. For the left sub-tree use the index as (2*i + 1) while for the right sub-tree use the index as (2*i + 2).

The time complexity of the above algorithm is O(n). Following is the code for checking if a binary tree is a complete binary tree.

C

```
/* C program to checks if a binary tree complete ot not */
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

/*  Tree node structure */
struct Node
{
    int key;
    struct Node *left, *right;
};

/* Helper function that allocates a new node with the
   given key and NULL left and right pointer. */
struct Node *newNode(char k)
{
    struct Node *node = (struct Node*)malloc(sizeof(struct Node));
    node->key = k;
    node->right = node->left = NULL;
    return node;
}

/* This function counts the number of nodes in a binary tree */
unsigned int countNodes(struct Node* root)
{
    if (root == NULL)
        return (0);
    return (1 + countNodes(root->left) + countNodes(root->right));
}

/* This function checks if the binary tree is complete or not */
bool isComplete (struct Node* root, unsigned int index,
                 unsigned int number_nodes)
{
    // An empty tree is complete
    if (root == NULL)
        return (true);

    // If index assigned to current node is more than
    // number of nodes in tree, then tree is not complete
    if (index >= number_nodes)
        return (false);

    // Recur for left and right subtrees
    return (isComplete(root->left, 2*index + 1, number_nodes) &&
            isComplete(root->right, 2*index + 2, number_nodes));
}
```

```
// Driver program
int main()
{
    // Le us create tree in the last diagram above
    struct Node* root = NULL;
    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);

    unsigned int node_count = countNodes(root);
    unsigned int index = 0;

    if (isComplete(root, index, node_count))
        printf("The Binary Tree is complete\n");
    else
        printf("The Binary Tree is not complete\n");
    return (0);
}
```

Java

```
// Java program to check if binay tree is complete or not
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /* This function counts the number of nodes in a binary tree */
    int countNodes(Node node) {
        if (node == null) {
            return (0);
        }
        return (1 + countNodes(node.left) + countNodes(node.right));
    }
    /* This function checks if the binary tree is complete or not */
    boolean isComplete(Node node, int index, int number_nodes)
    {
        // An empty tree is complete
        if (node == null)
           return true;

        // If index assigned to current node is more than
        // number of nodes in tree, then tree is not complete
        if (index >= number_nodes)
           return false;

        // Recur for left and right subtrees
        return (isComplete(node.left, 2 * index + 1, number_nodes)
            && isComplete(node.right, 2 * index + 2, number_nodes));

    }

    // Driver program
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        Node NewRoot = null;
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.left.left = new Node(4);
        tree.root.right.right = new Node(6);

        int node_count = tree.countNodes(root);
        int index = 0;

        if (tree.isComplete(root, index, node_count))
            System.out.print("The binary tree is complete");
        else
            System.out.print("The binary tree is not complete");
    }
}

// This code is contributed by Mayank Jaiswal
```

Python

```
# Python program to check if a binary tree complete or not

# Tree node structure
class Node:
    # Contructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None


# This function counts the number of nodes in a binary tree
def countNodes(root):
    if root is None:
        return 0
    return (1+ countNodes(root.left) + countNodes(root.right))

# This function checks if binary tree is complete or not
def isComplete(root, index, number_nodes):

    # An empty is complete
    if root is None:
        return True

    # If index assigned to current nodes is more than
    # number of nodes in tree, then tree is not complete
    if index >= number_nodes :
        return False

    # Recur for left and right subtress
    return (isComplete(root.left , 2*index+1 , number_nodes)
        and isComplete(root.right, 2*index+2, number_nodes)
          )

# Driver Program

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.right = Node(6)

node_count = countNodes(root)
index = 0

if isComplete(root, index, node_count):
    print "The Binary Tree is complete"
else:
    print "The Binary Tree is not complete"

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
The Binary Tree is not complete
```

This article is contributed by **Gaurav Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Leave a comment 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?

Like Share 19 people like this.

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