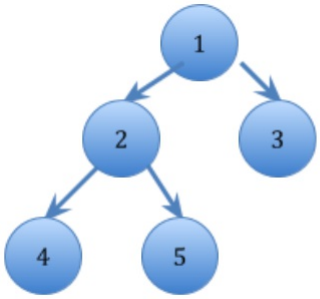
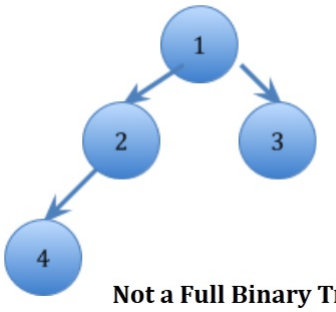
Check whether a binary tree is a full binary tree or not

A full binary tree is defined as a binary tree in which all nodes have either zero or two child nodes. Conversely, there is no node in a full binary tree, which has one child node. More information about full binary trees can be found here.

For Example:



Full Binary Tree



Not a Full Binary Tree

We strongly recommend to minimize your browser and try this yourself first.

To check whether a binary tree is a full binary tree we need to test the following cases:-

1) If a binary tree node is NULL then it is a full binary tree.
2) If a binary tree node does have empty left and right sub-trees, then it is a full binary tree by definition
3) If a binary tree node has left and right sub-trees, then it is a part of a full binary tree by definition. In this case recursively check if the left and right sub-trees are also binary trees themselves.
4) In all other combinations of right and left sub-trees, the binary tree is not a full binary tree.

Following is the implementation for checking if a binary tree is a full binary tree.

C

```c
// C program to check whether a given Binary Tree is full or 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 tests if a binary tree is a full binary tree. */
bool isFullTree (struct Node* root)
{
    // If empty tree
    if (root == NULL)
        return true;

    // If leaf node
    if (root->left == NULL && root->right == NULL)
        return true;

    // If both left and right are not NULL, and left & right subtrees
    // are full
    if ((root->left) && (root->right))
        return (isFullTree(root->left) && isFullTree(root->right));

    // We reach here when none of the above if conditions work
    return false;
}

// Driver Program
int main()
{
    struct Node* root = NULL;
    root = newNode(10);
    root->left = newNode(20);
    root->right = newNode(30);

    root->left->right = newNode(40);
    root->left->left = newNode(50);
    root->right->left = newNode(60);
    root->right->right = newNode(70);

    root->left->left->left = newNode(80);
    root->left->left->right = newNode(90);
    root->left->right->left = newNode(80);
    root->left->right->right = newNode(90);
    root->right->left->left = newNode(80);
    root->right->left->right = newNode(90);
    root->right->right->left = newNode(80);
    root->right->right->right = newNode(90);

    if (isFullTree(root))
        printf("The Binary Tree is full\n");
    else
        printf("The Binary Tree is not full\n");

    return(0);
}
```

Java

```
// Java program to check if binay tree is full or not

/*  Tree node structure */
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    /* this function checks if a binary tree is full or not */
    boolean isFullTree(Node node)
    {
        // if empty tree
        if(node == null)
        return true;

        // if leaf node
        if(node.left == null && node.right == null )
            return true;

        // if both left and right subtrees are not null
        // the are full
        if((node.left!=null) && (node.right!=null))
            return (isFullTree(node.left) && isFullTree(node.right));

        // if none work
        return false;
    }


    // Driver program
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(20);
        tree.root.right = new Node(30);
        tree.root.left.right = new Node(40);
        tree.root.left.left = new Node(50);
        tree.root.right.left = new Node(60);
        tree.root.left.left.left = new Node(80);
        tree.root.right.right = new Node(70);
        tree.root.left.left.right = new Node(90);
        tree.root.left.right.left = new Node(80);
        tree.root.left.right.right = new Node(90);
        tree.root.right.left.left = new Node(80);
        tree.root.right.left.right = new Node(90);
        tree.root.right.right.left = new Node(80);
        tree.root.right.right.right = new Node(90);

        if(tree.isFullTree(tree.root))
            System.out.print("The binary tree is full");
        else
            System.out.print("The binary tree is not full");
    }
}

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

Python

```
# Python program to check whether given Binary tree is full or not

# Tree node structure
class Node:

    # Constructor of the node class for creating the node
    def __init__(self , key):
        self.key = key
        self.left = None
        self.right = None

# Checks if the binary tree is full or not
def isFullTree(root):

    # If empty tree
    if root is None:
        return True

    # If leaf node
    if root.left is None and root.right is None:
        return True

    # If both left and right subtress are not None and
    # left and right subtress are full
    if root.left is not None and root.right is not None:
        return (isFullTree(root.left) and isFullTree(root.right))

    # We reach here when none of the above if condiitions work
    return False

# Driver Program
root = Node(10);
root.left = Node(20);
root.right = Node(30);

root.left.right = Node(40);
root.left.left = Node(50);
root.right.left = Node(60);
root.right.right = Node(70);

root.left.left.left = Node(80);
root.left.left.right = Node(90);
root.left.right.left = Node(80);
root.left.right.right = Node(90);
root.right.left.left = Node(80);
root.right.left.right = Node(90);
root.right.right.left = Node(80);
root.right.right.right = Node(90);

if isFullTree(root):
    print "The Binary tree is full"
else:
    print "Binary tree is not full"

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

Output:

```
The Binary Tree is full
```

Time complexity of the above code is O(n) where n is number of nodes in given binary tree.