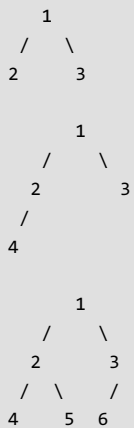


Check whether a given Binary Tree is Complete or not | Set 1 (Iterative Solution)

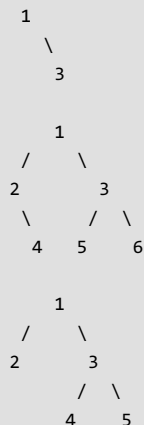
Given a Binary Tree, write a function to check whether the given Binary Tree is Complete Binary Tree or not.

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



The following trees are examples of Non-Complete Binary Trees

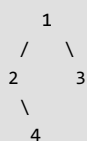


Source: [Write an algorithm to check if a tree is complete binary tree or not](#)

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.



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

C

```
// 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

/* 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 prototypes for functions needed for Queue data
   structure. A queue is needed for level order traversal */
struct node** createQueue(int *, int *);
void enqueue(struct node **, int *, struct node *);
struct node *dequeue(struct node **, int *);
bool isEmptyQueue(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(!isEmptyQueue(&front, &rear))
    {
        struct node *temp_node = dequeue(queue, &front);

        /* Check 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;

        /* Check 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 Child
        }
        else // If this a non-full node, set the flag as true
            flag = true;
    }

    // If we reach here, then the tree is complete Binary 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];
}

bool isEmptyQueue(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;
}

```

Python

```
# Check whether binary tree is complete or not

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Given a binary tree, return true if the tree is complete
# else return false
def isCompleteBT(root):

    # Base Case: An empty tree is complete Binary tree
    if root is None:
        return True

    # Create an empty queue
    queue = []

    # Create a flag variable which will be set True
    # when a non full node is seen
    flag = False

    # Do level order traversal using queue
    queue.append(root)
    while(len(queue) > 0):
        tempNode = queue.pop(0) # Dequeue

        # Check if left child is present
        if (tempNode.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 left child
            queue.append(tempNode.left)

            # If this a non-full node, set the flag as true
        else:
            flag = True

        # Check if right child is present
        if(tempNode.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 BT
            if flag == True:
                return False

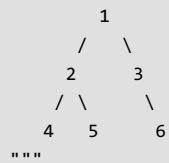
            # Enqueue right child
            queue.append(tempNode.right)

        # If this is non-full node, set the flag as True
    else:
        flag = True

    # If we reach here, then the tree is complete BT
    return True

# Driver program to test above function
```

```
""" Let us construct the following Binary Tree which
    is not a complete Binary Tree
```



```
"""
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)

if (isCompleteBT(root)):
    print "Complete Binary Tree"
else:
    print "NOT Complete Binary Tree"

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

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.