# 101. Symmetric Tree

April 14, 2016 | 298.6K views

Average Rating: 4.72 (190 votes)

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree `[1,2,2,3,4,4,3]` is symmetric:

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

But the following `[1,2,2,null,3,null,3]` is not:

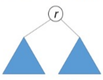
```
    1
   / \
  2   2
   \   \
   3    3
```

**Follow up:** Solve it both recursively and iteratively.

## Solution

### Approach 1: Recursive

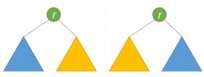
A tree is symmetric if the left subtree is a mirror reflection of the right subtree.



Therefore, the question is: when are two trees a mirror reflection of each other?

Two trees are a mirror reflection of each other if:

1. Their two roots have the same value.
2. The right subtree of each tree is a mirror reflection of the left subtree of the other tree.



This is like a person looking at a mirror. The reflection in the mirror has the same head, but the reflection's right arm corresponds to the actual person's left arm, and vice versa.

The explanation above translates naturally to a recursive function as follows.

Java

```java
public boolean isSymmetric(TreeNode root) {
    return isMirror(root, root);
}

public boolean isMirror(TreeNode t1, TreeNode t2) {
    if (t1 == null && t2 == null) return true;
    if (t1 == null || t2 == null) return false;
    return (t1.val == t2.val)
        && isMirror(t1.right, t2.left)
        && isMirror(t1.left, t2.right);
}
```

**Complexity Analysis**

- Time complexity : $O(n)$. Because we traverse the entire input tree once, the total run time is $O(n)$, where $n$ is the total number of nodes in the tree.
- Space complexity : The number of recursive calls is bound by the height of the tree. In the worst case, the tree is linear and the height is in $O(n)$. Therefore, space complexity due to recursive calls on the stack is $O(n)$ in the worst case.

### Approach 2: Iterative

Instead of recursion, we can also use iteration with the aid of a queue. Each two consecutive nodes in the queue should be equal, and their subtrees a mirror of each other. Initially, the queue contains `root` and `root`. Then the algorithm works similarly to BFS, with some key differences. Each time, two nodes are extracted and their values compared. Then, the right and left children of the two nodes are inserted in the queue in opposite order. The algorithm is done when either the queue is empty, or we detect that the tree is not symmetric (i.e. we pull out two consecutive nodes from the queue that are unequal).

Java

```java
public boolean isSymmetric(TreeNode root) {
    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    q.add(root);
    while (!q.isEmpty()) {
        TreeNode t1 = q.poll();
        TreeNode t2 = q.poll();
        if (t1 == null && t2 == null) continue;
        if (t1 == null || t2 == null) return false;
        if (t1.val != t2.val) return false;
        q.add(t1.left);
        q.add(t2.right);
        q.add(t1.right);
        q.add(t2.left);
    }
    return true;
}
```

**Complexity Analysis**

- Time complexity : $O(n)$. Because we traverse the entire input tree once, the total run time is $O(n)$, where $n$ is the total number of nodes in the tree.
- Space complexity : There is additional space required for the search queue. In the worst case, we have to insert $O(n)$ nodes in the queue. Therefore, space complexity is $O(n)$.

Rate this article:

Previous Next

## Comments: 77

Sort By

Type comment here... (Markdown is supported)

Preview Post

**v1s1on** ★ 499 February 18, 2019 8:37 AM Report

Approach 1 as currently written is doing every comparison twice! This is because isMirror is called with root as the value for both t1 and t2. So we'll end up validating that the left subtree is a mirror of the right subtree but also that the right subtree is a mirror of the left subtree. An easy fix would be to do the null check in the main method and call the recursive method with root->left and root->right. In C++ that would look like this:

Read More

195 Share Reply

SHOW 5 REPLIES

**xwyj5201314** ★ 66 December 10, 2018 3:55 AM

The iterative approach should be slightly changed:

```
if (root == null) {
    return true;
}
```

Read More

66 Share Reply

SHOW 5 REPLIES

**dd2233** ★ 220 June 26, 2018 6:51 PM Report

Approach 3:

Level order traversal, storing each node at the end of an arraylist for that depth.

Then check that each arraylist is symmetric.

Read More

53 Share Reply

SHOW 7 REPLIES

**haruncafer** ★ 38 September 2, 2018 2:48 AM

Aren't we comparing every element twice with the approach 2? We first put [1,1] and then [2,2,2,2] and then [3,3,4,4,3,3,4,4]. Approach 2 two makes every comparison twice.

28 Share Reply

SHOW 5 REPLIES

**digitalnomd** ★ 31 March 20, 2017 7:00 PM

Another approach is to take a preorder and postorder traversal of the tree and compare the lists returned by each. If equal then true, else false.

27 Share Reply

SHOW 7 REPLIES

**terrible_whiteboard** ★ 827 May 19, 2020 8:21 PM

I made a video if anyone is having trouble understanding the solution (clickable link)

https://youtube/3ilpnoulf-bg

[redacted]

Read More

21 Share Reply

SHOW 1 REPLY

**suckcodefuckU** ★ 26 August 7, 2018 2:46 PM Report

The approach 1 waste time.

```
if (r1 == r2 && r1!=null) {
    //it was root
    return isSymmetric(r1.left, r2.right);
```

Read More

14 Share Reply

SHOW 2 REPLIES

**wjwwei** ★ 9 October 23, 2018 5:11 AM

For the second approach, I think the Space complexity should be O(log(n)). Because each time we just need to store one level nodes. And the bottom level nodes is log(n), am I right? Correct me please.

9 Share Reply

SHOW 3 REPLIES

**leetcodedy** ★ 98 November 22, 2018 5:32 AM Report

for the iterative solution, using two stacks to keep track of left/right trees might be more straightforward than using a queue, and using DPS on both sides (but mirrored direction) to update the stacks

7 Share Reply

SHOW 1 REPLY

**dnmange** ★ 17 March 25, 2018 8:08 AM

@noran In iterative approach you could directly add the left and right child of root instead of adding root twice.

6 Share Reply

SHOW 1 REPLY