226. Invert Binary Tree

April 12, 2016 | 259K views

Average Rating: 4.77 (122 votes)

Invert a binary tree.

Example:

Input:

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

Output:

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

Trivia:
This problem was inspired by this original tweet by Max Howell:

> Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so f*** off.

Solution

Approach #1 (Recursive) [Accepted]

This is a classic tree problem that is best-suited for a recursive approach.

Algorithm

The inverse of an empty tree is the empty tree. The inverse of a tree with root r, and subtrees **right** and **left**, is a tree with root r, whose right subtree is the inverse of **left**, and whose left subtree is the inverse of **right**.

Java

```
public TreeNode invertTree(TreeNode root) {
    if (root == null) {
        return null;
    }
    TreeNode right = invertTree(root.right);
    TreeNode left = invertTree(root.left);
    root.left = right;
    root.right = left;
    return root;
}
```

Complexity Analysis

Since each node in the tree is visited only once, the time complexity is $O(n)$, where n is the number of nodes in the tree. We cannot do better than that, since at the very least we have to visit each node to invert it.

Because of recursion, $O(h)$ function calls will be placed on the stack in the worst case, where h is the height of the tree. Because $h \in O(n)$, the space complexity is $O(n)$.

Approach #2 (Iterative) [Accepted]

Alternatively, we can solve the problem iteratively, in a manner similar to breadth-first search.

Algorithm

The idea is that we need to swap the left and right child of all nodes in the tree. So we create a queue to store nodes whose left and right child have not been swapped yet. Initially, only the root is in the queue. As long as the queue is not empty, remove the next node from the queue, swap its children, and add the children to the queue. Null nodes are not added to the queue. Eventually, the queue will be empty and all the children swapped, and we return the original root.

Java

```
public TreeNode invertTree(TreeNode root) {
    if (root == null) return null;
    Queue<TreeNode> queue = new LinkedList<TreeNode>();
    queue.add(root);
    while (!queue.isEmpty()) {
        TreeNode current = queue.poll();
        TreeNode temp = current.left;
        current.left = current.right;
        current.right = temp;
        if (current.left != null) queue.add(current.left);
        if (current.right != null) queue.add(current.right);
    }
    return root;
}
```

Complexity Analysis

Since each node in the tree is visited / added to the queue only once, the time complexity is $O(n)$, where n is the number of nodes in the tree.

Space complexity is $O(n)$, since in the worst case, the queue will contain all nodes in one level of the binary tree. For a full binary tree, the leaf level has $\lceil \frac{n}{2} \rceil = O(n)$ leaves.

Analysis written by: @noran

Rate this article:

Previous Next

Comments: 58 Sort By

Type comment here... (Markdown is supported)

Preview Post

_adji ★ 75 November 18, 2018 6:50 AM Report

Elegant, recursive Python3 solution

```
class Solution:
    def invertTree(self, root):
        if root is None:
```

Read More

65 Share Reply

SHOW 7 REPLIES

qianbinbin ★ 172 April 27, 2019 12:48 PM

Nice solution.
Considering the problem itself, I don't think it's necessary to return a value, as it doesn't require a new copy of the inverted tree.

Read More

32 Share Reply

SHOW 2 REPLIES

anmingyu11 ★ 430 January 29, 2018 11:36 AM Report

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

Read More

30 Share Reply

SHOW 4 REPLIES

divyam ★ 22 November 26, 2018 2:33 PM

For Approach #2 not just BFS simple DFS will work too.
For this problem, all Preorder traversal will work.
Post Order traversal iterative solution using stack:

Read More

17 Share Reply

SHOW 2 REPLIES

wo_stanley ★ 7 March 2, 2018 8:24 AM

public TreeNode invertTree(TreeNode root) {
if (root != null){
TreeNode temp = root.left;
root.left = invertTree(root.right);

Read More

7 Share Reply

SHOW 3 REPLIES

emctwoo ★ 10 April 30, 2019 2:23 AM Report

Pythonic brevity!

```
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        if root:
```

Read More

7 Share Reply

SHOW 1 REPLY

xiaodonng ★ 19 September 17, 2016 8:34 AM

C

```
struct TreeNode* invertTree(struct TreeNode* root) {
    struct TreeNode     *tmp;
    if (root
```

Read More

4 Share Reply

SHOW 1 REPLY

mazhh ★ 3 March 19, 2019 8:44 PM Report

I don't think Approach #1 is faster than #2 in the sense of time complexity as the recursion involves function calling, which need to process return address, etc. And all these overheads increment the time complexity.

However, I compared both the approach in submission, and they both consumes 0 ms. The test case

Read More

3 Share Reply

space_hydra ★ 3 March 17, 2019 3:35 AM Report

Golang

```
func invertTree(root *TreeNode) *TreeNode {
    if root == nil {
        return root
```

Read More

3 Share Reply

huangzixun ★ 5 June 14, 2018 7:47 PM

Its difficult for me to understand the iterative code

3 Share Reply

SHOW 4 REPLIES