**6** 0 0

## 298. Binary Tree Longest Consecutive Sequence

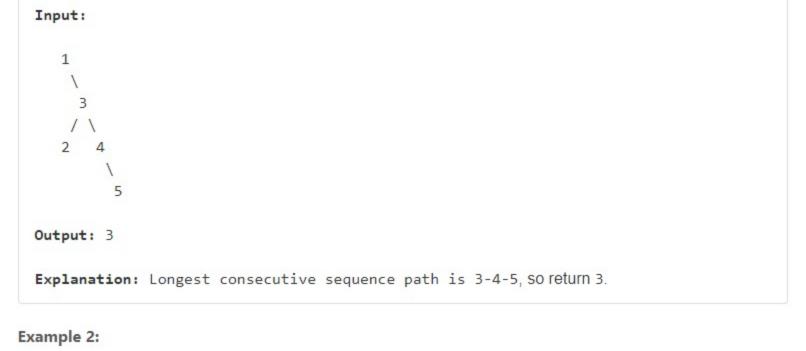
March 5, 2016 | 21.7K views

Average Rating: 4.71 (21 votes)

Given a binary tree, find the length of the longest consecutive sequence path. The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-

child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

### Example 1:



#### Input:

```
2
   2
1
Output: 2
Explanation: Longest consecutive sequence path is 2-3, not 3-2-1, so return 2.
```

### Approach #1 (Top Down Depth-first Search) [Accepted]

Solution

### A top down approach is similar to an in-order traversal. We use a variable length to store the current

#### consecutive path length and pass it down the tree. As we traverse, we compare the current node with its

Algorithm

parent node to determine if it is consecutive. If not, we reset the length. private int maxLength = 0; public int longestConsecutive(TreeNode root) {

```
dfs(root, null, 0);
     return maxLength;
 }
 private void dfs(TreeNode p, TreeNode parent, int length) {
     if (p == null) return;
     length = (parent != null && p.val == parent.val + 1) ? length + 1 : 1;
     maxLength = Math.max(maxLength, length);
     dfs(p.left, p, length);
     dfs(p.right, p, length);
 }
@lightmark presents a neat approach without storing the maxLength as a global variable.
```

return dfs(root, null, 0); }

public int longestConsecutive(TreeNode root) {

Approach #2 (Bottom Up Depth-first Search) [Accepted]

public int longestConsecutive(TreeNode root) {

```
private int dfs(TreeNode p, TreeNode parent, int length) {
      if (p == null) return length;
      length = (parent != null && p.val == parent.val + 1) ? length + 1 : 1;
      return Math.max(length, Math.max(dfs(p.left, p, length),
                                         dfs(p.right, p, length)));
  }
Complexity analysis
  • Time complexity : O(n). The time complexity is the same as an in-order traversal of a binary tree with
```

#### • Space complexity : O(n). The extra space comes from implicit stack space due to recursion. For a skewed binary tree, the recursion could go up to n levels deep.

consecutive path.

private int maxLength = 0;

dfs(root);

n nodes.

- Algorithm

The bottom-up approach is similar to a post-order traversal. We return the consecutive path length starting

at current node to its parent. Then its parent can examine if its node value can be included in this

```
return maxLength;
}
private int dfs(TreeNode p) {
    if (p == null) return 0;
    int L = dfs(p.left) + 1;
    int R = dfs(p.right) + 1;
   if (p.left != null && p.val + 1 != p.left.val) {
       L = 1;
    if (p.right != null && p.val + 1 != p.right.val) {
    int length = Math.max(L, R);
    maxLength = Math.max(maxLength, length);
    return length;
```

# skewed binary tree, the recursion could go up to n levels deep.

O Previous

Complexity analysis

which is O(n).

}

Rate this article: \* \* \* \* \*

• Time complexity : O(n). The time complexity is the same as a post-order traversal in a binary tree,

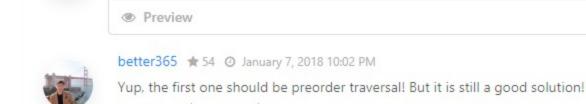
• Space complexity : O(n). The extra space comes from implicit stack space due to recursion. For a

Comments: 14 Sort By ▼

Next **1** 

Post

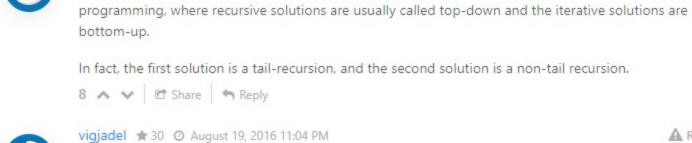
A Report



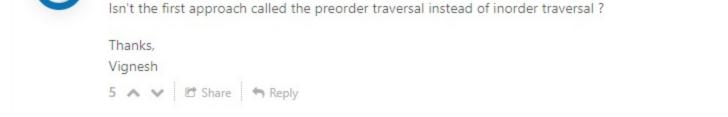
Type comment here... (Markdown is supported)

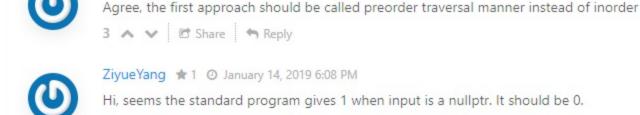
yunshengyao 🛊 53 ② October 15, 2018 2:26 AM

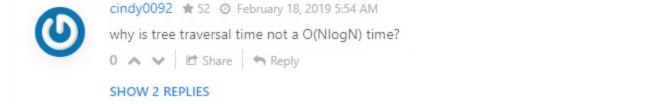
11 A V C Share Share



They difference is not top-down and bottom-up. To me these are the terms used in dynamic

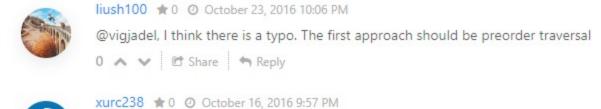








1 A V C Share Reply





Beautiful Solution!