Binary Tree Longest Consecutive Sequence II

Given a binary tree, you need to find the length of Longest Consecutive Path in Binary Tree.

Especially, this path can be either increasing or decreasing. For example, [1,2,3,4] and [4,3,2,1] are both considered valid, but the path [1,2,4,3] is not valid. On the other hand, the path can be in the child-Parent-child order, where not necessarily be parent-child order.

Example 1:

```
Input:
        1
       / \
      2   3
Output: 2
Explanation: The longest consecutive path is [1, 2] or [2, 1].
```

Example 2:

```
Input:
        2
       / \
      1   3
Output: 3
Explanation: The longest consecutive path is [1, 2, 3] or [3, 2, 1].
```

Note: All the values of tree nodes are in the range of [-1e7, 1e7].

Solution 1

```
public class Solution {
    int maxval = 0;
    public int longestConsecutive(TreeNode root) {
        longestPath(root);
        return maxval;
    }
    public int[] longestPath(TreeNode root) {
        if (root == null)
            return new int[] {0,0};
        int inr = 1, dcr = 1;
        if (root.left != null) {
            int[] l = longestPath(root.left);
            if (root.val == root.left.val + 1)
                dcr = l[1] + 1;
            else if (root.val == root.left.val - 1)
                inr = l[0] + 1;
        }
        if (root.right != null) {
            int[] r = longestPath(root.right);
            if (root.val == root.right.val + 1)
                dcr = Math.max(dcr, r[1] + 1);
            else if (root.val == root.right.val - 1)
                inr = Math.max(inr, r[0] + 1);
        }
        maxval = Math.max(maxval, dcr + inr - 1);
        return new int[] {inr, dcr};
    }
}
```

written by vinod23 original link here

Solution 2

```
public class Solution {
    int max = 0;

    class Result {
        TreeNode node;
        int inc;
        int des;
    }

    public int longestConsecutive(TreeNode root) {
        traverse(root);
        return max;
    }

    private Result traverse(TreeNode node) {
        if (node == null) return null;

        Result left = traverse(node.left);
        Result right = traverse(node.right);

        Result curr = new Result();
        curr.node = node;
        curr.inc = 1;
        curr.des = 1;

        if (left != null) {
            if (node.val - left.node.val == 1) {
                curr.inc = Math.max(curr.inc, left.inc + 1);
            }
            else if (node.val - left.node.val == -1) {
                curr.des = Math.max(curr.des, left.des + 1);
            }
        }

        if (right != null) {
            if (node.val - right.node.val == 1) {
                curr.inc = Math.max(curr.inc, right.inc + 1);
            }
            else if (node.val - right.node.val == -1) {
                curr.des = Math.max(curr.des, right.des + 1);
            }
        }

        max = Math.max(max, curr.inc + curr.des - 1);

        return curr;
    }
}
```

written by shawngao original link here

Solution 3

c++ solution:

```cpp
class Solution {
public:
    int longestConsecutive(TreeNode* root) {
        int longest = 0;
        dfs(root, root, longest);
        return longest;
    }

    pair<int, int> dfs(TreeNode * node, TreeNode * parent, int & longest) {
        if ( NULL == node ) {
            return make_pair(0, 0);
        }
        auto left = dfs(node->left, node, longest);
        auto right = dfs(node->right, node, longest);
        longest = max(longest, left.first + right.second + 1);
        longest = max(longest, left.second + right.first + 1);
        int inc = 0, dec = 0;
        if ( node->val == parent->val + 1 ) {
            inc = max(left.first, right.first) + 1;
        }
        if ( node->val == parent->val - 1 ) {
            dec = max(left.second, right.second) + 1;
        }
        return make_pair(inc, dec);
    }
};
```

python solution

```python
class Solution(object):
    def longestConsecutive(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        def dfs(node, parent):
            if not node:
                return 0, 0
            li, ld = dfs(node.left, node)
            ri, rd = dfs(node.right, node)
            l[0] = max(l[0], li + rd + 1, ld + ri + 1)
            if node.val == parent.val + 1:
                return max(li, ri) + 1, 0
            if node.val == parent.val - 1:
                return 0, max(ld, rd) + 1
            return 0, 0
        l = [0]
        dfs(root, root)
        return l[0]
```

written by zqfan original link here

From Leetcoder.