<https://leetcode.ca/all/549.html>

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].

**Attempt 1: 2022-12-29**

**Solution 1: Divide and Conquer (120min, similar as L124/P9.7.Binary Tree Maximum Path Sum, the difference is L549 has both candidate increasing and decreasing paths, we have to record both)**

**这道题的helper函数的思路是，从当前根节点出发，最长连续路径 = 左子树的最长递增路径 + 右子树的最长递减路径 - 1 或者 = 左子树的最长递减路径 + 右子树的最长递增路径 - 1，换言之，最终就是寻找最长递增路径 + 最长递减路径 -1 （不分左右）**

**Style 1: Create helper class Node {increase, decrease} to record current level recursion maximum count on path**

class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

// one.left = two;

// one.right = five;

// two.left = three;

// two.right = four;

// five.right = six;

// six.left = seven;

// seven.right = eight;

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

int result = s.longestConsecutive(two);

System.out.println(result);

}

private class Node {

public int increase;

public int decrease;

public Node(int increase, int decrease) {

this.increase = increase;

this.decrease = decrease;

}

}

int count = 0;

public int longestConsecutive(TreeNode root) {

helper(root);

return count;

}

private Node helper(TreeNode root) {

if(root == null) {

return new Node(0, 0);

}

Node left = helper(root.left);

Node right = helper(root.right);

// Initial both increase, decrease 'count' as 1 represent current node itself

Node cur = new Node(1, 1);

if(root.left != null) {

if(root.left.val == root.val - 1) {

cur.decrease = Math.max(cur.decrease, left.decrease + 1);

}

if(root.left.val == root.val + 1) {

cur.increase = Math.max(cur.increase, left.increase + 1);

}

}

if(root.right != null) {

if(root.right.val == root.val - 1) {

cur.decrease = Math.max(cur.decrease, right.decrease + 1);

}

if(root.right.val == root.val + 1) {

cur.increase = Math.max(cur.increase, right.increase + 1);

}

}

count = Math.max(count, cur.increase + cur.decrease - 1);

return cur;

}

}

Time Complexity: O(n), where n is the number of nodes in the tree.

Space Complexity: O(logn), on average for the recursion stack since this is a binary tree.

**Refer to**

<https://cheonhyangzhang.gitbooks.io/leetcode-solutions/content/solutions-501-550/549-binary-tree-longest-consecutive-sequence-ii.html>

public class Solution {

private class Node {

private int incr;

private int decr;

public Node() {

incr = 0;

decr = 0;

}

}

public int longestConsecutive(TreeNode root) {

int[] res = new int[1];

res[0] = 0;

process(root, res);

return res[0];

}

private Node process(TreeNode node, int[] res) {

if (node == null) {

return new Node();

}

Node left = process(node.left, res);

Node right = process(node.right, res);

Node curr = new Node();

int sum\_incr = 1;

int sum\_decr = 1;

if (node.left != null) {

if (node.left.val == node.val - 1) {

curr.decr = Math.max(curr.decr, left.decr + 1);

sum\_incr += left.decr + 1;

}

if (node.left.val == node.val + 1) {

curr.incr = Math.max(curr.incr, left.incr + 1);

sum\_decr += left.incr + 1;

}

}

if (node.right != null) {

if (node.right.val == node.val - 1) {

curr.decr = Math.max(curr.decr, right.decr + 1);

sum\_decr += right.decr + 1;

}

if (node.right.val == node.val + 1) {

curr.incr = Math.max(curr.incr, right.incr + 1);

sum\_incr += right.incr + 1;

}

}

res[0] = Math.max(res[0], Math.max(sum\_incr, sum\_decr));

return curr;

}

}

**Refer to**

**No need  'sum\_incr' or 'sum\_decr'**

<https://massivealgorithms.blogspot.com/2017/04/leetcode-549-binary-tree-longest.html>

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;

}

**Style 2: Return int[] array instead of create class Node {increase, decrease} to record current level recursion maximum count on path**

class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

// one.left = two;

// one.right = five;

// two.left = three;

// two.right = four;

// five.right = six;

// six.left = seven;

// seven.right = eight;

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

int result = s.longestConsecutive(two);

System.out.println(result);

}

int count = 0;

public int longestConsecutive(TreeNode root) {

helper(root);

return count;

}

private int[] helper(TreeNode root) {

// returns [longest\_decreasing\_length\_from\_root, longest\_increasing\_length\_from\_root]

if(root == null) {

return new int[]{0, 0};

}

int[] left = helper(root.left);

int[] right = helper(root.right);

// Initialize as 1 to represent current 'root' node

int increaseMax = 1;

int decreaseMax = 1;

if(root.left != null) {

if(root.left.val == root.val - 1) {

decreaseMax = Math.max(decreaseMax, left[0] + 1);

}

if(root.left.val == root.val + 1) {

increaseMax = Math.max(increaseMax, left[1] + 1);

}

}

if(root.right != null) {

if(root.right.val == root.val - 1) {

decreaseMax = Math.max(decreaseMax, right[0] + 1);

}

if(root.right.val == root.val + 1) {

increaseMax = Math.max(increaseMax, right[1] + 1);

}

}

count = Math.max(count, decreaseMax + increaseMax - 1);

return new int[] {decreaseMax, increaseMax};

}

}

Time Complexity: O(n), where n is the number of nodes in the tree.

Space Complexity: O(logn), on average for the recursion stack since this is a binary tree.

**Refer to**

[https://github.com/YaokaiYang-assaultmaster/LeetCode/blob/master/LeetcodeAlgorithmQuestions/549.%20Binary%20Tree%20Longest%20Consecutive%20Sequence%20II.md](https://github.com/YaokaiYang-assaultmaster/LeetCode/blob/master/LeetcodeAlgorithmQuestions/549. Binary Tree Longest Consecutive Sequence II.md)

**采用bottom-up的方法dfs (也就是Divide and Conquer). 每个点同时维护能向下延展的最大increasing 和 decreasing长度.**

Compared with [298. Binary Tree Longest Consecutive Sequence](https://github.com/YaokaiYang-assaultmaster/LeetCode/blob/master/LeetcodeAlgorithmQuestions/298. Binary Tree Longest Consecutive Sequence.md), this question includes more different conditions since it allows for:

* both increasing and decreasing order from a follows the parent-child path.
* child-parent-child path.

Hence this question actually contains 2 subproblems to solve:

* what is the longest increasing consecutive parent-child path sequence given a root node?
* what is the longest decreasing consecutive parent-child path sequence given a root node?

Based on the above 2 sub-solution, we know that the longest consecutive sequence for a given root is longest\_increasing\_sequence + longest\_decreasing\_sequence from this root. We can simply add up this 2 value because the longest increasing consecutive sequence and longest decreasing consecutive sequence is guaranteed to showed up in different child path (otherwise there will be a contradiction--a child's value cannot be greater than and less than the root's value at the same time).

If the root's value's value is not consecutive with a child's value, then the length of current sequence is simply 1.

Time complexity: O(n) where nis the number of nodes in the tree.

Space complexity: O(logn) on average for the recursion stack since this is a binary tree.

class Solution {

int max = 0;

public int longestConsecutive(TreeNode root) {

getLongestConsecutive(root);

return max;

}

private int[] getLongestConsecutive(TreeNode root) {

// returns [longest\_decreasing\_length\_from\_root, longest\_increasing\_length\_from\_root]

if (root == null) return new int[]{0, 0};

int[] left = getLongestConsecutive(root.left);

int[] right = getLongestConsecutive(root.right);

int dcr = 1, icr = 1;

if (root.left != null) {

if (root.left.val == root.val + 1) {

icr = left[1] + 1;

}

if (root.left.val == root.val - 1) {

dcr = left[0] + 1;

}

}

if (root.right != null) {

if (root.right.val == root.val + 1) {

icr = Math.max(icr, right[1] + 1);

} if (root.right.val == root.val - 1) {

dcr = Math.max(dcr, right[0] + 1);

}

}

max = Math.max(max, dcr + icr - 1);

return new int[]{dcr, icr};

}

}