Codility_

CodeCheck Report: trainingXHV6UB-83Y

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
TreeLongestZigZag Python	6 min	100%

Total score

Tasks Details

1. TreeLongestZigZag

Medium

Given a tree, find a downward path with the maximal number of direction changes.

Task Score	Correctness	Performance
100%	100%	100%

Task description

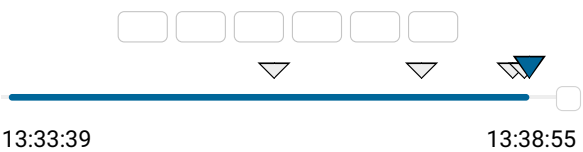
In this problem we consider binary trees. Let's define a *turn* on a path as a change in the direction of the path (i.e. a switch from right to left or vice versa). A *zigzag* is simply a sequence of turns (it can start with either right or left). The length of a zigzag is equal to the number of turns.

Consider binary tree below:

Solution

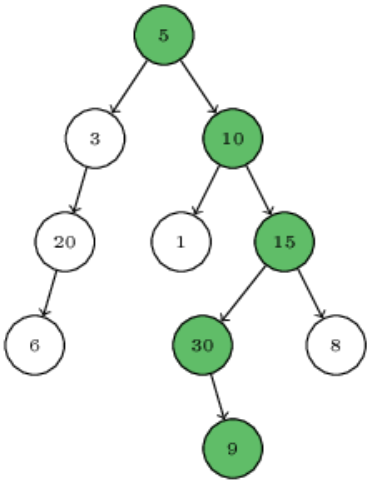
Programming language used:	Python
Total time used:	6 minutes
Effective time used:	6 minutes
Notes:	*not defined yet*

Task timeline



13:33:39 13:38:55

Code: 13:38:54 UTC, py, show code in pop-up



There are two turns on the marked path. The first one is at [15]; the second is at [30]. That means that the length of this zigzag is equal to 2. This is also the longest zigzag in the tree under consideration. In this problem you should find the longest zigzag that starts at the root of any given binary tree and form a downwards path.

Note that a zigzag containing only one edge or one node has length 0.

Problem

Write a function:

```
def solution(T)
```

that, given a non-empty binary tree T consisting of N nodes, returns the length of the longest zigzag starting at the root.

For example, given tree T shown in the figure above, the function should return 2, as explained above. Note that the values contained in the nodes are not relevant in this task.

Technical details

A binary tree can be specified using a pointer data structure. Assume that the following declarations are given:

```
from dataclasses import dataclass, field

@dataclass
class Tree:
    x: int = 0
    l: "Tree" = None
    r: "Tree" = None
```

An empty tree is represented by an empty pointer (denoted by None). A non-empty tree is represented by a pointer to an object representing its root. The attribute x holds the integer contained in the root, whereas attributes l and r hold the left and right subtrees of the binary tree, respectively.

For the purpose of entering your own test cases, you can denote a tree recursively in the following way. An empty binary tree is denoted by None. A non-empty tree is denoted as (X, L, R), where X is the value contained in the root and L and R denote the left and right subtrees, respectively. The tree from the above figure can be denoted as:

final, score: **100**

```
1  # you can write to stdout for debugging pur
2  # print("this is a debug message")
3
4  def solution(T):
5      # Implement your solution here
6      maxStep = [0]  # Store the maximum step
7
8      def dfs(root, isLeft, step):
9          if root is None:
10             return
11         maxStep[0] = max(maxStep[0], step)
12         if isLeft:
13             dfs(root.l, False, step + 1)  #
14             dfs(root.r, True, step)  # Rest
15         else:
16             dfs(root.r, True, step + 1)  #
17             dfs(root.l, False, step)  # Res
18
19     dfs(T, True, 0)
20     dfs(T, False, 0)
21
22     if maxStep[0] == 0:
23         return 0
24     return maxStep[0] - 1
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	**Example tests**	
▶	example example from problem statement	✔ OK
expand all	**Correctness tests**	
▶	corner_cases small test, N <= 10	✔ OK
▶	simple_balanced small balanced trees, N <= 20	✔ OK
▶	simple_skewed small skewed trees, N <= 30	✔ OK
▶	simple_random small, randomly generated trees, N <= 60	✔ OK
expand all	**Performance tests**	
▶	long_paths long paths with almost maximum tree height, N <= 8,000	✔ OK
▶	skewed big skewed trees, N <= 12,000	✔ OK
▶	random huge randomly generated tree, N <= 90,000	✔ OK
▶	extreme_max	✔ OK

```
  (5, (3, (20, (6, None, None), None), None),
(10, (1, None, None), (15, (30, None, (9,
None, None)), (8, None, None))))
```

maximum size tree in the shape of long path ending with huge binary tree, N <= 100,000

Assumptions

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..100,000];
- the height of tree T (number of edges on the longest path from root to leaf) is within the range [0..800].