Codility_

CodeCheck Report: training47KWB3-P7F

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
TreeHeight ⚠ Python	6 min	100%

Total score

100%

Tasks Details

Easy

1. TreeHeight
Compute the height of a binary tree.

Task Score	Correctness	Performance
100%	100%	Not assessed

Task description

In this problem we consider binary trees, represented by pointer data structures.

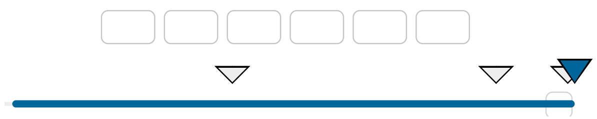
A *binary tree* is either an empty tree or a node (called the *root*) consisting of a single integer value and two further binary trees, called the *left subtree* and the *right subtree*.

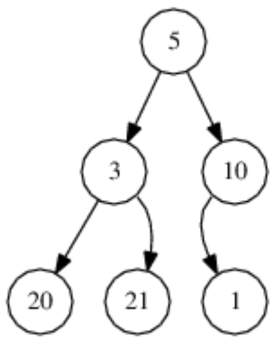
For example, the figure below shows a binary tree consisting of six nodes. Its root contains the value 5, and the roots of its left and right subtrees have the values 3 and 10, respectively. The right subtree of the node containing the value 10, as well as the left and right subtrees of the nodes containing the values 20, 21 and 1, are empty trees.

Solution

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

Task timeline





A *path* in a binary tree is a non-empty sequence of nodes that one can traverse by following the pointers. The *length* of a path is the number of pointers it traverses. More formally, a path of length K is a sequence of nodes P[0], P[1], ..., P[K], such that node P[I + 1] is the root of the left or right subtree of P[I], for 0 ≤ I < K. For example, the sequence of nodes with values 5, 3, 21 is a path of length 2 in the tree from the above figure. The sequence of nodes with values 10, 1 is a path of length 1. The sequence of nodes with values 21, 3, 20 is not a valid path.

The *height* of a binary tree is defined as the length of the longest possible path in the tree. In particular, a tree consisting of only one node has height 0 and, conventionally, an empty tree has height −1. For example, the tree shown in the above figure is of height 2.

Problem

Write a function:

```
def solution(T)
```

that, given a non-empty binary tree T consisting of N nodes, returns its height. 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 given 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

15:26:10 15:31:35

Code: 15:31:33 UTC, py, final, score: **100** show code in pop-up

```
# you can write to stdout for debuggin
# print("this is a debug message")

from extratypes import Tree  # library

def solution(T):
    # Implement your solution here
    if T is None:
        # Empty tree has height -1
        return -1

    left_height = solution(T.l)
    right_height = solution(T.r)

    return max(left_height, right_heig

```

Analysis summary

The solution obtained perfect score.

Analysis

expand all	**Example tests**
▶ example example, size=6	✔ OK
expand all	**Correctness tests**
▶ simple full binary tree, size=3	✔ OK
▶ simple_list left-biased linear tree, size=6	✔ OK
▶ just_root single node, size=1	✔ OK
▶ small_skewed almost linear, right-biased tree, size=10	✔ OK
▶ small_balanced balanced tree, size=10	✔ OK
▶ medium_skewed almost linear, right-biased tree, size=500	✔ OK
▶ medium_balanced balanced tree, size=500	✔ OK
▶ max_skewed almost linear, right-biased tree, size=1K	✔ OK
▶ max_balanced balanced tree, size=1K	✔ OK

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:

```
  (5, (3, (20, None, None), (21, None,
None)), (10, (1, None, None), None))
```

Assumptions

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

- N is an integer within the range [1..1,000];
- the height of tree T (number of edges on the longest path from root to leaf) is within the range [0..500].