Demo ticket

Session

ID: demoD2C9SG-2AE
Time limit: 120 min.

Status: closed

Created on: 2014-03-17 17:31 UTC
Started on: 2014-03-17 17:31 UTC
Finished on: 2014-03-17 17:35 UTC

Tasks in test

Task score

Test score

100%

100 out of 100 points

EASY

1. TreeHeight

Compute the height of a binary link-tree.

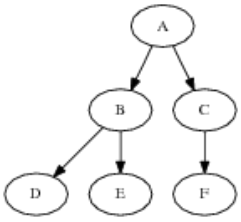
score: 100 of 100

Task description

In this problem we consider *binary trees*, represented by pointer data-structures. A pointer is called a *binary tree* if:

- it is an empty pointer (it is then called an *empty tree*); or
- it points to a structure (called a *node*) that contains a value and two pointers that are binary trees (called the *left subtree* and the *right subtree*).

A figure below shows a tree consisting of six nodes.
A *path* in a binary tree is a sequence of nodes one can traverse by following the pointers. More formally, a path is a sequence of nodes P[0], P[1], ..., P[K], such that node P[L] contains a pointer pointing to P[L + 1], for 0 ≤ L < K. K is called the *length* of such a 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 only of just one node has height 0 and the height of an empty tree is undefined.
For example, consider the following tree:



Subtrees of nodes D, E and F are empty trees. Sequence A, B, E is a path of length 2. Sequence C, F is a path of length 1. Sequence E, B, D is not a valid path. The height of this tree is 2.
Assume that the following declarations are given:

```
struct tree {
  int x;
  tree * l;
  tree * r;
};
```

Solution

Programming language used: C++

Total time used: 4 minutes

Effective time used: 4 minutes

Notes: correct functionality and scalability

Task timeline

17:31:53 17:35:10

Code: 17:35:10 UTC, cpp, final, score: **100.00**

```
01. // you can also use includes, for example:
02. #include <algorithm>
03. int solution(tree * T) {
04.     // we are done!
05.     if (T->l == NULL && T->r == NULL)
06.         return 0;
07.     // right-skewed tree
08.     else if (T->l == NULL)
09.         return 1 + solution(T->r);
10.     // left-skewed tree
11.     else if (T->r == NULL)
12.         return 1 + solution(T->l);
13.     else
14.         return 1 + max(solution(T->l), solution(T->r));
15. }
```

Analysis

```
};
```

Write a function:

```
int solution(tree * T);
```

that, given a non-empty binary tree T consisting of N nodes, returns its height.
For example, given tree T shown in the example above, the function should return 2.
Assume that:

- N is an integer within the range [1..1,000].

Complexity:

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(N).

Detected time complexity:
O(N)

test	time	result
example example, size=6	0.020 s.	**OK**
simple full binary tree, size=3	0.020 s.	**OK**
simple_list left-biased linear tree, size=6	0.020 s.	**OK**
just_root single node, size=1	0.020 s.	**OK**
small_skewed almost linear, right-biased tree, size=10	0.020 s.	**OK**
small_balanced balanced tree, size=10	0.020 s.	**OK**
medium_skewed almost linear, right-biased tree, size=500	0.020 s.	**OK**
medium_balanced balanced tree, size=500	0.020 s.	**OK**
max_skewed almost linear, right-biased tree, size=1K	0.020 s.	**OK**
max_balanced balanced tree, size=1K	0.020 s.	**OK**

Training center