2 Problem: Compute tree height

Problem Introduction

Trees are used to manipulate hierarchical data such as hierarchy of categories of a retailer or the directory structure on your computer. They are also used in data analysis and machine learning both for hierarchical clustering and building complex predictive models, including some of the best-performing in practice algorithms like Gradient Boosting over Decision Trees and Random Forests. In the later modules of this course, we will introduce balanced binary search trees (BST) — a special kind of trees that allows to very efficiently store, manipulate and retrieve data. Balanced BSTs are thus used in databases for efficient storage and actually in virtually any non-trivial programs, typically via built-in data structures of the programming language at hand.

In this problem, your goal is to get used to trees. You will need to read a description of a tree from the input, implement the tree data structure, store the tree and compute its height.

Problem Description

Task. You are given a description of a rooted tree. Your task is to compute and output its height. Recall that the height of a (rooted) tree is the maximum depth of a node, or the maximum distance from a leaf to the root. You are given an arbitrary tree, not necessarily a binary tree.

Input Format. The first line contains the number of nodes n. The second line contains n integer numbers from -1 to $n-1$ — parents of nodes. If the i-th one of them ($0 \le i \le n-1$) is -1, node i is the root, otherwise it's 0-based index of the parent of i-th node. It is guaranteed that there is exactly one root. It is guaranteed that the input represents a tree.

Constraints. $1 \le n \le 10^5$.

Output Format. Output the height of the tree.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 6 sec, `Python`: 3 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 3 sec, `Ruby`: 3 sec, `Scala`: 3 sec.

Memory Limit. 512MB.

Sample 1.

Input:

```
5
4 -1 4 1 1
```

Output:

```
3
```

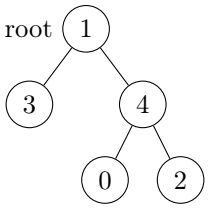
Explanation:

The input means that there are 5 nodes with numbers from 0 to 4, node 0 is a child of node 4, node 1 is the root, node 2 is a child of node 4, node 3 is a child of node 1 and node 4 is a child of node 1. To see this, let us write numbers of nodes from 0 to 4 in one line and the numbers given in the input in the second line underneath:

0 1 2 3 4

4 -1 4 1 1

Now we can see that the node number 1 is the root, because -1 corresponds to it in the second line. Also, we know that the nodes number 3 and number 4 are children of the root node 1. Also, we know that the nodes number 0 and number 2 are children of the node 4.



The height of this tree is 3, because the number of vertices on the path from root 1 to leaf 2 is 3.

Sample 2.

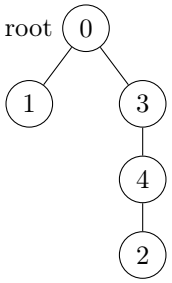
Input:

```
5
-1 0 4 0 3
```

Output:

```
4
```

Explanation:
The input means that there are 5 nodes with numbers from 0 to 4, node 0 is the root, node 1 is a child of node 0, node 2 is a child of node 4, node 3 is a child of node 0 and node 4 is a child of node 3. The height of this tree is 4, because the number of nodes on the path from root 0 to leaf 2 is 4.



Starter Files

The starter solutions in this problem read the description of a tree, store it in memory, compute the height in a naive way and write the output. You need to implement faster height computation. Starter solutions are available for C++, Java and Python3, and if you use other languages, you need to implement a solution from scratch.

What to Do

To solve this problem, change the height function described in the lectures with an implementation which will work for an arbitrary tree. Note that the tree can be very deep in this problem, so you should be careful to avoid stack overflow problems if you're using recursion, and definitely test your solution on a tree with the maximum possible height.

Suggestion: Take advantage of the fact that the labels for each tree node are integers in the range $0..n-1$: you can store each node in an array whose index is the label of the node. By storing the nodes in an array, you have $O(1)$ access to any node given its label.

Create an array of n nodes:

```
allocate nodes[n]
for i ← 0 to n − 1:
  nodes[i] =new Node
```

Then, read each parent index:

```
for child_index ← 0 to n − 1:
  read parent_index
  if parent_index == −1:
    root ← child_index
  else:
    nodes[parent_index].addChild(nodes[child_index])
```

Once you've built the tree, you'll then need to compute its height. If you don't use recursion, you needn't worry about stack overflow problems. Without recursion, you'll need some auxiliary data structure to keep track of the current state (in the breadth-first seach code in lecture, for example, we used a queue).

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.