CSE101 Binary Search Tree Questions: Left Pill or Right Pill?

- All code must be written in C/C++.
- Please be careful about using built-in libraries or data structures. The instructions will tell you what is acceptable, and what is not. If you have any doubts, please ask the instructors or TAs.

1 Setting

You are already provided with a Binary Search Tree (BST), that handles many basic operations. So far, there is code for "insert", "delete", various traversals, etc. So you already have the code to construct the tree and manipulate it.

The I/O format has been fixed. The input and output files are given as command line arguments. Each line of the input file is an operation on the BST. The various "print" operations print the BST in different orders, helping you reconstruct the tree.

For this test, you will have to implement one of the functions described below. *You cannot store the BST in some other data structure. You must operate on BST directly to perform these functions.*

1.1 The test questions

There are two questions that you need to study for the test. Assume that all keys in the BST are distinct.

- `Node* lca(int, int)`. Any two nodes in a BST (or any tree for that matter) have a *least common ancestor (LCA)*. This is the node furthest from the root that is an ancestor of both nodes. This function returns a pointer to the LCA of the nodes that contains keys corresponding to the arguments. Thus `lca(val1, val2)` returns (a pointer to) the LCA of the nodes containing val1 and val2 as keys. If either of these values is not present in the tree, return NULL.
- `int width()`. Define a *level* to be the set of nodes at the same distance from the root. The width is the size of the largest level. This function simply returns the width, as an int.

1.2 How it works

The header (that you will get in the test) will already have the right function declarations. Your job is to simply code up the function (that you will only get one as a test question). Note that if you design more functions to help with your code, you may need declare them in the header file.

The I/O will be taken care of, so there will be specific operations (in the input file) to perform these functions. So like "i <INT>" (for insert), etc. These will all be explained in a README file. You don't need to deal with any of this, but it might help you debug (if you run on your own inputs).

You will also get a small test input and test output. To make life easier, I will actually give you access to my grading scripts (that you can run directly through a shell script). If it catches a bug in your code, it will give you a test input where your code file. So you will get to see your score *before* you submit.

The test is completely closed book. You cannot bring any written material, but you can get blank scratch paper. You will get and can only open the specific Codio box for the test. You cannot refer to any other codes, or even open any other Codio box. If you do so, it will be considered cheating and you will get -10 points.

1.3 How should I go about it

Think recursion. Actually, no. First figure out the logic, before you think recursion. Start by explaining to yourself, in plain English, how to get the LCA and/or width of a tree. Then, try to break this up into recursive steps. You'll see the an iterative solution can be more complicated to design, while recursion is more natural. It is helpful to think about tree traversals, since the underlying logic on traversing a tree is central to many of these questions.

1.4 Other challenging questions

This is a collection of other questions, that might be good practice for exercising your skills. (And great interview practice.) For these, you don't need to write code, but you might want to figure out the logic. Trees and BSTs are a huge source of questions. For each of these, try to figure out the running time in $O(\cdot)$ notation.

- For any two nodes in a tree, there is a unique path between them. Give an algorithm to find it. (The LCA helps.)
- Print the keys of tree in "level order". So first print the root, then the nodes at level 1 (in order), then level 2, etc.
- Print all the leaves of a tree in sorted order. Do not use a sorting function.
- Given the preorder traversal of a BST, reconstruct the tree.
- Check if two BSTs are identical.
- Given a value `val`, delete all nodes in the tree less than `val`. Try to do this without visiting all the nodes less than `val`.