PA1b: AVL Trees

Overview

You will implement an AVL tree using your existing binary search tree code. You will then complete a reflection about the project.

Structure

This project is broken up into two parts:

1. Update the BST interface
2. Answer the questions for the PA1b reflection.

Update the bst Interface

You will use your existing code from PA1a and add new functions to support AVL rotations. From the outside perspective (i.e. from the perspective of the test cases) the public interface of the tree will not change. However, your tree will now be responsible for making appropriate rotations after inserting and deleting nodes to keep the tree balanced. We recommend making a copy of your original code and working on the copy, so you can start over from your original code if you need to. You can also remove the student_db code from the project, as we will not be retesting it.

bst Interface Details

You will modify your existing bst class to now represent a self-balancing AVL tree. Key and T refers to the key type and value type specialization of the templated deque. Since we won't call the rotations directly, you may implement them however you like. The following function signatures may help you get started:

Method	Description
`left_left_rotate(Node* &curr_node);`	Performs a left left rotation on curr_node.
`left_right_rotate(Node* &curr_node);`	Performs a left right rotation on curr_node.
`right_left_rotate(Node* &curr_node);`	Performs a right left rotation on curr_node.
`right_right_rotate(Node* &curr_node);`	Performs a right right rotation on curr_node.

File Details

To aid in completing the project, we have provided a new sample test file, called "avl_tests.cpp" on Canvas.

Testing

As always, we will use the Catch testing framework to test your implementation. Sample test cases are provided to show how your class will be used and to show how Catch tests are written so you can write your own. See the programming assignment 0 document for more details on structure of a catch test.

Compiling and Running Locally

The compilation and execution process is the same as PA1a, see the Programming Assignment 1a document for more details.

Compiling and Running Remotely

If you need a refresher on how to compile and run your project remotely, see the programming assignment 0 document.

Reflection

Write a reflection that answers the following questions:

1. What is the computational complexity of the methods?
2. What was the hardest part of this assignment?
3. What did you learn from this assignment?
4. What changes did you make from your initial design for part b and your implementation? Why? If you didn't make any changes, why not?

Submission

Submit the following files on canvas:

- Your updated bst.h file.
- Your reflection in docx or pdf format.

Note: **Do not** submit the student_db.h or student_db.cpp files for this part. We won't be retesting it.

When submitting, attach each file to your submission **separately**. **Do not** zip or tar the files.

Frequently Asked Questions

Why am I getting an error when I compile or run my code?

There are several things that could cause it. Before asking for help, try running `make clean` in your terminal to remove the build folder, then run `make` and `./build/test_pa1` again.

References

You may find this page helpful when implementing your bst::erase() function:
http://www.algolist.net/Data_structures/Binary_search_tree/Removal