COS2021 - Homework 3
Static vs Self-Balancing Trees

Due: Oct 17 Before midnight

Version: 00

Task Overview and Learning Goals

In this assignment, you will benchmark three different types of binary trees: Binary Search Tree, AVL Tree and Splay Tree. Code for the first two (BST and AVL) is given in a complete form. For the Splay Tree, you must fill in the splay() method with the proper rotations.

- Practice advanced C++ concepts, including inheritance, templates and classes.
- Implement a self-balancing tree algorithm.

Submission and Requirements

Submit your solution (all files) as zipped. I only want the code files (.hpp and .cpp) related to your project.

Your code should contain a header comment with your name and any issues or bugs I should know about. Remember: most programs contain some type of bug and it is a sign of a mature coder to acknowledge issues (not hide them).

Details

This assignment has two phases.

Phase 1

You must implement the *_splay* method in the Splay Tree class. This is a challenging method to implement at this level. Remember the splay tree uses two types of rotations (zig-zig and zig-zag) to shift a node to the root of the tree. Splaying happens whenever a node is inserted or found (using the find() method).

I strongly suggest you begin by drawing out a simple tree and practice splaying on paper. Write down the steps in pseudocode. And only then attempt to code a rotation.

Once you have the _splay method implemented, test it. Make sure it works before proceeding to phase 2.

Note: you are welcome to break the rotations up into separate functions.

Phase 2

With this homework, I am also providing full code for the Binary Search Tree and the AVL Tree. Phase 2 asks you to benchmark all three trees (BST, AVL and Splay) with respect to the average depth of nodes in the tree. You will do this in two ways.

1. You will add all values from 0 to 1000 inclusive to all three trees **in order**. Once all integers are in the trees, you will perform 100,000 random finds on each tree, recording the depth at which the integer is found within each tree. **Make sure that you search for the same integers in all three trees.** Calculate and output to the terminal the average depth returned by find for each tree across all 100,000 searches.
2. This task is exactly like 1, except instead of adding 0 to 1000 to all trees in order, you will add 0 to 1000 to the trees in random order. Make sure you add the random integers to the trees in the same order. Like task 1. perform 100,000 finds, calculate and print the average depth.

Miscellaneous and Hints

Please note that the BST and Splay Tree use a different node class from the AVL Tree. The BST and Splay Tree's node can be found in the *node.hpp* file. The AVL's node is at the top of the *avl.hpp* file. The AVLNode contains a height variable. The plain Node class does not have a height variable; however, it does keep a parent pointer. The parent pointer is not used by the BST, but the Splay Tree (and your implementation of the splay function) should make use of the parent pointer.

The Splay Tree's given methods are written such that for an insert, the new element is first inserted and then the splay is called at the point of insertion (the same happens for a find (minus the insertion of a new node)). This means that your splay function should work its way back up the tree via the parent pointer. I will admit, this is a somewhat inefficient way of doing things, but it is my opinion that implementing a splay tree like this once, gives you a better sense of the rotations.

Grading

It is not enough that the code solves the final problem. The solution must adhere to the requirements. It is possible for a submission to solve the program and receive a failing grade.

I will not try to crash your programs. This means, I will not give your programs malformed input. However, I do suggest you take every opportunity to develop code that fails gracefully.

If your program crashes, it will be hard to make above a 7 (or a C). Please test your code extensively.

- 4 pts for the splay implementation
- 4 pts for the two tests (2 pts each)
- 2 pts Miscellaneous (messy code, too much commented out code, poorly named variables, no header comment, very inefficient implementations, etc.).

The above breakdown does not mean you can skip part of the assignment. Submissions that do not *attempt* to solve all parts of the assignment will be counted as incomplete and receive no higher than a 5. What constitutes an attempt is up to my discretion.