CS2040C Data Structures & Algorithms Assignment #3: Binary Search Tree (BST)

Objective

In this assignment you will implement a binary search tree (BST) for yourself and add balancing operations. You will learn how to handle recursive tree operations and general pointer handling. This assignment expands on the previous assignment. You will need to implement many different functions in order to complete this assignment, it is suggested that you start with the simplest case and build up functionality. You will be provided a zip file that contains:

- tree.h, a header file with a skeleton definition of the template Tree class.
- tree_test.cpp with a number of tests.
- CMakeLists.txt for building and running the project.

Your task is to implement all of the methods marked as “TODO”, including:

1. **insert(T element)** to insert a new element into the tree.
2. **size() height()** and **empty()** to show the status of the tree.
3. **min()** and **max()** functions to return the minimum and maximum values in the tree.
4. **contains(T element)** returns true if the given element is in the tree.
5. **successor(T element)** finds the next element in the tree greater than the given value.
6. **in_order()** produces a string of the in-order traversal of the tree. See pre_order() for an example of how this functionality should work.
7. **post_order()** produces a string of the post-order traversal of the tree.See pre_order() for an example of how this functionality should work.
8. **Balancing** (not a separate function): your tree should always remain balanced. It’s suggested that you implement AVL rotations *after* the rest of the functionality is working.

Note that you may ignore duplicates.

A few general requirements:

1. Your code must compile and execute on Coursemology without crashing. **Code that fails to compile or crashes during testing may be given a zero mark**.
2. To that end, please **ensure you handle <u>all</u> corner cases in your code**. You are encouraged to use exceptions for any appropriate cases.
3. You **<u>must not use any libraries other than C++ primitives, such as string</u>**. This includes STL libraries, such as stringstream and containers.
4. **<u>Do not use signal handlers or attempt to bypass the autograding mechanisms</u>**, for example by using std::set_terminate. Any submissions found to be doing so will result in an automatic grade of zero (0) on the assignment and an investigation for academic integrity.

The unit tests given are a starting point. Passing them may not be sufficient to pass all of the private unit tests in Coursemology. As such **you are encouraged to add to or modify the unit tests given**. To get you started, there is a unit test that requires two rotations commented out in the unit test file. Once you have single rotations working, it is suggested that you uncomment that line and test again. You may also want to use simpler test cases for debugging certain conditions.

Submission

Submit to Coursemology the entire contents of tree.h. Coursemology will run a series of unit tests and provide you with feedback.

Extension Tasks

Here are some additional tasks that will further help you develop your C++ skills and prepare you for future assignments:

1. Implement a delete(T element) operation that removes a node from the tree. Add unit tests needed to ensure the operation is correct.
2. Add tree balancing operations after delete, including unit tests.
3. Can you perform repeated insertions and deletions?
4. Can you implement a generic method for applying a function to nodes in the tree in pre-, in-, and post-order?