PA1a: Binary Search Trees

Overview

You will implement a templated binary search tree (BST). You will then use your BST to create a program to track student information at a college. Lastly, you will consider what changes will be necessary to change your unbalanced BST implementation to a self-balancing Adelson-Velsky and Landis (AVL) Tree.

Structure

This project is broken up into two parts:

1. Complete the BST interface
2. Complete the student_db interface
3. Answer the questions for the PA1b design document.

Complete the Interfaces

Start by downloading the project zip folder from Canvas and extract it. You can use any text editor to open the project source files. As with PA0, if you are using an IDE like Visual Studio or CLion, take care that you are **only** using it to edit the source files, **not** to compile and run the code. You must do that on the terminal using the provided makefile (covered in the next section).

When completing the interface, you may add new public or private functions or member variables. However, you may not modify or remove existing functions or member variables. Additionally, you may not change the way the tests are written. E.g. You may not modify the tests so that a new function you create is called in the test case. Do not create a main function. The testing framework will generate it automatically.

You may use anything in the C++ STL to complete the project. We encourage you to explore the new features added in the C++11, C++14, and C++17 standards, such as smart pointers.

bst Interface Details

The bst class represents an unbalanced binary search tree. Key and T refers to the key type and value type specialization of the templated deque.

Method	**Description**
bst();	Constructs a bst object, initializing necessary member variables, etc.
~bst();	Destructs the bst object. Should free any dynamically allocated memory.
void insert(const Key &key, const T &t);	Inserts a new node into the bst, mapping key to t. If a node with Key key is already present in the bst, t replaces the previously mapped value.
void erase(const Key &key);	Removes the node with Key key from the bst. Throws an std::out_of_range error if a node with Key key is not found in the bst.
bool contains(const Key &key);	Returns true if a node with Key key is present in the bst and false otherwise.
T& at(const Key &key);	Returns a reference to the value type to which key maps. Throws an std::out_of_range error if a node with Key key is not found in the bst.

bool empty() const;	Returns true if the bst is empty and false otherwise.
size_t size() const;	Returns the number of nodes in the bst.
std::vector<std::pair<Key, T>> inorder_contents();	Performs an inorder traversal of the bst and returns the contents as a vector of pairs. Each pair's first element should be the key of the corresponding bst node and the second element should be the value of the corresponding bst node. See the test cases for details on the format expected.
std::vector<std::pair<Key, T>> preorder_contents();	Performs a preorder traversal of the bst and returns the contents as a vector of pairs. Each pair's first element should be the key of the corresponding bst node and the second element should be the value of the corresponding bst node.

student_db Interface Details

Method	**Description**
student_db();	Constructs a student_db object, initializing necessary member variables, etc.
~student_db();	Destructs the student_db object. Should free any dynamically allocated memory.
void insert(const int student_id, const std::string &student_name, const double gpa)	Inserts a new student with identification number student_id, name student_name, and grade point average gpa.
void erase(const int student_id)	Removes the student with identification number student_id. Throws an std::out_of_range error if a node with Key key is not found in the bst.
student_info& lookup(const int student_id)	Returns a student_info struct for the student with identification number student_id. Throws an std::out_of_range error if the student is not found.
bool empty() const	Returns true if there are no students in the database and false otherwise.
size_t size() const	Returns the number of students in the database.
std::vector<std::pair<int, student_info>> get_all_students()	Returns a vector of all students in the database in order of increasing identification number. Each element of the vector should be a pair where the first element is the identification number of the student and the second element is a student_info struct for the student. See the test cases for details on the format expected.

File Details

To aid in completing the project, we have provided a project skeleton on Canvas. Here is a short description of the files provided:

- bst.h - Contains the declaration and definition of the deque class and its member functions.
- student_db.h – Contains the declaration of the student_db class.
- student_db.cpp – Contains the definition of the member functions of the student_db class.
- catch.hpp – Catch framework header file. **Do not** modify this file.

- catch_main.cpp – Separate compilation unit for the catch framework to speed up the compilation process. **Do not** modify this file.
- bst_tests.cpp – Contains sample Catch unit tests for the bst class. You may add your own test cases to this file.
- student_db_tests.cpp – Contains sample Catch unit tests for the student_db class. You may add your own test cases to this file.
- makefile – Contains rules for compiling the project. **Do not** modify this file.

Note: There is no bst.cpp file, nor should you create one. Since the bst is a template class, the functions definitions must (unfortunately) go in the bst.h file.

Testing

As with PA0, 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

As with PA0, we have provided a makefile to compile your project. For more details on makefiles, see the programming assignment 0 document. To compile, open a terminal, `cd` to the directory containing the project files, and run `make` to compile the project. To run the project, run `./build/test_pa1` in your terminal.

The sample tests provided will ensure your implementation correctly conforms to the interface but are not meant to be exhaustive. You are encouraged to write your own test cases to ensure your project works properly under all edge cases.

For PA1 (and later programming assignments) you **do not** need to screenshot of the output of the above commands.

Compiling and Running Remotely

We will compile and test your project on the CISE servers. This will allow you to test your project on the exact same environment we will use for testing. Compiling and testing the project locally on your machine should give the same result as compiling and testing on the CISE servers, but there is always a small chance for some disparities. Thus, we recommend testing your project on the CISE servers so you can be absolutely sure your code will work when we test it.

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

Design Document

For part b of the programming assignment, you will replace your unbalanced binary search tree with a self-balancing AVL Tree. Write a design document that answers the following questions:

a) What methods will you need to add? Write pseudocode for these methods
b) How will you need to change your tree data structure to account for these new methods?

Submission

Submit the following files on canvas:

- Your completed bst.h file.
- Your completed student_db.h file.
- Your completed student_db.cpp file.
- Your design document in docx or pdf format.

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