CPSC 350: Data Structures and Algorithms
Spring 2024

Programming Assignment 5: LB-BSTs

Due: May 9th, 2024

The Assignment

In this assignment, you will push your C++ skills to the limit by implementing a simple database system using a special type of binary search tree. Though the final product will be a far cry from an Oracle or MySQL system, your DB will allow the user to insert, delete, and query data.

The DB itself will contain data that would be commonly found in a university's computer system. In our case, this information consists of student and faculty records. The information for each will be stored in its own tree (or "table" in DB terminology).

Though we will provide you with a general outline of the program, many of the implementation details will be up to you. In the same spirit, we will give you a point in the right direction as far as some of the C++ techniques go, but it will also be your responsibility to research the techniques in more detail if you have not utilized them before (e.g., operator overloading).

Lazily Balanced Binary Search Trees (LB-BSTs)

For our database table implementation, we will use a version of BSTs called Lazily Balanced Binary Search Trees, which behave similarly to classic BSTs, except when things get too imbalanced, the tree is totally rebuilt. To accomplish this, each node in the tree keeps track of the depth of the left subtree and the depth of the right subtree anchored at that node. When the difference between the depths of the left and right subtrees at the root of the tree is more than a factor of 1.5, the tree is automatically rebuilt to be perfectly balanced based on the following algorithm:

1. Each of the N values in the LB-BST is placed in an array of size N in ascending order
2. The median value of the collection is chosen and inserted into a new LB-BST as the root/leaf
3. The median value of everything less than the root/leaf is chosen and inserted into the tree is inserted as the root's/leaf's left child.
4. The median value of everything greater than the root/leaf is chosen and inserted into the tree is inserted as the root's/leaf's right child.
5. Steps 3 and 4 are repeated for each leaf until all N values have been added to the new LB-BST

Your first job will be to build a LB-BST implementation supporting the usual operations (including delete) that we specified in our BST implementation from class. Your class must be named *LazyBST* and must have the same methods and signatures as our in-class BST implementation. This should not be difficult in and of itself, as you can start with the BST code posted on

Canvas. Be sure to keep with the templated implementation we did and cover any missing error conditions.

You should test your LB-BST implementation thoroughly before moving on to other parts of the assignment.

The Database Tables

Your database system will consist of two tables, Student and Faculty, which contain the following types of records, respectively:

Student Records

Student records will be stored in a *Student* class. Student records contain a unique student ID (an integer), a string name field, a string level field (Freshman, Sophomore, etc.), a string major field, a double GPA field, and an integer advisor field, which will contain the Faculty ID of their advisor. These are the only fields the class contains.

The Student class must overload equality, less than, greater than operators, etc., so that we can compare them to one another.

Faculty Records

Faculty records are similar to student records and will be stored in a *Faculty* class. This class will also require overloaded operators.

Faculty records contain an integer Faculty ID, a string name, a string level (lecturer, assistant prof, associate prof, etc.), a string department, and a list of integers corresponding to all of the faculty member's advisees' ids. These are the only fields the class contains.

The Database System

Your program will keep references to both the faculty and student tables in memory. These references are simply *LazyBST* instances.

Once the tables have been created, a menu should be presented to the user to allow them to manipulate the database. The choices should include:

1. Print all students and their information (sorted by ascending id #)
2. Print all faculty and their information (sorted by ascending id #)
3. Find and display student information given the student id
4. Find and display faculty information given the faculty id
5. Add a new student
6. Delete a student given the id
7. Add a new faculty member
8. Delete a faculty member given the id.
9. Change a student's advisor given the student id and the new faculty id.
10. Remove an advisee from a faculty member given the ids

11. Exit

When a command is selected, you should prompt the user for the required data and execute the command. If there are any errors, you should inform the user by describing what the error is and aborting the command.

All of the above commands should enforce referential integrity. That is to say, a student can not have an advisor that is not in the faculty table. A faculty member can't have an advisee not in the student table. If a faculty member is deleted, then their advisees must have their advisors changed, etc. Your commands will be responsible for maintaining referential integrity. If a user issues a command that would break referential integrity, you should warn them and abort the command or execute the command and fix any violations as appropriate.

After each command is executed, the menu should be displayed again, and the user allowed to continue.

If the user selects the Exit option, prior to quitting your program should create a file named runLog.txt, which contains the information of both tables at the time the program was terminated. (This is the information from option 1 and option 2, just printed to the file instead of the terminal).

The Design

At this point, you should realize this is a non-trivial assignment. To successfully complete it, you need to make use of your best OO design and programming skills. Think modularly, sketch out a solution before you start coding, and START EARLY.

Rules of Engagement

- You may not use any data structures from the C++ STL or other third-party libraries. Of course, to do the file processing, you may use any of the standard C++ IO classes.
- For this assignment, you may work in groups of AT MOST 2 students.
- Develop using VSCode and make sure your code runs correctly with g++ using the course docker container.
- Feel free to use whatever textbooks or Internet sites you want to refresh your memory with C++ IO operations, just cite them in a README file turned in with your code. All code you write, of course, must be your own. In your README, please be sure to include the g++ command for compiling your code.

Due Date

The assignment is due on **5-9-24 at 11:59 pm**. Submit all your commented code as a zip file to Canvas. The name of the zip file should be LastName_FirstInitial_A5.zip

Grading

Grades will be based on correctness, adherence to the guidelines, and code quality (including the presence of meaningful comments). An elegant OO solution will receive much more credit than procedural spaghetti code. I assume you are familiar with the standard style guide for C++, which you should follow. (See the course page on Canvas for a C++ style guide and Coding Documentation Requirements.)

Code that does not follow the specification EXACTLY will receive an automatic 25% deduction. Code that does not compile will receive an automatic 50% deduction.