Project 4 – AVL Trees

Computer Science 2413 – Data Structures – Fall 2017

Due by 11:59 pm CST on Thursday, 7 December 2017

This project is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple system to read, efficiently merge, efficiently purge, sort, search, and write eclipse data with error checking and providing fast lookup by catalog number."

Objectives:

1	Use C++ file IO to read and write files, while using C++ standard I/O (`cin` and `cout`) for user interaction, using appropriate exception handling and giving appropriate error messages.	*5 points*
2	Encapsulate primitive arrays inside a templated class that provides controlled access to the array data, retains information on array capacity and use, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated array class.	*5 points*
3	Efficiently sort and search the data based on the field specified by the user. Integrate appropriate exception handling into classes that implement searching and sorting.	*5 points*
4	Encapsulate linked lists inside a templated class that provides controlled access to the linked-list data, retains information on linked list size, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated linked list class.	*5 points*
5	Encapsulate linked hash tables inside a templated class that provides controlled access to the linked hash table data, retains information on linked hash table capacity and load factor, and can be used to store data of any class or primitive type. Integrate appropriate exception handling into the templated linked hash table class.	*10 points*
6	Store eclipse data in an AVL tree sorted by catalog number.	*20 points*
7	Find eclipse data stored in the AVL tree based on catalog number.	*5 points*
8	Remove eclipse data from the AVL tree based on catalog number.	*15 points*
9	Provide an in-order traversal of the AVL tree.	*5 points*
10	Encapsulate AVL trees inside a templated class that provides controlled access to the AVL tree data, retains information on AVL tree size (number of entries), and can be used to store data of any class or primitive type.	*10 points*
11	Integrate appropriate exception handling into the templated AVL tree class.	*5 points*
12	Develop and use an appropriate design.	*5 points*
13	Use proper documentation and formatting.	*5 points*

Description:

For this project, you will revise and improve EclipseR 3.0 from Project 3 in one important way. You are encouraged to reuse and build on your code from Project 3. *EclipseR 4.0* will have the same basic functionality as EclipseR 3.0 but it will have one major change "under the hood"– because it was very time-inefficient to keep the list of eclipse data in a linked list that was always sorted by catalog number while data was read in, merged, and purged, EclipseR 4.0 will instead keep an AVL tree of eclipse data using catalog numbers as keys. This will allow for data for each eclipse to be inserted and removed in $\Theta(log_2\, n)$ time rather than $\Theta(n)$ time (where n is the number of eclipses in the data).

Note that EclipseR 4.0 will still copy the eclipse data to a linked hash map for fast lookup by catalog number and will still store the list of eclipse data to be sorted and searched using fields besides catalog number using a resizable array.

Operational Issues:

From a user interface perspective, EclipseR 4.0 will behave as described for EclipseR 3.0, except that the merge and purge operations may take noticeably less time.

Implementation Issues:

In most areas, EclipseR 4.0 will be implemented just as was EclipseR 3.0. This includes how EclipseR reads files and prints data, carries out user interaction via standard in and standard out, encapsulates C primitive arrays, implements exception handling for arrays and similar classes, and stores the lists of eclipses as linked lists. The big implementation change will be the data structure used to store the eclipses during merging and purging. For EclipseR 4.0, you are no longer allowed to store this data in a linked list; you need to use an AVL tree instead. Similarly, display/printing option C should now be accomplished by performing an in-order traversal of the AVL tree.

The only libraries you may use for this assignment are `iostream`, `iomanip`, `string`, and `fstream` (`#include <iostream>`, `#include <iomanip>`, `#include <string>`, `#include <fstream>`).

Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, accessibility modifiers, etc.

Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

Due Date:

You must submit an electronic copy of your Eclipse project to the appropriate dropbox in Canvas by **11:59 pm CST on Thursday, 7 December 2017**.

Important Note:

To ensure proper object-oriented design while making your libraries templated, it is a good idea to overload the comparison operators (e.g., `operator==`, `operator>`, `operator<`) for the `Eclipse` class.