Augmenting and Using an AVL Tree ADT

Out: 9/22
Due: 10/19 by 11:59 PM

"To iterate is human, to recurse divine."[L. Peter Deutsch]

Learning Objective

- ⊙ To Explore Properties of the AVL Tree ADT

On average, a binary search tree with n keys generated from a random series of insertions has expected height $O(\lg n)$. However, a worst-case performance of $O(n)$ is possible. This occurs, for example, when the keys are inserted in sorted order and the tree degenerates into a list. The self-balancing property of an AVL tree generally ensures that its height is shorter than a simple binary search tree for the same series of insertions. The AVL tree guarantees a worst-case performance of $O(\lg n)$ irrespective or the order in which the keys are inserted.

In this project you will augment the implementation of a parametric extensible AVL tree ADT (abstract data type). You will then write a program that instantiates an AVL tree to store strings. Your program will execute a series of statements from the grammar shown in Listing 1. The program will be called *Dendrologist*. It will take the *order-code* and name of a *command-file* as a command line arguments and execute the instructions in the file while performing a trace of the instructions as they are executed. The command file consists of instructions in one of these formats:

Listing 1: A Simple AVL ADT Grammar

```
insert <word>
delete <word>
traverse
paths
stats
```

The *insert* command in the file is followed by a word. When your program reads this instruction, it inserts the item into the AVL tree and displays a

trace message *Inserted: < item >*. Similarly, the *delete* command is followed by a word. When your program reads this instruction, it removes the item from the AVL tree and displays a trace message *Deleted: < item >*. When the *traverse* command is executed, it displays the heading *In-Order Traversal:* followed by the in-order traversal of the entries in the tree, displayed one per line. When the *paths* command is executed, it displays the heading *#Root-to-Leaf Paths: X*, where X is the number of root-to-leaf paths in the tree. The paths are displayed from the left-most to the right-most. Each path is represented by a string that denotes a sequence of vertices from the root to a leaf node. The $\rightarrow$ symbol is shown between every pair of vertices along the path. The paths are displayed one per line. When the *stats* instruction is read, your program displays the following trace message:
Stats: size = ..., height = ..., #full-nodes = ..., fibonacci? = ... The ellipsis are replaced with values obtained by invoking the relevant methods/functions of the AVL abstract data type. The *#full-nodes* label is followed by a non-negative integer and the *fibonacci?* label is followed by either *true* or *false*, depending on whether the AVL tree is a fibonacci tree. Prior to parsing the *command-file*, after the command line tokens are accessed, an AVL tree is instantiated using a comparator based on the *order-code*. The valid order codes are shown in Listing 2. To run the program:

Listing 2: Running Dendrologist

```
Dendrologist <order-code> <command-file>
   <order-code>:
   0        ordered by increasing string length, primary key, and
            reverse lexicographical order, secondary key
   -1       for reverse lexicographical order
   1        for lexicographical order
   -2       ordered by decreasing string length
   2        ordered by increasing string length
   -3       ordered by decreasing string length, primary key, and
            reverse lexicographical order, secondary key
   3        ordered by increasing string length, primary key, and
            lexicographical order, secondary key
```

Your program throws an exception and terminates if the correct number of command line arguments are not entered: *invalid_argument* for C++ and IllegalArgumentException, for Java™ programmers. It also displays the usage information shown in Listing 2, throws an exception and terminates if an invalid *order-code* flag is provided and gracefully exits. It also displays the usage information and throws an exception if the *command-file* does not exist. It throws an exception and displays "parsing error" and terminates if a command in the file is invalid.

I have provided starter code on Moodle that you will download and complete and a sample command file *strings.avl* containing various instruction on an AVL tree. See the starter code for additional details. Do not modify the code except where indicated. Be sure to test your program using various order codes. You may create additional command files to test other scenarios.

Submitting Your Work

1. Complete the preamble documentation in the starter code, if you augmented the file. Do not delete the GNU GPL v2 license agreement in the documentation, where applicable.

```
 /**
  * Describe what the purpose of this file
  * @author Programmer(s), YOUR NAME
  * @see list of files that this file references
  * <pre>
  * Course: CS3102.01
  * Programming Project #: 2
  * Instructor: Dr. Duncan
  * Date: LAST DATE MODIFIED
  * </pre>
*/
```

2. Verify that your code has no syntax error and that it is ISO C++11 or JDK 8 compliant. Be sure to provide documentation, where applicable, for the methods that you have been asked to write. When you augment the starter code, add your name after the @author tag and put the last date modified.

3. Enclose your source files-

   (a) for Java programmers, **AVLTree.java** and **Dendrologist.java**

   (b) for C++ programmers, **AVLTree.cpp** and **Dendrologist.cpp**

   - in a zip file. Name the zip file *YOURPAWSID_proj02.zip* and submit your project for grading using the digital drop box on Moodle. YOURPAWSID denotes the part of your LSU email address that is left of the @ sign.

Here is a partial sample trace output after the *Dendrologist* program is run with these command line arguments: *1 string.avl*.

Listing 3: A Partial Sample Trace: *Dendrologist 1 strings.avl*

```
Stats: size = 0, height = -1, #full-nodes = 0, fibonacci? = true
Inserted: twelve
Stats: size = 1, height = 0, #full-nodes = 0, fibonacci? = true
Inserted: nine
#Root-to-Leaf Paths:1
twelve->nine
Inserted: eleven
Inserted: ten
In-Order Traversal:
eleven
nine
ten
twelve
Stats: size = 4, height = 2, #full-nodes = 1, fibonacci? = false
Inserted: five
Inserted: four
Inserted: three
Inserted: eight
Inserted: one
Inserted: two
#Root-to-Leaf Paths:4
nine->five->eleven->eight
nine->five->four
nine->three->ten->one
nine->three->twelve->two
Inserted: six
Inserted: seven
Stats: size = 12, height = 4, #full-nodes = 4, fibonacci? = false
Deleted: two
Deleted: three
Deleted: five
Stats: size = 9, height = 3, #full-nodes = 3, fibonacci? = false
Deleted: seven
In-Order Traversal:
eight
eleven
four
nine
one
six
ten
twelve
Exception in thread "main" java.lang.IllegalArgumentException: File ↩
    Parsing Error.
```