Augmenting and Using an AVL Tree ADT

Out: 2/18
Due: 3/10 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. The difference in heights between an AVL tree and a general binary search tree, after the same series of insertions, becomes increasingly larger as the number of insertions increases. On the other hand, the difference in heights between an AVL tree and a binary search tree of minimal height with the same set of entries is relatively small.

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 or to store integers. Your program will execute a series of statements from the grammar shown in Listing 1. The program will be called *Arboretum*. It will take the name of a *command-file* as a command line argument 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> | <integer>
delete <word> | <integer>
traverse
stats
sort
```

The *insert* command in the file is followed by a word, in the case of a command file with string data, or an integer, in the case of a command file with integer data. 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 or an integer. 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 *stats* instruction is read, your program displays the following trace message: Stats: size = ..., actual-height = ..., optimal-height = ..., #leaves = ..., full? = ..., perfect? = ... The ellipsis are replaced with values obtained by invoking the relevant methods/functions of the AVL abstract data type. The label *optimal-height* is followed by the height of a binary tree with the same size but the shortest possible height. Recall that a complete tree has optimal height. The *full?* and *perfect?* labels are followed by either *true* or *false*, depending on whether the AVL tree is perfect or full. Wnen the *sort* instruction is read, the label *Sorted List of Tree Items:* is displayed on a line followed by a sorted list of entries in the AVL tree enclosed in a pair of brackets with each pair of entries separated by a comma and a white-space as shown in the Listing 3. To run the program:

Listing 2: Running Arboretum

```
Arboretum <data-type> <order-code> <command-file>

<data-type>: -s or -S for strings
   <order-code>:
   -1 for reverse lexicographical order
   1 for lexicographical order
   -2 for decreasing string length and reverse lexicographical order
   2 for increasing string length and lexicographical order
<data-type>: -i or -I for integers
   <order-code>:
   -1 for decreasing numerical order
   1 for increasing numerical order
<command-file>: name of the command file name
```

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 throws an ex-

ception and terminates if an invalid *data-type* or *order-code* flag is provided and gracefully exits if the *command-file* does not exist.

I have provided starter code on Moodle that you will download and complete. See the starter code for additional details. Do not modify the code except where indicated. I have also provided command files *strings.avl* and *integers.avl* that perform operations on an AVL tree of strings and integers, respectively. 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. Do not delete the GNU GPL v2 license agreement in the documentation, where applicable.

```
/**
 * Describe what the purpose of this file
 * @author Programmer(s)
 * @see list of files that this file references
 * <pre>
 * Course: CS3102.01
 * Programming Project #: 2
 * Instructor: Dr. Duncan
 * Date: 9999-99-99
 * </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, **AVLTreeAPI.java**, **AVLTree.java** and **Arboretum.java**

   (b) for C++ programmers, **AVLTree.h**, **AVLTree.cpp** and **Arboretum.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 *Arboretum* program is run with these command line arguments: *-s -1 string.avl*.

Listing 3: A Partial Sample Trace

```
Inserted: twelve
Inserted: nine
Inserted: eleven
Inserted: ten
In-Order Traversal:
twelve
eleven
nine
ten
Stats: size = 4, actual-height = 2, optimal-height = 2, #leaves = 2,↩
    full? = false, perfect? = false
Inserted: five
Inserted: four
Inserted: three
Stats: size = 7, actual-height = 2, optimal-height = 2, #leaves = 4,↩
    full? = true, perfect? = true
Sorted List of Tree Items:
[twelve, eleven, three, nine, four, five, ten]
Inserted: eight
Inserted: one
Inserted: two
In-Order Traversal:
twelve
eleven
three
eight
nine
four
five
two
ten
one
Inserted: six
Inserted: seven
Sorted List of Tree Items:
[twelve, eleven, three, seven, eight, nine, four, five, two, ten, ↩
   six, one]
Stats: size = 12, actual-height = 3, optimal-height = 3, #leaves = ↩
   6, full? = false, perfect? = false
Deleted: two
Deleted: three
Stats: size = 10, actual-height = 3, optimal-height = 3, #leaves = ↩
   4, full? = false, perfect? = false
```