Assignment 2: Extended AVL Trees

Overview

In this assignment, you will implement an ***enhanced AVL*** (***EAVL***) tree. The EAVL tree differs from an AVL tree in that it has member variables that store

- its current height,
- the number of nodes in the tree (called its size), and
- the internal path length of the tree.

It also provides a method for reporting the values of each of these metrics, as well as the number of nodes visited by find operations. Lastly, it has a method to provide the average number of nodes visited in all find operations so far.

Your main program will read a text file that contains commands, one per line, and will process those commands one after the other. The file includes commands to insert, remove, and find a specific word, to print the tree's contents in sorted order, and to report on the tree's statistical properties. The syntax and semantics of the commands are described in the detailed requirements below. Only the main program is permitted to read from an input file or write to an output file. The EAVL tree is permitted to write to an output stream that is passed to it, but not to any files.

Detailed Requirements

Input and Output

The program must get the name of the input file from its only command line argument. Specifically, the program must parse the command line and extract the input file name from the first command line argument. If there is no command line argument, it must report this as an error and exit. If the file name is supplied but cannot be opened for any reason, it must report this error and exit. The program is to put all output on the standard output stream, not in any file; to repeat this, it is not to place its output into a file.

Input File Syntax, Semantics, and Error Handling

The input file will consist of an unlimited sequence of lines, each of which starts with a command. The allowable commands are listed in the table below. The lines are free form, which means that any number of white space characters may precede or separate the tokens in the line. The lines are case-sensitive, i.e., "insert" and "Insert" are considered to be two different words. More importantly, the data is case-sensitive; the words "apple" and "Apple" are two different words and would have to be stored separately if they were each inserted.

The table below defines the semantics of each command. For each command, your program must take the action indicated in the right hand column. In this assignment, a *word* is any sequence of one or more non-blank, non-control characters, including letters, digits, and punctuation marks. Words may be up to 32 characters long. In the table, **boldface** indicates command keywords and *italics* represent placeholders for data.

Command	Description
`insert` *word*	If *word* is not already in the tree, create a new entry for it with frequency 1; otherwise increment its frequency in the tree and output the new frequency in the form: *word* <tab> *frequency*

Command	Description
`remove` *word*	If *word* is in the tree, decrement the frequency of *word*, and delete it if the frequency is zero. Then display a line of output in the form: *`word`* <tab> *`frequency`* If *word* is not in the tree, display *`word`* `<tab> not found`
`find` *word*	Search the tree for *word*. If it is found, display *word* and its frequency on a line of output. If it is not found, display *word* with a zero frequency. In either case, output the number of nodes visited in the search.
`display`	Display the contents of the tree in sorted order, including the frequencies of the items in the tree. Use the default collating sequence. This means that uppercase will precede lowercase. The libraries use this ordering by default in C and C++.
`report`	Produce a report for the tree consisting of: 1. the size of the tree 2. the height of the tree 3. the internal path length of the tree, and 4. the average number of nodes visited by the find command so far. The report should list each of these metrics, on a single line, in the above order, with a label that indicates what it is, such as `size = 120`, in the output stream.
`quit`	Clean up resources and terminate the program.

The main program should catch any line that is not one of the forms listed in the table above and display an error message on the ***standard error stream*** for each such line that it finds. It should continue to the next line of input after flagging the bad line.

Main Program and Project Structure

The main program should do all I/O and act like a client to the EAVL class. Specifically, it needs to repeatedly read the next command from input file, process the command, and write the appropriate output. It must also clean up all memory used before exiting.

The EAVL Tree Class Interface

Your EAVL Tree class must contain the following public methods. You may add others if you choose. You must implement the deletion algorithm by using replacement by the in-order successor, not the in-order predecessor. The methods are described by how they would be called by a client in the left column.

Public Method	Description
`int n = insert(`*`item`*`)`	If *item* is not in the tree, insert *item* into the tree, otherwise increment *item*'s frequency. In either case, return the frequency of *item* after the insertion has been performed.

Public Method	Description
`int n = remove(item)`	If *item* is not in the tree, return -1. If *item* is in the tree, decrement its frequency, and if the new frequency is zero, remove *item* from the tree. In either case return the frequency of *item* after the removal has been performed (so that a zero indicates the item is removed completely).
`int n = find(&item, &freq)`	Find the node containing *item* and, if it is found, update the frequency of *item* in the tree with the new frequency, *freq*. If *item* is not in the tree, set *freq* to 0. In either case, return the number of nodes visited.
`int n = height()`	Return the current height of tree.
`int n = int_pathlength()`	Return the current internal path length of the tree.
`int n = size()`	Return the total number of nodes in tree.
`float x = avge_nodevisits()`	Return the average number of nodes visited by all completed `find` operations on the tree so far. The average is, by definition, the total number of nodes visited divided by the total number of completed `find` operations. A `find` operation is complete if it returns a value of any kind.
`display(ostream)`	Write the items in the tree in sorted order, one per line, onto the given output stream.

Testing Your Program

You should design your own input files and test your program using your own input. You should carefully check that the output of your program is correct for the inputs you gave to it. Include files with bad lines, files with no lines, files that cannot be opened, files with all kinds of spacing, and so on.

Programming Constraints

- You are free to use either the code from my notes or the book, but you must cite this in the preamble if you do.
- You are not permitted to use any features of C++-11; the program must compile with the GNU C++ compiler in the lab, version 4.7.2 without the need to specify C++-11.
- For full credit your solution must maintain the size, height, and internal pathlength information as efficiently as possible.
- Your program must conform to the programming rules described in the Programming Rules document on the course website. It is to be your own work alone.

Grading

The program will be graded based on the following rubric.

- If the program does not compile on a cslab machine, it receives only 25%.
- For programs that compile:
  - Correctness and implementation 50%
  - Performance 10%
  - Design (modularity and organization) 20%
  - Documentation: 10%
  - Style and proper naming: 10%

Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on April 7, 2014. Create a directory named named *username*_hwk2. Put all project-related source-code files into that directory. ***Do not place any executable files or object files into this directory.*** You will lose 1% for each file that does not belong there, and you will lose 2% if you do not name the directory correctly. With all files in your directory, run the command

```
zip -r username_hwk2.zip ./username_hwk2
```

This will compress all of your files into the file named `username_hwk2.zip`.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the `cslab` machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to put your zip file into the directory

```
/data/biocs/b/student.accounts/cs335_sw/projects/project2
```

Give it permission 600 so that only you have access to it. To do this, `cd` to the above directory and run the command

```
chmod 600 username_hwk2.zip
```

where *username_hwk2.zip* is the name of your zip file.

If you put a file there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.