University of Calgary
Department of Electrical and Computer Engineering
Fall 2019
ENSF 593/594
Due Date: Friday December 6 before 11:59 PM

Assignment 11

Binary Search Trees

You are given an input ASCII text file containing an arbitrary number of student records in the following format:

Operation code:	1 character ('I' for insert, 'D' for delete)
Student number:	7 characters
Student last name:	25 characters
Home department:	4 characters
Program:	4 characters
Year:	1 character

Each record is stored as one line in the text file; i.e. there is a newline character immediately following the *year*.

Create a Java program that does the following:

1. Build a binary search tree using the data from the input file. Both insertion into and deletion from the tree will be done. The tree should be ordered by *student last name* (use a case-insensitive comparison). There are only unique records in the input file. Each node must contain the student data (exclude the *operation code*), a left child pointer, and a right child pointer. A parent pointer is optional but might prove useful for some operations.
2. Traverse the binary search tree recursively, printing out the nodes in ascending logical order; i.e. do a *depth-first*, *in-order* tree traversal. Print the node data to a text file.
3. Traverse the binary search tree, starting at the top level (the root node), proceeding downwards level-by-level. At each level print out the nodes from left to right. In other words, do a *breadth-first* traversal. You may have to use a queue to implement this. Print the node data to a text file.

Be sure to use proper headings and fixed-width columns for both of the output files. The program will be invoked from the command line as follows:

```
java Assign11 input output1 output2
```

where *Assign11* is the name of the file containing executable bytecode for your program, *input* is the name of the input text file, *output1* is the name of the text file containing node data from the depth-first, in-order tree traversal, and *output2* is the name of the text file containing node data from the breadth-first traversal. If the command line arguments are improperly specified, the program should print out a “usage” message and abort. Be sure that the specified files can be opened or created.

Create your own input files to test your code. Start with a file that specifies only insertions of data into the tree. Later, you should create a second file that also specifies some deletions as well. An official input file is available on D2L; use this file to create your output files.

Your program must be able to handle deletions from the binary search tree. For this, your program will need to examine the first field (the *operation code*) in each record of the input file. If it is the character 'D', then delete the node that matches from the tree. If there is no match, then report the error to standard output.

You must program all the data structures for this assignment from scratch; in other words, do not use any classes from the Java libraries to implement the binary search tree or queue. Also draw a picture of the binary search tree, as it appears after processing all the insertions and deletions specified in the input file. Within each node, only show the data that is used to order the tree. The picture can be hand drawn, or you may use a graphics application to create it.

Bonus (Optional)

Create a second version of your program, called *Assign11b*, which uses an AVL tree to store the input data. This version of the program should do the appropriate rotations when inserting data into the tree. Only insertions into the tree need to be handled (i.e. use an input file that specifies only insertions, and no deletions).

Complexity Analysis

Let *n* be the total number of records stored in the data structure.

1. Assuming that the records are inserted into the tree in random order, what is the height of your tree expressed using big-O notation?
2. What is the worst-case height of the tree? What input gives the worst case?
3. What is the worst-case space complexity of the depth-first, in-order traversal and breadth-first traversal? Compare your implementation of these two methods: is there one that will outperform another in terms of memory usage for a specific data set? Discuss.

Submit electronically via D2L:

1. A *readme* text file, indicating how to compile and run your program.
2. Your source code files. Your TA will run your program to verify that it works correctly
3. The two output files for the given input file.
4. Your picture of the binary search tree.
5. The answers to the Complexity Analysis questions.

Collaboration

The assignment must be done individually, so everything that you hand in must be your original work, except for the code adapted from the text, lectures, or other sources to implement your data structures. When someone else's code is used like this, you must acknowledge the source explicitly in your program. Copying another student's work, in whole, or in part, will be deemed academic misconduct. Contact your TA if you have problems getting your code to execute properly.

Assignment 11 Grading

Program			
Command-line arguments	2	______	
Reading from input file	2	______	
Tree data structure	4	______	
Insertion operation	4	______	
Deletion operation	4	______	
Depth-first traversal	2	______	
Breadth-first traversal	4	______	
Output files (well ordered & formatted)	4	______	
Code structure (documentation, formatting, etc.)	3	______	
Output files	2	______	
Picture of resulting tree	2	______	
Complexity Analysis			
Question 1	1	______	
Question 2	2	______	
Question 3	4	______	
Total	**40**	______	_____**%**