CS 130A (Winter 2018)

Second Project

Due: Friday, March 16, 2018 at 11:59 PM

This is the second part of your course project. This project can be done in a group of maximum two students. For this part of the project, you should be using Makefiles. One person per group should submit your code (as a zip archive) on Gauchospace before the due date. It should contain a README file with the full names (as it appears on the course roster) of both people in the group, separated by a comma.

As we make progress in the project, you will be reusing and improving this code a lot. We strongly recommend that you design it cleanly and comment it well. You will thank yourself in a few weeks. Please read all of the questions before you begin the implementing the project. The questions build up toward a coherent whole, so the way you solve one question may impact the solution of subsequent questions.

We strongly recommend that you create separate files for each of the classes you will write. Also, you should separate your header files from the code for each of them. Follow good practice in terms of coding style.

1. **Implement AVL Tree and 2-5 Tree classes.**

   Each node in the AVL or 2-5 trees is a pair of *(word,counter)* where counter shows the number of occurrence of the word in the dataset.

   Each class should have at least the following functions:

   (a) A constructor and a destructor.

   (b) A function for *searching* a word (the word may or may not exist).

   (c) A function for *inserting* a new word or incrementing the counter if the word is already inserted.

   (d) A function for *deleting* a word if the counter reaches zero or decrement the counter by one. Deletion of already deleted word should be ignored.

   (e) A function to *sort* all the words lexicographically.

   (f) A function for doing a *range search*. The function takes as input two words. Given two words, the function should find all the words in between. The resulting words need not be sorted.

   You are expected to find 3 pairs of such words which have 10, 100, and 1000 words in between them and measure the running time for performing range query.

   For this project, you should use UCI OpinRank Review dataset. Please use the "hotels-small" dataset from the last project. You are expected to parse each document in the dataset, ignore all the stopwords and non-alphabetical texts. The words should not be case-sensitive. Insert each word using the *insert* method you implemented for both Trees.

   Implementation details:

   Implement a simple text-based (cin, cout) user interface for your code. It should allow you to:

   (a) Search a given word, show whether the word exist in the dataset, and show the required time to search a word in AVL vs. 2-5 Tree

(b) Insert a new word and show the required time to insert a word into AVL vs. 2-5 Tree

(c) Delete a word and show the required time to delete a word from AVL vs. 2-5 Tree

(d) Sort all the words alphabetically and write them in a separate file and show the required time to sort all the words in AVL vs. 2-5 Tree

(e) Do a range search and show the required time to do range search in AVL vs. 2-5 Tree. The output of range search is inclusive if the words given exist in the trees.

Program Output Details:

Your code should compile into a single binary by running make. Call this binary "main". Once you run main, your code should build an AVL and a 2-5 tree out of the dataset. Then your code should prompt the user to enter 1, 2, 3, 4, or 5 to search, insert, delete, sort, or range search, respectively.

Then, for search, insert, and delete, you should prompt the user for an argument. For sort, you will prompt for an argument specifying a path to a file to output your results. For range search, you will prompt for two arguments, separated by a newline.

Your program should loop forever unless given a kill signal (CTRL-C); a user should be able to enter consecutive commands (insert, delete, insert, etc.) without restarting the program.

Your output should follow this structure EXACTLY to ensure full credit! However, the timings below are arbitrary examples.

```
> make

> ./main
> 1
> wordThatDoesExist
true
AVL: 0.10s
2-5: 0.001s

> ./main
> 1
> wordThatDoesNotExist
false
AVL: 0.2s
2-5: 1.0s

> ./main
> 2
> wordToInsert
AVL: 0.10s
2-5: 0.001s

> ./main
> 3
> wordToDelete
AVL: 0.10s
2-5: 0.001s

> ./main
> 4
```

```
/path/to/output.txt
AVL: 0.10s
2-5: 0.001s

> ./main
> 5
> startWord
> endWord
startWord
...
endWord
...
possiblyOtherWords
AVL: 0.10s
2-5: 0.001s
```

The output file for the sorted list should have each word separated by a newline, with the AVL tree and 2-5 tree output separated by a blank entry (2 newlines).

So for ['a','b','c'], you'd have:

```
a
b
c

a
b
c
```

Your code will be tested on CSIL. You can specify the compiler + options in your Makefile, but ensure any tools you use are installed system-wide on CSIL.

Do NOT submit the dataset with your code. You can assume the dataset exists in the root of your code repository, as a sibling to your Makefile.

e.g.

```
> ls code
Makefile hotels-small etc..


> ls code/hotels-small
beijing chicago dubai las-vegas etc..
```

2. **Results**

   A simple report should be submitted along with the code. The report should mainly consist the tabulated results shown in Figure 1 and to explain the findings in no more than two paragraphs. Time 100 operations of search, insert, and delete in order to deter variation from external factors. Lastly, please list the worst-case time complexities of each of the data structures.

Method	BST time	Big-O BST	Hash Table time	Big-O HT	AVL Tree time	Big-O AVL Tree	2-5 Tree time	Big-O 2-5 Tree
Search								
Insert								
Delete								
Sort								
Range Query (n=10)								
Range Query (n=100)								
Range Query (n=1000)								

Table 1: Comparison of time complexities of data structures