CS 130A (Winter 2018)

First Project

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

This is the first part of your course project. This project can be done in a group of maximum two students. For this part of the project, and all other subsequent ones, you should be using Makefiles. You should submit your code (as a zip archive) on Gauchospace before the due date. It should contain a README file with the instructions on how to compile and run your code.

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 Binary Search Tree and Hash Table classes.**

   Each node in the BST or row in the Hash Table 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 in the BST/Hash Table (the word may or may not exist).

   (c) A function for *inserting* a new word into the BST/Hash Table or increment the counter if the word is already inserted.

   (d) A function for *deleting* a word from the BST/Hash Table 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. The dataset has reviews of cars and hotels; for this project please use ONLY "Hotels review" data. 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 BST and Hash Tables.

   For Hash Table implementation, the size of the table should be $3/2n$, where n is the dataset size after removing the stopwords and non-alphabetical words. You have the freedom of choosing the hash function. On collisions, the function should store the data is the next available empty slot.

   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 BST vs. Hash Table

(b) Insert a new word and show the required time to insert a word into BST vs. Hash Table

(c) Delete a word and show the required time to delete a word from BST vs. Hash Table

(d) Sort all the words alphabetically and write them in a separate file and show the required time to sort all the words in BST vs. Hash Table

(e) Do a range search and show the required time to do range search in BST vs. Hash Table

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 a BST and a Hash Table 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 not prompt for an argument. 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
BST: 0.10s
Hash: 0.001s

> ./main
> 1
> wordThatDoesNotExist
false
BST: 0.2s
Hash: 1.0s

> ./main
> 2
> wordToInsert
BST: 0.10s
Hash: 0.001s

> ./main
> 3
> wordToDelete
BST: 0.10s
Hash: 0.001s
```

```
> ./main
> 4
/path/to/output.txt
BST: 0.10s
Hash: 0.001s

> ./main
> 5
> startWord
> endWord
startWord
...
endWord
...
possiblyOtherWords
BST: 0.10s
Hash: 0.001s
```

EDIT: The output file for the sorted list should have each word separated by a newline, with the BST and HashMap 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.

EDIT: We will test your code on a smaller dataset called 'hotels-small'. We will replicate this directory as 'hotels' to ensure backwards compatibility with previous instructions, in case this update goes unnoticed by some students.

e.g.

```
> ls code
Makefile hotels 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.

Method	BST	Worst case time complexity of BST	Hash Table	Worst case time complexity of HT
Seach	<time taken to perform 100 searches>		<time taken to perform 100 searches>	
Insert	<time taken to perform 100 inserts>		<time taken to perform 100 inserts>	
Delete	<time taken to perform 100 deletes>		<time taken to perform 100 deletes>	
Sort	<time taken to sort all the words>		<time taken to sort all the words>	
Range query (n=10)	<time taken to range query *n* words>		<time taken to range query *n* words>	
Range query (n=100)	<time taken to range query *n* words>		<time taken to range query *n* words>	
Range query (n=1000)	<time taken to range query *n* words>		<time taken to range query *n* words>	

Figure 1: BST vs. Hash Table