Lab #1 – shingles

David John

January 2020

1 Introduction

This lab is due Tuesday, February 4, at noon. We will shortly discuss how to use gitHub Classroom. Late penalties acrue at the rate of 10 points per day.

k-shingles are used in natural language processing to assist in making decisions about text authorship. It provides a frequency based statistic that often captures the writing style of an author. In this assignment we want to implement a Java program that creates the *4-shingles* for an input text. There are several goals related to this assignment:

- work with an old friend, the ArrayList,
- become re-established as a proficient Java programmer,
- acquire program information from the Java command line,
- efficiently implement the *extended* for to traverse an ArrayList, and
- review and continue to improve your design and debugging skills.

This is a **Pledged Work** assignment. You may only consult with me and the TA about the details of your work. You may not share the details of your design and implementation with others. The work you submit for grading must be your own work. You may not submit for grading work designed or implemented by others. Any violation will be referred to the Honor Council.

Here is the idea. Suppose your input file is:

```
abcdabcd abc
abcd abcdabcd
whataba tabat
```

For each 4-byte sequence (a *4-shingle*) in the file you want to determine its frequency:

```
 abc  3
 tab  1
 wha  1
a ta  1
aba   1
abat  1
abc   1
abcd  5
atab  1
ba t  1
bc a  1
bcd   3
bcda  2
c ab  1
cd a  2
cd w  1
cdab  2
d ab  2
d wh  1
dabc  2
hata  1
taba  2
what  1
```

Notice that a space can occur in a shingle.

Design Details

1. The data from the input text file must be normalized. You must remove all non-alphabetic information, spans of white space should be reduced to a single space. We will talk in class about *regular expressions* that will help.

2. The last characters on the current line of text must also be considered with the next line. This is an easy option to add once you have the basic "extracting strings of length 4" operation written.

3. You MUST create a class that contains a *4-shingle* and its frequency. Of course, you need other methods as well. We will discuss these in class.

4. Your "main" program must contain an ArrayList<Shingle> which will be an **ordered list** of Shingles. Anytime your search/traverse this ArrayList you MUST use the extended form of *for*. We will discuss this in class.

5. Your input must be read from a text file. The name of the text file must be a program command line parameter. We will discuss this in class.

6. Once your ArrayList is built from the input file, you must interactively query the user for 4-byte inputs. The response must be the frequency associated with that 4-byte input, or the message "Not in list".

7. At all times your ArrayList must be sorted, using insertion sort. You have not call upon any other sorting routine.

8. Searching your ordered list can be done sequentially (5 points) or using binary search search (10 points).

Your Java program must be well written and correct. The TA and I should be able to easily read your code. You must comment appropriately and use white space and indention effectively. **Points will be deducted for programs sloppily written.** You must sign every file at the top, this clearly indicates that this file is your work product. **Points will be deducted if you do not sign your work.**

How to start

There is no best way to start this assignment, but perhaps the easiest task to take on first is the design and implement the class for *4-shingle*. For data, we need each object to contain a string of length 4 and a frequency. What makes sense for a constructor? What getters/setters make sense? In your main class, create an object or two in this class. Test and make sure that everything works as you want.

Another fundamental task is the one that opens the input file, reads it line by line, and extracts all *4-shingles* out per line. There is an interesting bit moving from line to line, but maybe not initially. Finally the input file is closed. Test that this actually works.

Now create an ArrayList of *4-shingles*. As you find the *4-shingles* put them all into the ArrayList. We don't want this in the end, but might be a good task for testing and debugging.

Next, make your ArrayList ordered, do a smarter insertion into the list that does not insert duplicates but increases the frequencies. You might need to create some additional methods in your *4-shingle* class. Test to make sure everything is working. Test it again.

Next, modify your program to read the file name from the program command line.

Last, add in a section to interactively ask the user for a string of 4 characters and return either the corresponding frequency or an error message. (Most valuable for those of us grading your work.)