CMSC 206: Data Structures
Assignment 7: Maps
due on Gradescope by the beginning of class on Thursday, April 19, 2018

Part I.

In this part of the assignment, you will rewrite your Assignment 2 (the first zip-codes assignment) to use more efficient searching techniques. Specifically, we will take advantage of the fact that the zip codes in the *uszipcodes.csv* file are all in lexicographic order (which is the ordering used by `String`'s `compareTo` method). Since the array you read them into will naturally be sorted, you can use binary search.

1. Modify the `lookupZip` method (in *LookupZip.java* from Assignment 2) to use binary search. You can write the binary search either iteratively or recursively – your choice.

2. Save a screen shot testing your program looking up the following zip codes: 19010, 00501, 99950, 12345, 99705, 00500, 99951, 12346. Name the screen shot *binarysearch.png* (or whatever file format your system makes screen shots in).

Part II.

Write a Markov bigram babbler. A bigram is a sequence of two words. The babbler reads in a large text file (for example, the text of *Alice's Adventures in Wonderland*) – this file is called the *corpus* – and stores every bigram that appears. So, if our corpus were the first sentence of this paragraph, the stored bigrams would be (Write a), (a Markov), (Markov bigram), and (bigram babbler.). Note that capitalization and punctuation are preserved (your program will generally preserve these features without any special effort on your part). After reading in the file, your program will ask the user for the number of words to babble. You may wish to use the method in *ReadFile.java* (posted on our syllabus) to read in the file and break it up into words.

To babble, first have your program choose a bigram at random from the stored collection. Print out the two words in the bigram. Then, choose another bigram at random that begins with the second word of the first bigram. Print that bigram. Then, choose yet another bigram at random, chaining on the previous. Continue doing this until the program prints the requested number of words.

Though you are welcome to derive your own implementation, I will describe my structure. I used a `Babbler` class with one constructor that takes as a parameter the filename of the corpus. The constructor then reads in and processes the corpus, storing the bigrams in my class's sole field, a `Map<String, List<String>>`. A word maps to the list of possible words that may follow it. I then wrote a method `babble`, which takes the number of words to be printed. This method uses the `Map` to choose random bigrams to be printed. I then wrote a `main` method in another class to collect user input and to run the `Babbler`.

3. Write your babbler with your `main` method in *Babble.java*.

4. Make a screen shot (in *babble.png*) of running your babbler on the text of *Alice's Adventures in Wonderland*, available from our syllabus page.

For inspiration, here is a run of my babbler on *Alice's Adventures in Wonderland*.

```
Enter a source filename: alice30.txt
How many words should I babble? 100
now? That'll be treated with his mind, and was all these cakes,' she
said, `for bringing these words: `Yes, but he pleases!' CHORUS. (In
which way she was opened his head down, and the moral of great
curiosity. `What else for the door--I do that,' she could not, would
be NO mistake it makes people near the Mouse. `--I proceed. "Edwin and
say to know is, that cats eat one that had finished the verses to give
birthday presents to ear. Alice tried to listen to repeat it, or might
catch hold of the Dodo solemnly, rising to dry very
```