CS310: Advanced Data Structures and Algorithmns

Fall 2021 Programming Assignment 1

Due: Friday, Oct. 15 before midnight

Goals

This assignment aims to help you:

- Learn about hash tables
- Review packages
- Start thinking about performance and memory use

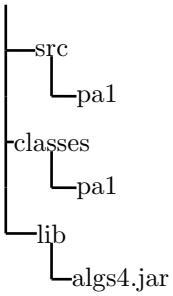
Preliminaries

Reading

- S&W Chapter 3.1 (Symbol tables) 3.4 (hashing)
- S&W Chapter 2.4 (Priority Queues)
- S&W code instructions http://algs4.cs.princeton.edu/code/ .
- Java documentation about packages: https://docs.oracle.com/javase/tutorial/java/package/packages.html.

Installation and compilation

- Create a directory named `pa1` (small letters) under your `cs310` directory (or wherever you test). Remember that Unix is case sensitive!



The source files you create for this assignment should reside in a package called `pa1`. When creating a package, all the source files should reside in a directory whose name is the same as the package name, **or your code won't compile**. The java source files should have a the following declaration at the top:

```
package pa1;
```

You have to include this statement at the top, **or your code won't compile**.

- Download the S&W code in a jar file. See here: http://algs4.cs.princeton.edu/code/ . At the bottom of the page there are installation and running instructions. Create a `lib` directory under your working directory, and copy the `algs4.jar` library there. Do not submit it into Gradescope! But you'll need it for your implementation.
- To compile the code you first change to the src directory or wherever your files are:

  ```
  cd src (the top-level source directory)
  javac -cp .:../lib/algs4.jar -d ../classes pa1/*.java
  ```

  The -d flag redirects the class files to the `../classes` subdirectory. The -cp (for classpath) flag tells the compiler to load the jar file during compilation. If you omit this part **your code won't compile** since your source code will not find the S& W classes it needs. Notice that on Windows you should put a ; (semi-colon) instead of a : (colon).

Note: It is **very** important that you follow these instructions to a t, and also understand what every command and every flag means. Even a small deviation will cause a Unix/compilation/runtime error that may cost you hours of extra work and will most likely cost both you and me some major headache.

Task

In this assignment you have to implement "The Markovian Candidate" assignment from S&W:
https://introcs.cs.princeton.edu/java/assignments/markovian-candidate.html.

Here are some tips and guidelines. Please do not write a single line of code before reading all of it!

1. Please read the instructions very carefully. It is a lot of text (but not THAT much programming). Make sure you understand exactly what is needed of you **before** you start programming. Also make sure you understand the math.
2. Read the checklist carefully. It will give you several guidelines to common problems and road blocks: https://introcs.cs.princeton.edu/java/assignments/checklist/markovian-candidate.html
3. Before you implement `BestModel`, implement and test `MarkovModel` separately. This will require you to write a `main` function for `MarkovModel` as well.
4. The class `MarkovModel` needs to have the function `public double laplace(String s)` as specified in the text. I will test this function, please implement it with this prototype. s is an input string. Implement getters – `getK()` (for the order of the model) and `getS()` (for the size of the alphabet).
5. See here for many test files: ftp://ftp.cs.princeton.edu/pub/cs226/markov. It may not work anymore, but there is a copy on the course webpage.
6. You will need a hash table (or possibly more than one) to store the substrings and their frequencies. You can either use algs4's `ST` or Java's `TreeMap` (you'll need a sorted data structure to produce the strings in alphabetical order as per the instructions).
7. The class `BestModel` needs to have a constructor with the following prototype: `public BestModel(int order, String s1, String s2)` (The two strings are the texts used to build the models. Obviously the constructor has to build two models).
8. The class `BestModel` also needs to have getters – `public MarkovModel getModel1()` and `public MarkovModel getModel2()`.
9. You will probably need a priority queue for the last part of the assignment where you print out the top 10 substrings. The priority should be the absolute difference in the log-likelihood under the two models. The way I found was easiest to implement it is to define a private inner class inside `BestModel`, named `DiffModel`. Your class should contain the substring, the two log-likelihoods and possibly the difference between them (you don't absolutely have to include the difference since it can be calculated at any time from the two log likelihoods, but it saves you a bit of time and makes your code nicer). Remember that the priority is the **absolute** difference between the two log-likelihoods. Design your `CompareTo` and `equals` functions accordingly, as seen in class.

10. For inner classes read here: https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html. Notice the difference between nested and inner classes. An inner class will work better.

11. The function `Math.abs` calculates the absolute value. The function `Math.log` calculates ln (natural logarithm). You will need to import `java.lang.Math`.

12. To read the files in you can either use a `Scanner` or algs4's `In`. The `In` class contains a method called `readAll()` which reads the entire file as a single string. Please replace all spaces by white spaces (see how to do it in the checklist above). Notice that it does not remove the newline in the end of the file. This means that your results may differ a little bit from the instructions. You may leave it or get rid of it. It will not affect your grade as long as your implementation is otherwise correct.

Running the Code

1. The command line options should be the same as in the instructions above, but the running is a bit different. Here is how it's done:

2. Go to the `classes` subdirectory under `pa1`.

3. To run `MarkovModel` use: `java -cp .:../lib/algs4.jar pa1.MarkovModel <num> <file>`, for example, to build a 2^{nd} order model for the text in a file `example.txt`: `java -cp .:../lib/algs4.jar pa1.MarkovModel 2 example.txt` . The `-cp` flag serves the same purpose as during compilation. If you don't add it **your code won't run**. (That is if you use the `algs4` classes). The `pa1` prefix indicates that the class `MarkovModel` belongs to the `pa1` package.

4. For the code to run as indicated above the file `example.txt` has to reside in your `classes` directory. **This is very important and a source of a lot of confusion**. If the file resides somewhere else you need to supply its path (absolute or relative). If you have no idea what that means, please please **learn about the Unix directory structure**. Also, no file named `example.txt` is provided as part of the assignment, it is just a generic name. You will have to download the files and experiment with them, or create your own.

5. Do NOT hard code the input file name or read it from the standard input. Use the command line exactly as is, or the grading script will fail.

6. Similarly, to run `BestModel` use: `java -cp .:../lib/algs4.jar pa1.BestModel <num> <model1> <model2> <test1> ...` (the three dots mean you can have as many test files as you want). For example, to build a 2^{nd} order models from the 2004 presidential debates and test them on two texts by George W. Bush, use `java -cp .:../lib/algs4.jar pa1.BestModel BestModel 2 bush1+2.txt kerry1+2.txt bush3-00.txt bush3-01.txt` , where the files are the ones residing in the ftp directory linked above.

7. As before, you should download the files and copy them over to your `classes` directory.

The memo.txt file

Create a `memo.txt` file to be submitted to Gradescope. As with pa0, it's a plain text file where you answer the following questions:

1. Estimate using order-of-growth notation (big-Oh) how much time and space your program requires to construct a Markov model from a data file of total size N characters, where k is the order parameter and S is the alphabet size. Regard k as a constant, and assume that symbol tables require logarithmic time per operation, and takes space proportional to the number of keys they contain. It doesn't have to be an exact number, just a big-Oh.

2. Estimate using order-of-growth notation how much time and space your program requires to compute the log likelihood of a new test string of length n under such a model. Disregard the time and space used to construct the Markov model.

3. Build a 2^{nd} order Markov model for the string `aabcabaacaac` (create a file containing the string and build the model). Copy and paste your output to `memo.txt`. It should resemble the output in the assignment description. Add the laplace formula for the three strings `aac, aaa, aab` as described in the assignment.

4. Do the same for a 1^{st} order model. Produce the laplace formula for the three strings `aa, ab, bc`.

5. Produce an average log-likelihood estimate of the John Kerry and Geroge W. Bush debate using a 2^{nd} order model and input as shown in the description of the assignment. The results should be similar (not necessarily identical) to the results in the description. Copy-paste your results to the memo.txt file.

6. Do the same for Obama and McCain. Use the command line:
`java -cp .:../lib/algs4.jar pa1.BestModel 2 obama1+2.txt mccain1+2.txt obama3-00.txt mccain3-02.txt`
.

7. The files will be provided as handouts.

Delivery

Before the due date, assemble files and try to compile and run them in a Linux environment. Remember again that UNIX is case sensitive, so the file names, class names and package name have to be as instructed. This is important since we use automated scripts. Make sure the sources compile and run on UNIX! They should, since Java is very portable. It's mainly a test that the file transfer worked OK and that you know how to compile and run from a command line.

- `memo.txt` (plain txt file, try "`more memo.txt`" on UNIX)
- `MarkovModel.java`
- `BestModel.java`