CMPS 12B: HW 5
Now that's a bard problem

- All assignments must be submitted through git. Please look at the Piazza guide on submitting assignments.
- Follow instructions, and carefully read through the input/output formats. Please run the checker!
- Clearly acknowledge sources, and mention if you discussed the problems with other students or groups. In all cases, the course policy on collaboration applies, and you should refrain from getting direct answers from anybody or any source. If in doubt, please ask the instructors or TAs.

1 Problem description

Main objective: We're going to do a word analysis of all the compositions of William Shakespeare. Get the full text of all compositions of William Shakespeare from the course website, on the HW page. The file is called `shakespeare.txt`. (It was obtained from Project Guterberg, so there are some copyright messages in it.) **It is vital that you download the file and save it. Do not copy and paste after clicking the link, since that does funky things with whitespace.**

We want to answer the following queries from the text.

- For any word length ℓ and number k, what is the k most frequent word of length ℓ?

Format and Output: You should provide a Makefile. On running `make`, it should create "Bard.jar". You should run the jar with *two* command line arguments: the first is an input file, the second is the output file.

Each line of the input file corresponds to a new query. The query is a pair of numbers: LENGTH RANK. The first number is the length of the word, the second number is the *rank*, which starts from 0. The ranking is done in decreasing order of frequency, and *increasing* lexicographic order. Thus, if two words have the same frequency, then the word that is earlier lexicographically has the smaller (earlier) rank.

So, "7 0" refers to the most frequent word of length 7. Or, "9 3" refers to the fourth most frequent word of length 9.

The output for each line is the corresponding word. If no word exists, the output is '-'. This can happen if the Bard decided not to use any words of that length, or there is no word of that rank. For example, if the input file is:

```
10 0
9 2
8 14
8 15
26 0
```

The output file is:

```
gloucester
messenger
business
personal
-
```

So “gloucester” is the most frequent word of 10 letters, etc. It turns out that both “business” and “personal” (both with 8 letters) have the same frequency of 230, but we rank “business” earlier than “personal”.

Instructions on parsing: It is very important that you follow these instructions. In general, parsing literary texts is a fairly messy task, since we have to make numerous decisions on how to split words. There is no one right way, but you need to follow my way so that your output matches mine.

The short answer on how to parse is to tokenize by whitespace *and* any of the following punctuation marks/symbols:

- Question mark (?)
- Comma (,)
- Period (.)
- Exclamation mark (!)
- Colon (:)
- Semicolon (;)
- Square brackets ([or])

Here’s one way of doing this. It is probably convenient to read the file line by line, so you will get a string (say) with a line. You first replace each of the above symbols, by a whitespace (like a tab). This can be done by the “replace” method for strings. (https://www.javatpoint.com/java-string-replace)

Then, you tokenize the resulting string by whitespace. You can use the “split” method. If the resulting string was called `str`, the command `tokens = str.trim().split("\\s+")` would produce a string array `tokens` obtained by splitting `str` by all whitespace. The `trim` method just removes any whitespace at the beginning or end, to prevent empty strings from appearing in `tokens`.

Each element (string) in `tokens` is a word. Furthermore, we will convert all words lowercase, using the method `toLowerCase()`.

Note that we are *not* splitting by hyphens, apostrophes, or quotation marks. One can debate the merits of this, but please follow this scheme for consistency.

Suggestions on coding: *You can use any data structures you want, including any built in Java type/class.* Hashtables are incredibly useful for this problem, so you will want to check out Java's hashtable class

(https://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html).

It is possible to pass through the file once, parse as described above, and produce a hashtable that stores all words with their frequencies. Now, you need to iterate over the hashtable to collect, for each length, all words of that length. Then, you can sort these words by decreasing order of frequency, and break ties among frequency using (increasing) lexicographic order. Comparators are a convenient way to do this, and there are numerous examples online for this. I liked the following: https://www.tutorialspoint.com/java/java_using_comparator.htm. Basically, if you define a class for words that stores frequencies, you can define a `compare` method that compares two objects of this class. Then, you can call Java's built in sort function on any arraylist of objects through `Collections.sort`. Check out the link above for doing this.

If you set this all up, then the actual task of producing the output is quite trivial. For each length, you already have words sorted by rank. So you do a single lookup for each input given.

Example commands and outputs: On the HW website, you will find a file `Examples.zip`. Download and unzip it. There are a number of input and corresponding output files.

- `test-input.txt, test-output.txt`. If you run your program with input file `test-input.txt`, the output file should be exactly the same as `test-output.txt`. This will be used by the checking script.
- `more-input.txt, more-output.txt`. This is a pretty comprehensive suite of tests. If you get this correct, you're reasonably guaranteed to be correct overall.

2 Grading

You code should terminate within 3 minutes for any input file with, say, at most 100 queries. If it doesn't, we will not give you credit. It's quite hard to give partial credit for this problem. You'll notice that once you get a few corner cases correct, your code will completely work.

1. (10 points) For a full solution as described above.
2. (3 points) Only answers word frequency queries.

3 Fun facts

Have some fun with this assignment. Can you find out the number of unique word that dear Bill used? What if you also tokenize by hyphens? What changes?

Some things that I learned.

- Not surprisingly, “the” is the most common three letter word.
- The longest word without hyphens is “honorificabilitudinitatibus”.
- The coolest word, IMHO, is “anthropophaginian”.
- “caeser” is more frequent that “brutus”, who both beat “othello”.
- Clearly, “tennis” was played in Elizabethan times.
- There are two “impossibilities” in all of Shakespeare’s work.