CIS36a – SPRING 2020

DUE May 5th

skills tested: file reading, use of HashMaps, use of random number generation, solving some tricky edge cases involving words at the end of a file.

Markov chain pt 1

Create a class called MarkovGenerator with the following private member variable:

HashMap<String, ArrayList<String>> markovEntry ;

In a markovEntry, the key represents a word, and the value is an ArrayList containing words that can follow that word.

Your MarkovGenerator class should contain a method for reading a text file.

public void readFile(String filename)

This method should open the file, and then do the following:

- Check if the current word is already a key in our HashMap.

    If not:

        Add the new key to the hash map

        Add the word *following* the current word to that key's associated ArrayList.

    If it *is* in foundWords, add it to the ArrayList already associated with that key

Note: it is perfectly reasonable to have multiple copies of the same word in the ArrayList associated with a given key.

When you reach the last word in the file, consider the *first* word as that word's successor.

For example, if your file consisted of the string `"one fish two fish red fish blue fish",` your foundWords array would hold five items:

Key	value
`one`	`[fish]`
`two`	`[fish]`
`red`	`[fish]`
`blue`	`[fish]`
`fish`	`[two, red, blue, one]`

Markov chain pt 2

Once you get the MarkovGenerator class from the previous question working correctly, it's time to actually use them to generate new text.

Write a new function for the MarkovGenerator class

String generateText(int numberOfWords)

that starts off by adding a random key from your markovEntries HashMap to the output String. Proceed by picking a random word from the ArrayList of potential successors to the current word. Add that word to the string, then find a random successor to that word. Repeat this process until you have generated the requested number of words.

Note that if a successor word often appears after a word, that successor word will be in the successors array multiple times. This means that your program is more likely to choose that word when generating the next word.

USE THE FILE emma.txt ON CANVAS AS YOUR INPUT FILE FOR TESTING.

Once you know it's working well, try it on a text file of your choice. Upload your favorite version of the results to canvas, with the filename markov.txt

Markov chain pt 2

For longer input files you can get better results by considering multiple words that appear side by side, then finding the word that's most likely to appear *after those words in order*.

Write a new version of MarkovGenerator with the following member variable:

HashMap<ArrayList<String>, ArrayList<String> >

The key should be two words that appear side-by-side, with the value being an ArrayList that contains all of the words that can potentially follow those two words side-by-side. For example, if your source text were:

```
It was the best of times, it was the worst of times, it was the
age of wisdom, it was the age of foolishness, it was the epoch
of belief, it was the epoch of incredulity,
```

Our sets of words and successors would start as follows:

```
[it, was] [the, the, the, the, the, the]

[was, the] [best, worst, age, age, epoch, epoch]

[the, best] [of]

[best, of] [times]
```

(and so forth…)

Use this new version to generate text, as in the previous version. I leave the implementation details for this one to you.

Test this on whatever file you'd like. Upload to canvas a file called betterMarkov.txt with your favorite autogenerated texts.