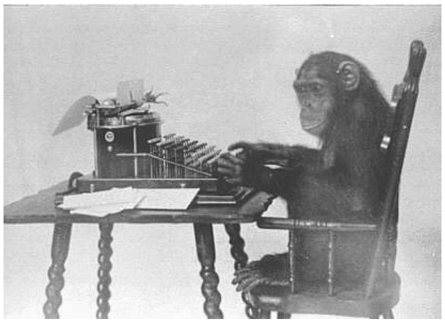
Lab Assignment 5
Random Thoughts of Infinite Monkeys

Lab Objectives



- Understand and apply collections appropriate to solving a problem.
- Think critically and analytically.
- See connections between similar problems and their solutions.
- Think about randomness and probability and their usefulness in problem solving.
- Think about monkeys.

Problem Description

The Infinite Monkey Theorem (IFT) says that if a monkey hits keys at random on a typewriter it will almost surely, given an infinite amount of time, produce a chosen text (like the Declaration of Independence, *Hamlet*, or a script for Star Trek the Next Generation). The probability of this actually happening is, of course, very small but the IFT claims that it is still possible. Some people intent on testing this hypothesis have set up virtual monkey sweatshops where poor cyber-monkeys type relentlessly in less than humane conditions. After billions and billions of simulated years, one monkey in the group was able to type out a sequence of 19 letters that can be found in Shakespeare's "The Two Gentlemen of Verona." (See the April 9, 2007 edition of *The New Yorker* if you're interested.)

The IFT might lead to some interesting philosophical discussions, but from a practical point of view – who cares? A more promising line of inquiry might go like this: Could a monkey with prior knowledge of Claude Shannon's information theory and Markov chains produce a reasonable imitation of a known text in a reasonable length of time? Now *that* might actually be interesting. It's so interesting, in fact, that we should give it a name – the Markov Monkey Question (MMQ) – and we should spend some time thinking about it.

It turns out that the answer to the MMQ is "maybe." Here's an example of a Star Trek TNG script produced by one of these "Markov monkeys" after reading representative samples of real TNG scripts.

```
Riker: Geordi, what is the status of the expected upgrade.
Geordi: Our scanners have picked up an increase in Borg storage and CPU capacity, but we still have no choice.
Requesting permission to begin emergency escape sequence 3F!
Geordi: [excited] Wait, Captain! Their CPU capacity has suddenly dropped to 0% !
Picard: Data, what do your scanners show?
Data: [studying displays] Appearently the Borg ship - with no life support suits! How can they survive the
tortures of deep space?!
Data: I believe you will look closer I believe that those are humans, sir. If you will look closer I believe
you will see that they are carrying something recognized by twenty-first century man as doeskin leather
briefcases, and wearing Armani suits.
Riker and Picard, together [horrified] Lawyers!!
Geordi: It can't be. All the Lawyers were rounded up and sent hurtling into the sun in 2017 during the Great
Awakening.
Data: True, but appearently some must have survived.
Riker: They have surrounded the Borg have found the answer by searching through our archives on late
Twentieth-century computing technology.
```

Besides a few spelling errors and some rather odd things that make you wonder about the author, this passage is surprisingly human-like. It turns out that Markov monkeys can apply a basic idea first discussed by Claude Shannon to produce pretty good imitations of real text.

So, here's the basic idea: Imagine taking a book (say, *Tom Sawyer*) and determining the probability with which each character occurs. You would probably find that spaces are the most common, that the character 'e' is fairly common, and that the character 'q' is rather uncommon. After completing this "level 0" analysis, you'd be able to produce random Tom Sawyer text based on character probabilities. It wouldn't have much in common with the real thing, but at least the characters would tend to occur in the proper proportion. In fact, here's an example of what you might produce:

Level 0
rla bsht eS ststofo hhfosdsdewno oe wee h .mr ae irii ela iad o r te u t mnyto onmalysnce, ifu en c fDwn oee iteo

Now imagine doing a slightly more sophisticated level 1 analysis by determining the probability with which each character follows every other character. You would probably discover that 'h' follows 't' more frequently than 'x' does, and you would probably discover that a space follows '.' more frequently that ',' does. You could now produce some randomly generated Tom Sawyer by picking a character to begin with and then always choosing the next character based on the previous one and the probabilities revealed by the analysis. Here's an example:

Level 1
"Shand tucthiney m?" le ollds mind Theybooure He, he s whit Pereg lenigabo Jodind alllld ashanthe ainofevids tre lin--p asto oun theanthadomoere

Now imagine doing a level k analysis by determining the probability with which each character follows every possible sequence of characters of length k. A level 5 analysis of Tom Sawyer for example, would reveal that 'r' follows "Sawye" more frequently than any other character. After a level k analysis, you'd be able to produce random Tom Sawyer by always choosing the next character based on the previous k characters (the *seed*) and the probabilities revealed by the analysis.

At only a moderate level of analysis (say, levels 5-7), the randomly generated text begins to take on many of the characteristics of the source text. It probably won't make complete sense, but you'll be able to tell that it was derived from *Tom Sawyer* as opposed to, say, *The Sound and the Fury*. Here are some more examples:

Level 2
"Yess been." for gothin, Tome oso; ing, in to weliss of an'te cle -- armit. Papper a comeasione, and smomenty, fropeck hinticer, sid, a was Tom, be suck tied. He sis tred a youck to themen

Level 4
en themself, Mr. Welshman, but him awoke, the balmy shore. I'll give him that he couple overy because in the slated snufflindeed structure's kind was rath. She said that the wound the door a fever eyes that WITH him.

Level 6
people had eaten, leaving. Come -- didn't stand it better judgment; His hands and bury it again, tramped herself! She'd never would be. He found her spite of anything the one was a prime feature sunset, and hit upon that of the forever.

Level 8
look-a-here -- I told you before, Joe. I've heard a pin drop. The stillness was complete, how- ever, this is awful crime, beyond the village was sufficient. He would be a good enough to get that night, Tom and Becky.

Level 10
you understanding that they don't come around in the cave should get the word "beauteous" was over-fondled, and that together" and decided that he might as we used to do -- it's nobby fun. I'll learn you."

(And by the way, this basic idea has very important applications in image processing and the automatic generation of realistic terrains and surfaces in computer graphics.)

What You Must Do

You are to implement a Java class named MarkovMonkey that provides the writing application described above for a given text source. Your class must have a public main method that takes the following four command line arguments:

- A non-negative integer *k*
- A non-negative integer *length*.
- The name of an input file *source* that contains more than *k* characters.
- The name of an output file *result*.

Your program must validate the command line arguments by making sure that *k* and *length* are non-negative, that *source* contains more than *k* characters and can be opened for reading, and that *result* can be opened for writing. If any of the command line arguments are invalid, your program must write an informative error message to System.out and terminate. If there are no command line arguments at all, your program must attempt to read a file named banana.txt in the current working directory. If present, this file will contain one or more lines, each of which contains the four arguments specified above and your program should process each line. If banana.txt is not present, your program must write an informative error message to System.out and terminate.

Given either valid command line arguments or the banana.txt file, your program must pick *k* consecutive characters at random from *source* and use them as the initial seed in the writing process outlined above. Your program must then write *length* characters to *result*. Each of these additional characters must be chosen based on the current seed. (Each time a character *c* is written to *result*, the seed is updated by removing its first character and appending *c* to the end.)

For example, suppose that *k* = 2 and the source file contains

the three pirates charted that course the other day

Here is how the first three characters might be chosen:

- A two-character seed is chosen at random to become the initial seed. Let's suppose that "th" is chosen.

- The first character must be chosen based on the probability that it follows the seed (currently "th") in the source. The source contains five occurrences of "th". Three times it is followed by 'e', once it is followed by 'r', and once it is followed by 'a'. Thus, the next character must be chosen so that there is a 3/5 chance that an 'e' will be chosen, a 1/5 chance that an 'r' will be chosen, and a 1/5 chance that an 'a' will be chosen. Let's suppose that we choose an 'e' this time.

- The next character must be chosen based on the probability that it follows the seed (currently "he") in the source. The source contains three occurrences of "he". Twice it is followed by a space and once it is followed by 'r'. Thus, the next character must be chosen so that there is a 2/3 chance that a space will be chosen and a 1/3 chance that an 'r' will be chosen. Let's suppose that we choose an 'r' this time.

- The next character must be chosen based on the probability that it follows the seed (currently "er") in the source. The source contains only one occurrence of "er", and it is followed by a space. Thus, the next character must be a space.

If your program ever gets into a situation in which there are no characters to choose from (which can happen if the only occurrence of the current seed is at the exact end of the source), your program must pick a new random seed and continue.

While there is no specific design or class architecture required, you must not submit a single-class solution. And in particular, the main method must not contain all the logic for your solution.

Project Gutenberg (http://www.gutenberg.org) maintains a huge library of public domain books that you can use as source texts. If your program generates something that is particularly good or funny, please send it to me so I can share it with the class. Be sure to identify the source text and the level of the analysis.

Lab Turn-In

You must turn in your submission(s) as specified in Canvas.