CS 441
Semester project – Spring 2020

For this project you'll be carrying out some simple text analysis, and comparing single- and multi-threaded performance. Though the original intention was to require this project to be done in Clojure, you may optionally do the project in Python *provided* you use a functional-programming style; in other words, this analysis should be done using Map/Reduce, rather than the usual imperative approach. *Python programs that do not use Map/Reduce will be heavily penalized.* (The point of the assignment is functional programming; solving this particular problem is mildly interesting but not really the point of all of it. Clojure programs will be functional by default, because of the language.)

The amount of *information* in a data stream is a measure of the uncertainty of the data, of how unexpected the next data item is, given what's come before. (That's why the news doesn't issue an alert when the sun rises in the East—if something is certain to happen, there's literally no information in it.)

More formally, the information in a character-based data stream S is given by $H(S)=\sum(n_c)(-p_c)lg(p_c)$ where p_c is the probability of each character (item) in a data stream, n_c is the number of times each character *c* occurs, and the summation is over all characters in the stream. If the logarithm is in base 2, then the information is measured in bits. (The reason this matters is that the information in a stream provides a hard limit to any compression if we want to recover the original data exactly. We cannot hope to compress it below this size without loss.)

The simplest measure is *first-order entropy*, in which we treat each character as independent of everything else. In this case, p_c is just the number of occurrences of *c* divided by the total number of characters. Thus we can just count characters and quickly find the total information.

But we know that in text, characters aren't really independent. In English text, if the last letter was *q* then the next letter will be *u*. If the last two letters were *Th*, the next letter is much more likely to be *e* than *t*, even though *e* and *t* are about equally likely when we examine single characters. We can capture this by examining groups of characters rather than single characters. In this case, our 'alphabet' consists of pairs of letters, rather than single letters. This gives us a larger alphabet but it also captures some of the dependencies in the language. Likewise, we can extend this idea to character triples just as easily. As each of these will account for more of the dependencies between characters, we should see the total information in the stream go down as we process larger groups.

Your program will be exploring this, and will also be exploring how concurrency can affect performance.

You are given a moderately large text file. Begin by counting the occurrences of each character (again, using Map/Reduce if you are working in Python), the probability distribution that results, and the total information in the file. Then find the same information using character pairs; then using triples. You should find that the total information in the file decreases under each of these measures.

Finding these quantities, should, we hope, be enough to keep your computer busy for a little bit. We'll now explore what happens as we add concurrency. Add code that measures the time needed for your program to compute these statistics (excluding file I/O). Run the program 3 times and note the average time needed. Now modify the program so it uses multiple threads. (In Clojure, use `pmap` rather than `map`; in Python, use a worker pool.) Run the program using 2, 4, 8, 16, 32, and 64 threads, running the

program 3 times under each condition, and finding the average time. (In Python, you may find it useful to also use a ProcessPool—the syntax to using a worker pool is almost identical—and see how the results compare.)

Prepare a short report including a graph showing execution time as a function of the number of threads, summarizing your findings and explaining any patterns you observe. Submit this report along with your source code.

Due Sunday night May 10.