Author: Ryan Timbrook
UW Net ID: timbrr
Project: Ling 473 Project 4
Date: September 1, 2016

Description:

This project was setup as a find substring problem in a 'Big Data' set. The specifics were to search for DNA sequences in the 24 chromosomes of the complete hg19 GRCh37 human genome. There were a total of 4,965 unique target sequences of variable length to find in 2.8 gigabytes of text data. The language for this search was composed of four characters which represent the human chromosome nucleotide bases; they are A, T, C, and G. The rules for matching characters included both upper and lowercase representations due to a "masking" convention used in DNA data to represent lower complexity. A fifth character, N, is also used in DNA data to represent Unknowns so this character represents a no match in the search routine.

Approach:

I chose to implement this project using the Python programming language version 3.4. For time efficiency purposes the solution was broken into two core components. The first component was to load the target sequences into a prefix "Trie" data structure. This data structure represents the most efficient way to search for multiple, arbitrary-length, substrings in a string. It enables the search to look for all targets at once yielding a O(n log m) complexity scaling. The second component was to then read in the DNA sequences in the 24 chromosomes which were separated into 24 .dna files stored on a network file drive.

Challenges:

Time complexity of the data set proved to be a major issue. Even with preloading the target sequence data into the prefix trie data structure before searching, this did not enable my solution to fully complete its search through all 24 .dna files in a single attempt on dryas. Many updates were made to the search function however they were unable to make any significant gains in time. Due to timing and not having full awareness into what was causing the major delays the search function only got through a subset of the files before I terminated the process so I could have the output data file for submission. As part of my search process I did take advantage of python's mmap module in efforts to speed up searching, but this did not have a significant effect on the overall success of the search time problem.