Report:

1. System configuration:

CPU: 2.9 GHz Intel Core i5

Clock rate: 2.9GHz

RAM: 8GB 1867 MHz DDR3

Cache size:

L2 Cache (per Core): 256 KB

L3 Cache: 3MB

2. Construction time:

chr12.fas: Insufficient disk space

colorblind\_human\_gene\_fasta: 12588 us

colorblind\_mouse\_gene\_fasta: 22636 us

Human-BRCA2.cds.fasta: 108767 us

Slyco.fas: 97941777 us

3. Justification:

The suffix tree construction run times were faster than I expected, considering the number of data points per file. I did not have tangible expectations but when the first colorblind file for humans ran in less than 13ms, I was surprised. After that file had run, it was no surprise that the mouse colorblind file and BRCA2 files would also run very quickly. The ~98 second runtime for Slyco then fell into expectations as well.

4. Implementation constant:

I do not yet know how to determine this.

5. BWT Index is included in the zip file.

When I run it for Human-BRCA2.cds.fasta, my results do not match the provided sample file.

I suspect that I have an error in the insertion functions.

Testing my code on the s2.fas file, the leaf node order does not match what I determine it should be by hand. There seems to be an issue with adding a suffix tree that goes beyond an internal node. I am still trying to fix this.

6. Exact matching repeat:

This can be determined by finding the lowest internal node that has at least one internal node as a parent. Reading its path label will give the string and subsequently, its length.

I do not yet have this implemented in code but I have an algorithm:

To determine the start coordinates:

\* Store the path label as a string

\* hash the string

\* store the length of the string as an integer, n

\* using strncpy function, copy n characters at a time from the beginning of the original string, into a temp string

\* use the same hash function on this temp string and compare their results,

\* if they match, perform a secondary confirmation by comparing our path label string with the temp string one character at a time (because no hash is perfect, could produce false positive)

\* if a perfect match is found, add the current index to an array (indexArray) and continue evaluating the original string by advancing the index one character at a time up to [(length of original string) – (length of path label)] (no point comparing a string that is shorter than the path label)

\* once you have iterated through the entire original string, iterate through indexArray and print all found indexes plus the length of path label