1. System Configuration

CPU Used: Intel Core i5-10600K

Clock Rate: 4.10 GHz

RAM: 32GB at 3200 MHz

Cache Size: 12.1 GB? (Not fully sure)

1. Construction Performance

With Naïve Tree Building

Slyco.fas and ch12.fas were too big to run with Naïve Tree building

**s1.fas**

4,745 microseconds

**s2.fas**

5,082 microseconds

**colorblind\_human\_gene.fasta**

64,346 microseconds

**colorblind\_mouse\_gene.fasta**

90,508 microseconds

**Human-BRCA2-cds.fasta**

1,275,705 microseconds

**Slyco.fas**

?

**chr12.fas**

?

**Covid\_Wuhan.fasta**

18,323,803 microseconds

**Covid\_USA-CA4.fasta**

17,593,789 microseconds

**Covid\_Australia.fasta**

17,172,338 microseconds

**Covid\_India.fasta**

17,313,510 microseconds

**Covid\_Brazil.fasta**

18,283,507 microseconds

With Suffix-Link Tree Building

I couldn’t get the suffix linking working correctly. I’m confident that my buildTree function (commented out below my current one) is implemented correctly and I feel the same way with my NodeHops function. The only issue is I kept getting a segmentation fault in NodeHops and couldn’t find out why.

I worked really hard on it though and spent many hours (around 20+ or so at least) on this project and would love if you’d be kind to give some partial credit for my efforts on the implementation of the suffix links. I understand the algorithm and how it should work, I just couldn’t get myself to implement it correctly. Thank you for understanding.

**s1.fas**

?

**s2.fas**

?

**colorblind\_human\_gene.fasta**

?

**colorblind\_mouse\_gene.fasta**

?

**Human-BRCA2-cds.fasta**

?

**Slyco.fas**

?

**chr12.fas**

?

**Covid\_Wuhan.fasta**

?

**Covid\_USA-CA4.fasta**

?

**Covid\_Australia.fasta**

?

**Covid\_India.fasta**

?

**Covid\_Brazil.fasta**

?

1. Justification

As far as the performance goes, I’d say that they met my expectations. The smaller s1.fas and s2.fas run really quickly since the string is so small, but the two largest files Slyco.fas and chr12.fas would take for ever to run when running the Naïve build tree method. It goes to show that the suffix linking is very important in tree construction to cut down time and cost, which is what we have learned about in class.

Additionally, As the size of the files went up, so did the time it took for the tree to be fully built and the program to be fully ran. This clearly makes sense as the files are larger since the strings are larger, meaning more nodes and computations to make.

And lastly as the time went up so did the size of space required. This, again, clearly makes sense because the nodes are increasing as the size of the string increases, meaning that we need more space to hold onto these nodes, thus making each tree have more bytes required the larger they got.

1. Implementation Constant

**s1.fas**

144 bytes

**s2.fas**

200 bytes

**colorblind\_human\_gene.fasta**

43,048 bytes

**colorblind\_mouse\_gene.fasta**

49,968 bytes

**Human-BRCA2-cds.fasta**

148,440 bytes

**Slyco.fas**

?

**chr12.fas**

?

**Covid\_Wuhan.fasta**

386,480 bytes

**Covid\_USA-CA4.fasta**

386,144 bytes

**Covid\_Australia.fasta**

386,368 bytes

**Covid\_India.fasta**

385,728 bytes

**Covid\_Brazil.fasta**

386,072 bytes

1. BWT Index

In my zip folder, I have a directory named “BWT\_Outputs”. In there are all the genome sequences, with a suffix of .txt.

For example, the BWT index for s1.fas is in “/BWT\_Outputs/s1.txt”

1. Exact Matching Repeat

To find the exact matching repeat, what I did was keep track of the deepest internal node through the creation of the tree. This was important because the path from the deepest internal node up to the root will be the longest exact matching repeat. This is because if there is an internal node, that means its edge is being used multiple times by other leaf nodes, so the deepest internal node would have the longest edge all the way up to the root, meaning the longest shared edge, implying the longest exact matching repeat.

The algorithm worked as follows, now that I have the deepest internal node saved. I made a function that essentially traverses back up the tree starting at that node and will be in a while loop up until the current node, we are on is equal to the root node.

While it’s not equal to the root node however, we append to some string the reverse of the parentEdgeLabel, doing this will give us the actual string in the correct order on the way back up. And each time we go through this loop, we must set our currentNode equal to currentNode->parent. Lastly, to get the size of this string, we just use the .length() function. And to get the starting coordinate, I represent it as the deepest internal nodes ID.

TLDR: Start Position = Deepest Internal Node’s ID

Length = Longest Exact Match Repeat Length

**s1.fas**

Start Position = 9

Length = 3

**s2.fas**

Start Position = 15

Length = 4

**colorblind\_human\_gene.fasta**

Start Position = 5374

Length = 470

**colorblind\_mouse\_gene.fasta**

Start Position = 6239

Length = 542

**Human-BRCA2-cds.fasta**

Start Position = 18546

Length = 2366

**Slyco.fas**

Start Position = ?

Length = ?

**chr12.fas**

Start Position = ?

Length = ?

**Covid\_Wuhan.fasta**

Start Position = 48268

Length = 5115

**Covid\_USA-CA4.fasta**

Start Position = 48247

Length = 5116

**Covid\_Australia.fasta**

Start Position = 48254

Length = 5113

**Covid\_India.fasta**

Start Position = 48206

Length = 5113

**Covid\_Brazil.fasta**

Start Position = 48244

Length = 5119