Git Repo: https://github.com/jhdavis8/cmsc858d

PART A

The most challenging part of implementing the buildsa program was figuring out how to serialize all of the relevant parts of the index to the disk in a binary format. I ended up relying on the cereal library to handle this, but I found their documentation to be surprisingly thin, at least for getting started. I also used the SDSL int vector's serialize function to write the suffix array itself to disk. I don't often write code to serialize objects, so I usually end up having to do some documentation hunting when it's needed and this being the most difficult part wasn't surprising to me. Overall I thought this part of the assignment was straightforward and I did not need to do much debugging at all. I used a library called FastaReader to read in the reference from disk and the SDSL qsufsort construct_sa function to build the suffix array.

Figures 1 and 2 below show the impact of the preftab k value on suffix array build time and index disk file size, for the provided e coli reference genome and three other genome files downloaded from the UC Santa Cruz Genome Browser[1] and listed in Table 1. Generally, both metrics increase as k increases, with a roughly exponential trend but sometimes with a plateau at higher k values. The metrics are also increase proportionally to the genome size. For example, for the E. coli genome, when no prefix table is created, the index file is about 40 MB, and it takes about 1.5 seconds to build. These both increase as k increases, with a roughly exponential trend, and the increase plateaus slightly after k=12. At the highest k tested, k=16, the build takes about 4.9 seconds and produces a 179 MB index file.

Assuming no prefix table is built, constructing the suffix array seems to take roughly 10x the number of bytes of the .fa genome file. So, assuming that trend holds, on a machine with 32 GB of RAM, the suffix array could be constructed for a 3.2 GB genome. If a prefix table is built with a higher k value, this relationship is more like 40x, so only a 0.8 GB genome can be processed.

Name	Description	.fa file size (MB)
SARS-CoV-2	Coronavirus	0.03
Escherichia coli	Bacteria	4.5
Saccharomyces cerevisiae	Yeast	11.8
Ciona intestinalis	Sea vase	112.1

Table 1: Genome files used

[1] https://hgdownload.soe.ucsc.edu/downloads.html

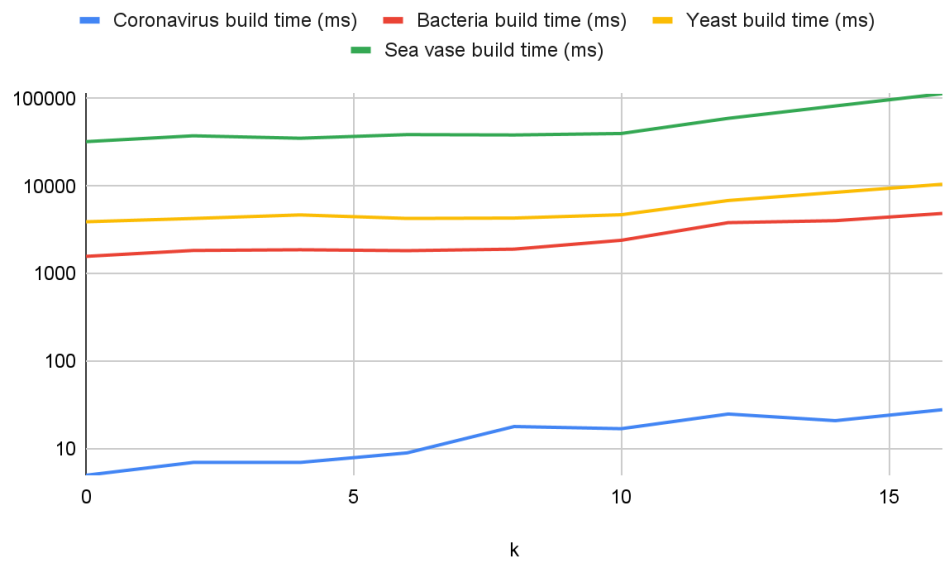

Fig. 1: Suffix array build time as preftab k increases

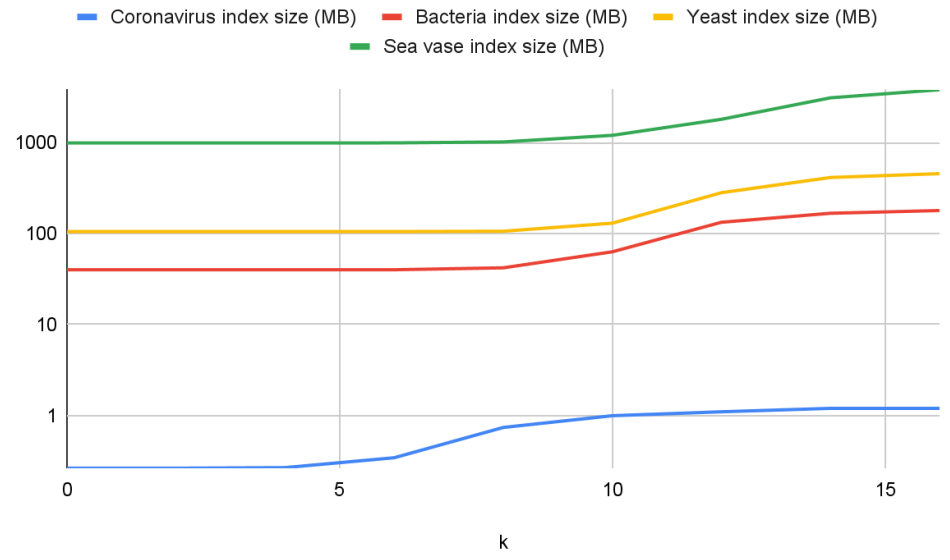

Fig. 2: Suffix array index size on disk as preftab k increases

PART B

The most challenging part of creating the querysa program was, for me, implementing the simpaccel optimization. I had trouble getting the optimization to actually show a performance improvement, and I ended up having to refer to the original textbook[2] where it's described in order to really understand it. I also had some trouble handling edge cases that come up in the search when the prefix table is used, such as when the initial low bound of the search matches the query. The provided E. coli test case was extremely helpful for debugging. Overall this part was more difficult than A, and took some time to debug, but it wasn't too bad.

I again use the genomes listed in Table 1 for my evaluation. Figs. 3 and 4 show the results of running the naive and simpaccel algorithms with the first 100 queries from the provided query set. The query time increases proportionally with the genome size. In comparing simpaccel vs. naive, the difference in time is mostly within measurement noise range. Occasionally simpaccel is slightly slower than naive. I believe this is due to the overhead of computing the LCP for the lower and upper search bounds on every search, compared to the benefit of fewer character comparisons which is mostly only gained for longer searches. Generally, the k value shows a "sweet spot" relationship with query time, with optimal values typically between 2 and 10. At higher ranges, I believe the slowdown comes from the poor performance of the C++ std::unordered_map I use for the prefix table, which may be performing poorly with a large number of keys.

Figs. 5 and 6 show performance of the naive and simpaccel algorithms with the yeast genome on a query set with small query lengths and larger query lengths, respectively. The sweet spot behavior is also reflected in these figures. Additionally, I see some slowdown of the simpaccel algorithm over the naive. I believe this is, again, due to the overhead of the LCP computation on each search, and the relatively small size of the reference genome which limits the benefit of the comparison skips. Overall, the size of the queries does not seem to have a significant impact on the query performance in my testing.

The k size trade-off of memory versus performance seems to be best around a k value of 10, as mentioned above. The memory requirements do not increase excessively at this point.

[2] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511574931

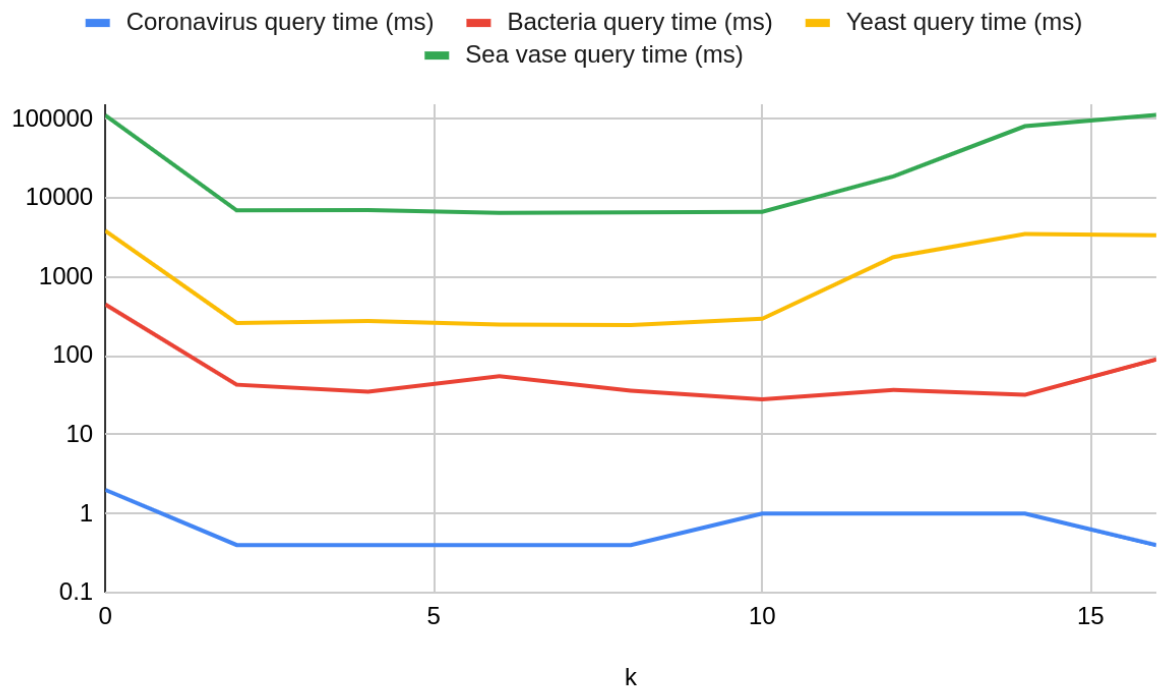

Fig. 3: Naive query time

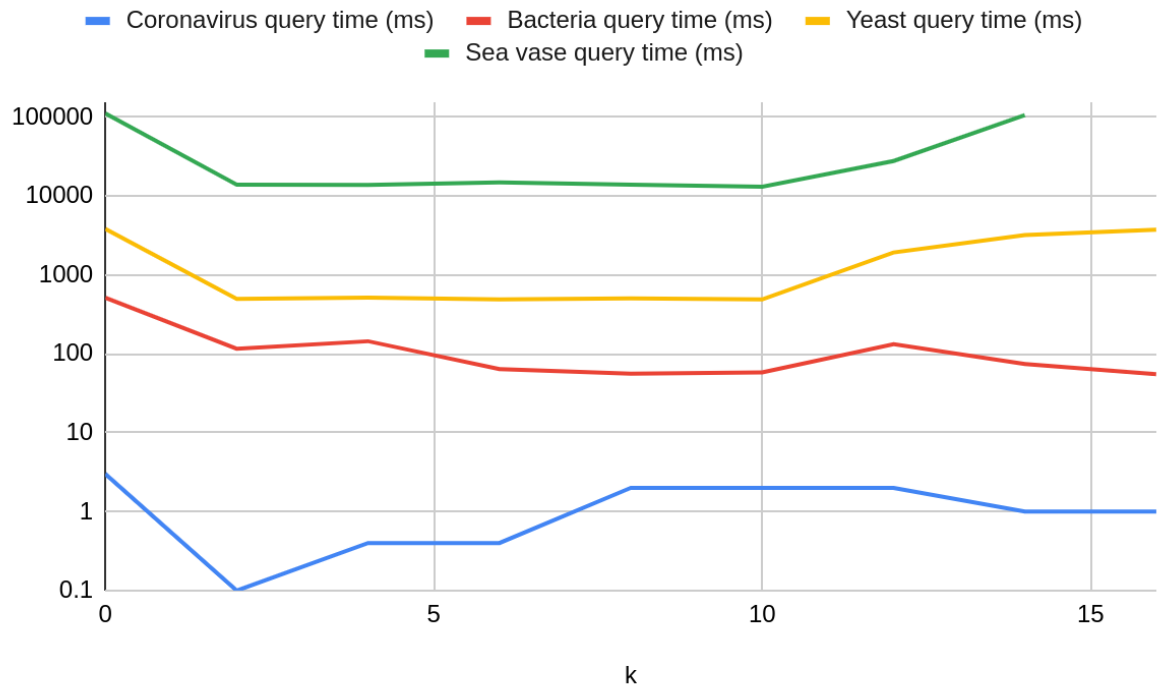

Fig. 4: Simpaccel query time

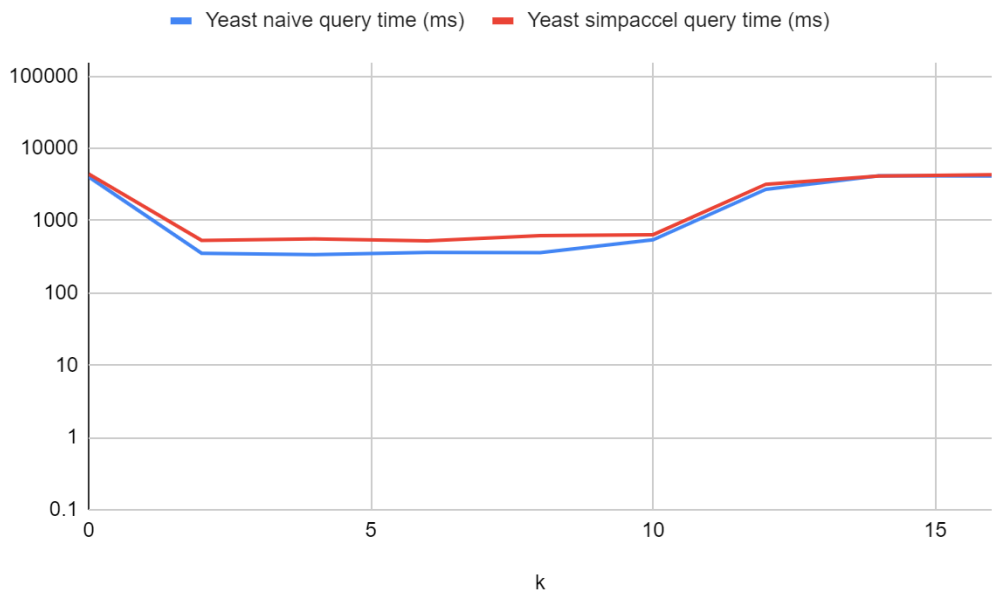

Fig. 5: Yeast query time for smaller query lengths

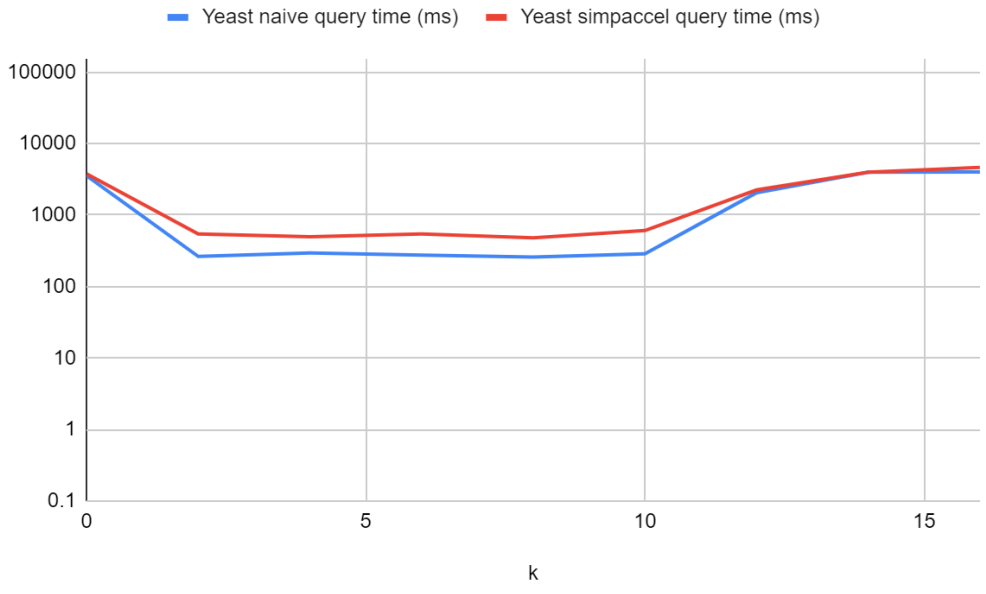

Fig. 6: Yeast query times for larger query sizes