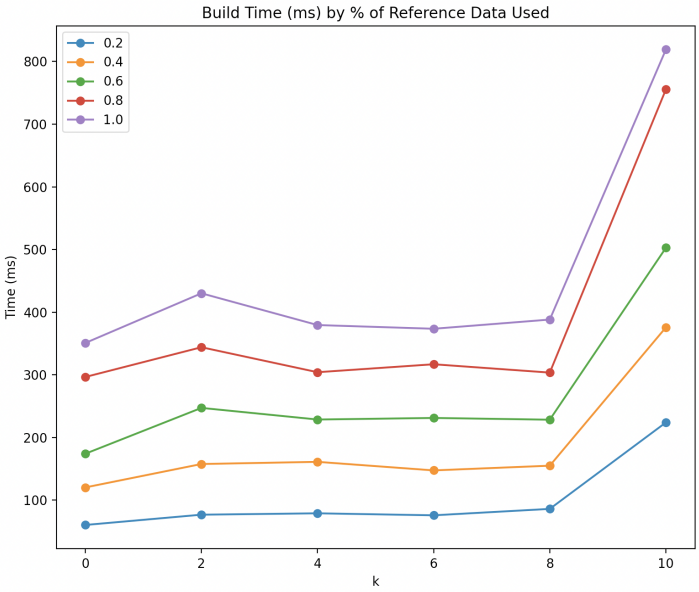
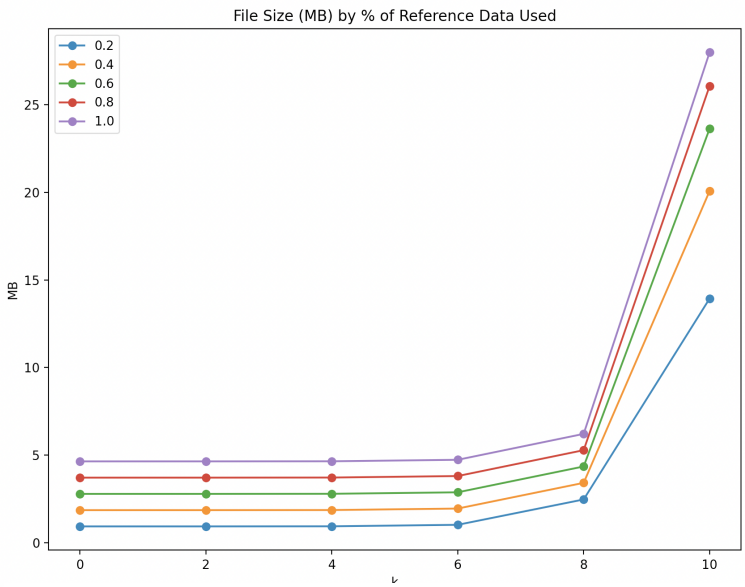
Anton Goretsky
CMSC858D Assignment 2 Writeup

Part 1 : Building



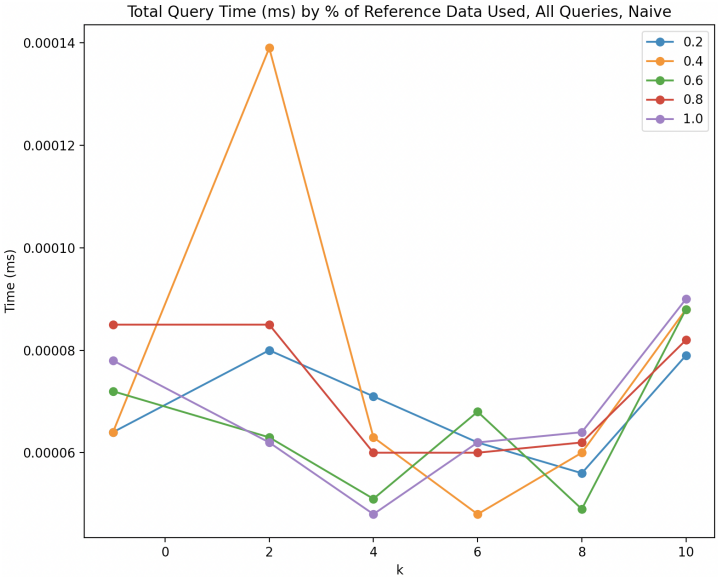


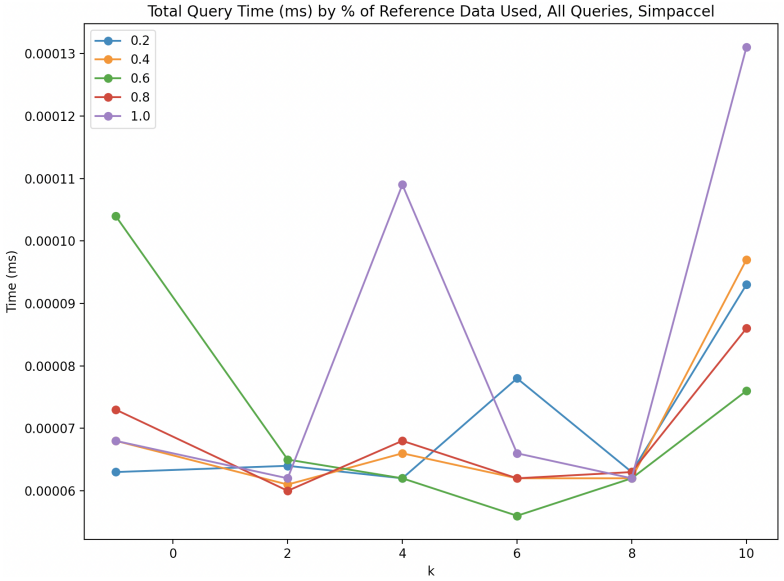
I performed my building experiment by constructing the suffix array over varying percentages of the given reference genome (20% to 100%, in 20% increments), and over varying prefix table sizes, staring with 0 (no prefix table), and up to a prefix of size 10. We can clearly see that the time to construct the suffix array and the file size are essentially directly proportional to the size of the reference. We can also see that the time to build the prefix table significantly impacts out size and our speed once we reach k of size 8 and especially size 10. I used a rather naive approach to building the prefix table, just going through strings until I get out of the upper bound of prefixes, hence the construction was likely longer than it can be overall. I will note, I did expect a little bit more of a linear relationship rather than a sharp rise like this, but I'm not sure if this is significant.

Without using a prefix table, the proportionality of the suffix array is almost one to one with the size of the reference string. (~4.8 MB file for around 4.6 million characters without a prefix table) Thus in theory one could be around the realm of a 30 billion character long genome. The bigger the prefix table however the faster this limit would drop.

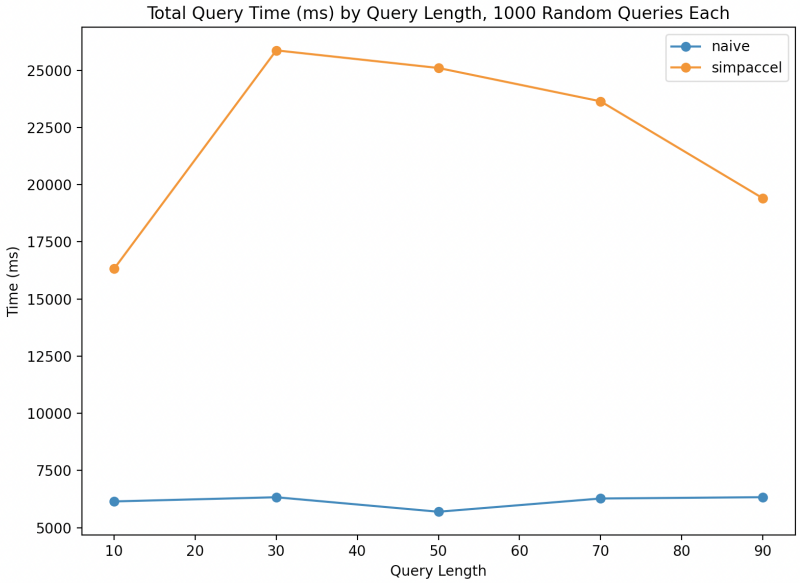
The most challenging part of this was making the serializing work. I managed to get Cereal working for the custom data structures, but the SDSL int vector serialization would not want to load after being serialized. (Even though the exact same steps worked in the first assignment, not sure what's going on there). Thus I save the input file name and reconstruct the SA before querying (excluding time to construct for querying).

Part 2 : Querying





For the first part of the query experiments I measure the time it takes to query all of the entries provided in the query.fa file in dataset, once again varying the levels of percentage of the reference data used, the size of the prefix table, and using the simple accelerant. We can see that the query times (regardless of method) are very very fast (I triple checked, my query results entirely match the file of provided results). Given the naive method there might be a small trend of improvement with a larger prefix table up to a point, but that is difficult to conclude with this amount of data. We see some varying spikes, which are likely either due to cutting of the reference or using different cache types. There doesn't seem to be a significant difference using the simple accelerant or the naive method here.



For a query length experiment, I generated random queries from the given alphabet of size 10, 30, 50, 70, and 90. Running on these queries (this chart was with a prefix table of size 4) took significantly longer than on the provided queries. This is highly likely due to the lack of relationship randomly generated queries would have with the actual dataset. Most queries likely had to exhaustively search. I'm not a fan of the results above frankly. There could be a small bug in my simpaccel that is causing it to behave that way, maybe in my querying from the int vector (suffix array) or checking substrings from the reference, but I'm frankly not sure what's going on.

Given these accelerants and the data we had, it really didn't make much sense to use more memory for these marginal benefits that didn't always even appear. I don't feel comfortable concluding that however, as this would have to be thoroughly tested on other real and synthetic datasets, but from what we've discussed and seen, the accelerations aren't all that significant and can indeed be worse in certain respects. At this point it is honestly highly dependent on how correctly one codes things up as well, making sure there is no accidental writing to memory, an extra O(n) lookup somewhere, etc (such as string appending vs using streams).

The hardest part here was debugging and understanding the simple accelerant. Besides that, if I had the opportunity time wise, I would also generate random queries that are substrings of the existing queries provided and organize them by length, and use those as a non synthetic reference. That would provide a more grounding comparison.