Second meeting (20-09-2022): The string indexing with compressed pattern problem

Mathias Søndergaard, s174426

September 19, 2022

What has been done since last time

The implementation of phrase trie (V1) with $O(n^2)$ space and $O(z+log(n)+occ)$ time was completed and tested. The phrase trie (V2) with $O(n)$ space and same time was implemented and tested. Both were found to be working as expected. Currently, I can handle strings of max size around 15.000. Two clear bottlenecks were identified: My naive LCP implementation and naive suffix compression. Using the theory in [2] I was able to implement a RMQ data structure with $O(n)$ space and prepossessing time, and $O(1)$ query time, thus removing the LCP bottleneck. The problem with my naive suffix compression is that it results in $O(n^2)$ space and time. Finally, test results show that V1 and V2 in general are faster than the suffix tree solution. However, as I cannot handle big strings yet, these results are not that interesting.

Reading wise, I completed section 4 of [1]. Also [2] was read and understood. I have not done any writing yet.

Plans for the next weeks

The goals for the next few weeks is to fix the generalized suffix compression. In [2] you refer to a paper by Keller et al. (Generalized substring compression), in which I can hopefully find better solutions. With a better implementation of generalized suffix compression, I think the phrase trie can handle strings of great size, making the implementation part very interesting. Moreover, I will start writing.

Questions for this meeting

One question regarding the time analysis shown in section 3.1 of [2]. It is stated that they can find the type of a given block in $O(s)$ time, where s is the block length. Here type refers to the "Four-Russians-Trick", which was shown in Algorithms for Massive Data sets as part of $\pm 1RMQ$. The graph

part of the proof makes it clear why $O(s)$ steps are needed. However, each step involves computing a "Ballot number", which has a closed formula containing a binomial coefficient. As far as I know, $\binom{n}{k}$ requires $O(k)$ to compute, and in this case $k = O(s)$. This would result in $O(s^2)$ time. However, using dynamic programming I believe $O(s)$ time is obtainable as the Ballot numbers have a recursive formula. The type of many blocks are computed, which means that DP can be used to significantly speed the process up. Computing the type of all blocks result in the following: $O(s^2 + \frac{n}{s} \cdot s)$. $O(s^2)$ is used to find all needed ballot numbers using the recursive formula. We have $\frac{n}{s}$ blocks. And for each block, $O(s)$ steps are needed when using DP. Here $s = O(log(n))$, resulting in a total complexity of $O(n)$. So, obtaining $O(n)$ for all blocks is possible, but I am not sure about $O(s)$ for each individual block.

References

[1] Philip Bille, Inge Li Gørtz, and Teresa Anna Steiner. "String Indexing with Compressed Patterns". In: (2020).

[2] Johannes Fischer and Volker Heun. "Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE". In: (2006).