A Brief List Of Thesis Topics (Bachelor)

Enno Ohlebusch[*]

November 27, 2018

1 Generalizations of the shortest unique substring problem

A substring of a string S is *unique* if it occurs exactly once in S. The *shortest unique substring problem* is to find all shortest unique substrings of S. Generalizations of the SUS problem are:

- For each position p in S, find the shortest unique substring that begins at p.
- For each position p in S, find all shortest unique substrings that cover position p.

The SUS problem and the first generalization can be solved in linear time and space with the help of the suffix array and the LCP-array of S; see e.g. [6]. The last problem was solved by Ileri et al. [3]. Their solution also uses the suffix array and the LCP-array of S. Your task is to explain the algorithm of Ileri et al. and to implement it in the *C++* library sdsl (succint data structure library). The sdsl supports various algorithms to construct the suffix array as well as the LCP-array, which can be used in the implementation. The practical performance of the implementation should be experimentally compared with other methods and implementations.

2 Computation of all maximal exact matches between two strings based on k-mer indexes

Maximal exact matches (MEMs) are exact matches between two strings S_1 and S_2 that cannot be extended to the left or right without producing a mismatch. A simple strategy to find all MEMs of length at least L is to sample k-mers (substrings of length k) and to store them in a hash table. This must be done in such a way that every MEM contains at least one k-mer that is in the hash table; see [1] for an overview article. Based on this idea, an algorithm to find all MEMs first computes all pairs (p_1, p_2) so that the k-mer starting at position p_1 in S_1 is in the hash table and coincides with the k-mer starting at position p_2 in S_2. Each

[*]Institute of Theoretical Computer Science, University of Ulm, D-89069 Ulm, enno.ohlebusch@uni-ulm.de

such pair (p_1, p_2) represents an exact match of length k, and (using character-by-character comparisons) this match is extended to the left and to the right until a mismatch occurs. The resulting MEM is reported if it has a length at least L. Recently, Grabowski and Bieniecki [2] proposed a novel sampling strategy. Your task is to explain the method of Grabowski and Bieniecki and to provide a prototype implementation. The practical performance of the implementation should be experimentally compared with other methods and implementations.

3 Construction of generalized suffix arrays (suffix arrays for string collections)

Let S be a string of length n. For every i, $1 \leq i \leq n$, S_i denotes the i-th suffix $S[i..n]$ of S. The *suffix array* SA of S is an array of integers in the range 1 to n specifying the lexicographic order of the n suffixes of the string S. That is, it satisfies $S_{\mathsf{SA}[1]} < S_{\mathsf{SA}[2]} < \cdots < S_{\mathsf{SA}[n]}$. In other words, to construct the suffix array of a string S boils down to sorting all suffixes of S lexicographically. We can easily expand the definition of a suffix array to include multiple strings. Let $\mathcal{T}$ be an indexed set of strings and $\mathcal{T}_j$ be element $\mathcal{T}[j]$. We define $\mathsf{SA}_{\mathcal{T}}[i] = (j, k)$ if $\mathcal{T}_j[k..|\mathcal{T}_j|]$ is the i-th lowest suffix among all suffixes of strings from $\mathcal{T}$. In such a generalized suffix array two suffixes can be lexographically equal. In this case, we break ties by comparing the indices of the strings. In other words, we treat each string in $\mathcal{T}$ as if it was terminated by a unique sentinel character $\$_i$ where $\$_i < \$_j$ when $i < j$. Louza et al. [4] recently modified the linear time suffix array construction algorithm SAIS of Nong et al. [5] in such a way that it can construct the suffix array of a string collection. Your task is to explain the algorithm (called gSAIS) of Louza et al. Moreover, gSAIS should be implemented and integrated into the *C*++ library sdsl (succint data structure library).

References

[1] M. Almutairy and E. Torng. Comparing fixed sampling with minimizer sampling when using k-mer indexes to find maximal exact matches. *PLOS One*, 13(2):e0189960, 2018.

[2] S. Grabowski and W. Bieniecki. copMEM: Finding maximal exact matches via sampling both genomes, 2018. `arxiv.org/pdf/1805.08816.pdf`.

[3] A.M. Ileri, M.O. Külekci, and B. Xu. A simple yet time-optimal and linear-space algorithm for shortest unique substring queries. *Theoretical Computer Science*, 562:621–633, 2014.

[4] F.A. Louza, S. Gog, and P. Telles. Inducing enhanced suffix arrays for string collections. *Theoretical Computer Science*, 678:22–39, 2017.

[5] G. Nong, S. Zhang, and W.H. Chan. Linear suffix array construction by almost pure induced-sorting. In *Proc. Data Compression Conference*, pages 193–202. IEEE Computer Society, 2009.

[6] E. Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction.* Oldenbusch Verlag, 2013.