Master's Thesis
LCP-Array Construction in Distributed Memory

Overview

The *suffix array* (SA) of a text contains the starting positions of all of its suffixes in lexicographical order, see Fig. 1 for an example. The SA is one of the most well-researched text indices. It has applications in different string algorithms, among others, as a full-text index and in text compression. For many of these applications, the SA has to be combined with auxiliary information stored in additional arrays.

The *longest common prefix array* (LCP-array) is such an array. It contains the sizes of the longest common prefix of all lexicographically consecutive suffixes, see also Fig. 1. The LCP-array can be used to speed up pattern matching queries in the SA and to compute a compressed version of the text, as it contains information about redundant parts of the text.

All LCP-array construction algorithms can be divided into two approaches. The first one is to compute it alongside the SA, using information that is gained by comparing suffixes to sort them to compute entries in the LCP-array. While this approach allows information to be used twice, it introduces running time and memory overhead, as the information for computating of both arrays has to be stored at the same time.

The alternative for constructing the LCP-array is to construct the SA first and to compute the LCP-array afterwards using the SA. One advantage that arises from this approach is that the LCP-array construction does not depend on the SA construction. Thus, not every SA construction algorithm has to be modified to compute the LCP-array and the best SA construction algorithm can be used to compute the LCP-array. In practice, this approach is faster (and more memory efficient) on most inputs in many models of computation, e.g., main memory [2], shared memory [6], and external memory [5].

	0	1	2	3	4	5	6	7	8	9	10	11
T	m	i	s	s	i	s	s	i	p	p	i	$
SA	11	10	7	4	1	0	9	8	6	3	5	2
LCP	0	0	1	1	4	0	0	1	0	2	1	3

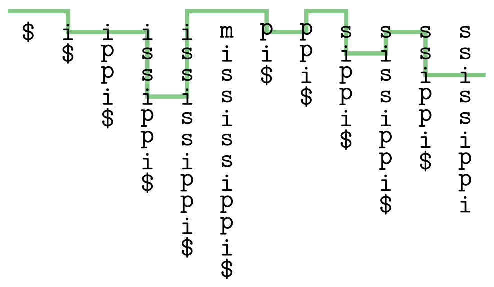

Figure 1: SA and LCP-array for the text $T =$ `mississippi$`. The LCP-values are marked in green (🟢).

Requirements

- Excellent C++ programming skills
- Interest in distributed memory and string algorithms

Objective

The main objective of this Master's thesis is to experimentally evaluate whether the computation of the LCP-array based on the SA is also faster than the simultaneous construction in distributed memory. To this end, at least one distributed memory LCP-array construction algorithm that uses the SA has to be developed, because at the present time, there exists only one implementation of a distributed memory LCP-array construction algorithm [4], which computes the SA and LCP-array at the same time. Existing distributed memory SA construction algorithms [1, 3, 4] can be used as staring point.

Contact

Dr. Florian Kurpicz (kurpicz@kit.edu)

References

[1] Timo Bingmann, Simon Gog, and Florian Kurpicz. Scalable construction of text indexes with thrill. In *IEEE BigData*, pages 634–643. IEEE, 2018.

[2] Johannes Fischer and Florian Kurpicz. Dismantling divsufsort. In *Stringology*, pages 62–76. Department of Theoretical Computer Science, Czech Technical University in Prague, 2017.

[3] Johannes Fischer and Florian Kurpicz. Lightweight distributed suffix array construction. In *ALENEX*, pages 27–38. SIAM, 2019.

[4] Patrick Flick and Srinivas Aluru. Parallel distributed memory construction of suffix and longest common prefix arrays. In *SC*, pages 16:1–16:10. ACM, 2015.

[5] Juha Kärkkäinen and Dominik Kempa. LCP array construction in external memory. *ACM J. Exp. Algorithmics*, 21(1):1.7:1–1.7:22, 2016.

[6] Julian Shun. Fast parallel computation of longest common prefixes. In *SC*, pages 387–398. IEEE Computer Society, 2014.