Bachelor's Thesis

Parallel Vectorized Wavelet Tree Construction

Overview

The *FM-index* combines two data structures: the *Burrows-Wheeler transform* and *wavelet trees*. It is a very prominent full-text index and used in most DNA read aligners [6] and in Bioinformatics in general. In this Master's thesis, we focus on the efficient construction of the second data structure—wavelet trees. The wavelet tree is a binary tree data structure that can be used to answer *rank* and *select* queries on texts of size n over an alphabet of size σ in $O(\lg \sigma)$ time. Here, $rank_\alpha(i)$ queries ask for the number of occurrences of the symbol α before the position i and $select_\alpha(i)$ queries return the text position of the i-th occurrence of the symbol α.

Let T be a text of length n over an alphabet of size σ. The corresponding wavelet tree consists of $\lceil \lg \sigma \rceil$ bit vectors of size n, see Fig. 1. Even though all $n\lceil \lg \sigma \rceil$ entries in the bit vectors have to be looked at during construction, the wavelet tree can be computed in $O(n \lg \sigma / \sqrt{\lg n})$ time using broad-word programming [1, 7]. There exists an implementation of such an algorithm by Dinklage et al. [3], which heavily relies on specialized CPU instructions like *parallel bit extract* and *packed shuffle bytes*. The reported construction times are faster than the previously fastest sequential WT construction algorithm [2, 4]. However, the algorithm has one significant disadvantage—it does not scale well using multiple CPU cores.

Objective

The main objective of this Master's thesis is to develop a fast *and* scaling wavelet tree construction algorithm that computes the wavelet tree in $O(n \lg \sigma / \sqrt{\ln n})$ work using specialized CPU instructions. To this end, space-efficient and bit vectors should be used [5]. Making this algorithm space-efficient is another (minor) goal of this Bachelor's thesis.

Requirements

- Excellent C++ programming skills
- Interest in string algorithms and compact data structures

Contact

Dr. Florian Kurpicz (kurpicz@kit.edu)

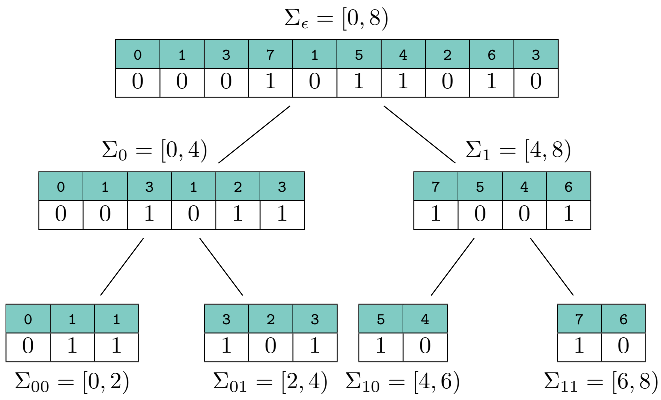

Figure 1: The wavelet tree of $T = [0, 1, 3, 7, 1, 5, 4, 2, 6, 3]$. The light teal (●) arrays contain the characters represented at the corresponding position in the bit vector and are not a part of the wavelet tree. Σ_α denotes the characters that are represented by the bit vector for $\alpha \in \{\epsilon, 0, 1, 00, 01, 10, 11\}$. All this auxiliary information is not stored explicitly.

References

[1] Maxim A. BabenkoMunroNV2016WT, Pawel Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *SODA*, pages 572–591. SIAM, 2015.

[2] Patrick Dinklage, Jonas Ellert, Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Practical wavelet tree construction. *ACM J. Exp. Algorithmics*, 26:1.8:1–1.8:67, 2021.

[3] Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. Bit-parallel (compressed) wavelet tree construction. In *DCC*, pages 81–90. IEEE, 2023.

[4] Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Simple, fast and lightweight parallel wavelet tree construction. In *ALENEX*, pages 9–20. SIAM, 2018.

[5] Florian Kurpicz. Engineering compact data structures for rank and select queries on bit vectors. In *SPIRE*, volume 13617 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2022.

[6] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357, 2012.

[7] J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theor. Comput. Sci.*, 638:91–97, 2016.