Speeding up Online String Matching by Partial Indexing

Stefano Scafiti[1]

[1]) Dipartimento di Matematica e Informatica,
Università di Catania, viale A.Doria n.6, 95125, Catania, Italia
stefano.scafiti@studium.unict.it

The *string matching problem* is a fundamental task in the field of text processing. Formally, it consists in finding all the occurrences of a given pattern x, of length m within a text y of length n, both strings over a common alphabet Σ of size σ.

Depending on whether the input text is preprocessed or not, two class of solutions to the problem are possible. *Online string matching* [3] assume the input text is not known in advance and perform an online full scan of the text when searching for a specific pattern. Despite memory requirements of this class of algorithms are limited to a precomputed table, whose size is usually proportional to the length of the pattern or to the size of the alphabet, performance may stay poor in practice, expecially when processing huge texts. On the contrary, *offline string matching* tries to drastically speed up searching by preprocessing the text and building a data structure that allows searching in time proportional to the length of the pattern [1, 6]. For this reason such kind of problem is also known as *indexed searching*. However, despite their optimal time performance, space requirements of full-index data structures, as suffix-trees and suffix-arrays, are from 4 to 20 times the size of the text [5]. This is why, *partial-indexed searching* is usually preferred in practical applications, reaching a tradeoff between space requirements and time efficiency. The purpose of a partial-index is to significantly speed up online string matching using reduced extra informations. For example, in the *sampled string matching* approach [2, 4], searching operations are performed on a sampled version of the original text, thus reducing the amount of time needed to process large collections of texts. A part from speeding up online searching, in order to be of any practical and theoretical interest, a partial-index of the text should be *succint*, *fast to build* and allow for *fast update*.

In this work, a partial index of the text inspired by the suffix-array data structure is proposed. Suffix array still remains the data structure of reference in the field of full-text indexed searching. Since it has been introduced [4], several variations of the original data structure have been developed, mainly aiming at reducing space requirements without significantly affecting search time. However, for the reasons discussed above, storing a suffix array for an entire text can be prohibitive. The idea of the new partial index is to ease space requirements by only indexing partial informations of the text, instead of the entire set of suffixes.

Given an integer $k > 1$, the input text y is virtually split into $k+1$ blocks, $\langle y^{(1)}, y^{(2)}, ..., y^{(k)}, y^{(k+1)} \rangle$, each of size $\lfloor n/k \rfloor$ (except for the last, which is possibly empty and contains $n - \lfloor n/k \rfloor$ elements). Also, given a positive parameter q, the set formed by all the distinct q-grams of the text is computed and stored in a lexicographically sorted array S. Each unique q-gram x is also associated with a bitvector $B[1..k]$ of size k, where $B[i] = 1$, $0 \leq i \leq k$, if and only if an occurrence of x starts within the i-th chunk of the text.

Given a pattern x of size $m \geq q$, searching begins by locating the bitvector B_x corresponding to $x[0..q-1]$. This can be accomplished by performing a binary search on S. Once B_x has been located, a full scan is required for a block $y^{(i)}$ only if $B_x[i] = 1$, thus allowing for a restricted subset of blocks to be processed.

Experiments show that the presented solution allows to save about 96% of the space needed to store a suffix-array, speeding up searching up to 90%.

References

[1] Alberto Apostolico. The Myriad Virtues of Subword Trees. In Alberto Apostolico and Zvi Galil, editors, *Combinatorial Algorithms on Words*, pages 85–96, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[2] Francisco Claude, Gonzalo Navarro, Hannu Peltola, Leena Salmela, and Jorma Tarhio. String matching with alphabet sampling. *Journal of Discrete Algorithms*, 11:37–50, 2012.

[3] Simone Faro and Thierry Lecroq. The Exact Online String Matching Problem: A Review of the Most Recent Results. *ACM Comput. Surv.*, 45(2), March 2013.

[4] Simone Faro, Francesco Marino, and Arianna Pavone. Efficient Online String Matching Based on Characters Distance Text Sampling. *Algorithmica*, pages 1–23, 2020.

[5] Stefan Kurtz. Reducing the Space Requirement of Suffix Trees. *Softw. Pract. Exper.*, 29(13):1149–1171, November 1999.

[6] Udi Manber and Gene Myers. Suffix Arrays: A New Method for on-Line String Searches. *SIAM J. Comput.*, 22(5):935–948, October 1993.