Theoretical Computer Science 638 (2016) 112–121

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computing minimal and maximal suffixes of a substring ☆

Maxim Babenko [a], Paweł Gawrychowski [b,1], Tomasz Kociumaka [b,2], Ignat Kolesnichenko [c], Tatiana Starikovskaya [d,*]

[a] National Research University Higher School of Economics, Myasnitskaya ul. 20, Moscow, 101000, Russia
[b] Institute of Informatics, University of Warsaw, Stefana Banacha 2, 02-097 Warsaw, Poland
[c] Moscow Institute of Physics and Technology, Institutskiy per. 9, Dolgoprudnyy, 141701, Russia
[d] University of Bristol, Senate House, Tyndall Avenue, Bristol, BS8 1TH, United Kingdom

ARTICLE INFO

Article history:
Received 1 April 2015
Received in revised form 12 July 2015
Accepted 24 August 2015
Available online 28 August 2015

Keywords:
Data structures
Substring queries
Lexicographic order
Minimal suffix
Maximal suffix

ABSTRACT

We consider the problems of computing the maximal and the minimal non-empty suffixes of substrings of a longer text of length n. For the minimal suffix problem we show that for every τ, $1 \le \tau \le \log n$, there exists a linear-space data structure with $\mathcal{O}(\tau)$ query time and $\mathcal{O}(n \log n/\tau)$ preprocessing time. As a sample application, we show that this data structure can be used to compute the Lyndon decomposition of any substring of the text in $\mathcal{O}(k\tau)$ time, where k is the number of distinct factors in the decomposition. For the maximal suffix problem, we give a linear-space structure with $\mathcal{O}(1)$ query time and $\mathcal{O}(n)$ preprocessing time. In other words, we simultaneously achieve both the optimal query time and the optimal construction time.

1. Introduction

Computing the lexicographically maximal and minimal suffixes of a string is both an interesting problem on its own and a crucial ingredient in solutions to many other problems. For example, the famous constant-space pattern matching algorithm of Crochemore and Perrin and its more recent variants are based on the so-called critical factorizations, which can be derived from the maximal suffixes [1,2].

The first non-trivial solution of the maximal and minimal suffix problems is due to Weiner, who introduced the suffix tree [3]. The suffix tree of a string can be constructed in linear time and occupies linear space. Once constructed, it allows to retrieve the maximal and the minimal suffixes in constant time. Later, this result was improved by Duval [4] who showed that the suffixes can be found in linear time and constant additional space.

We consider a natural generalization of these problems. We assume that the strings we are asked to compute the maximal or the minimal suffixes for are actually substrings of a text T and that they are specified by their endpoints in T. Then, one can preprocess T and subsequently use this information to significantly speed up the computation of the desired

☆ This article is based on a study first reported at 24th and 25th Symposiums on Combinatorial Pattern Matching.

* Corresponding author.
E-mail addresses: maxim.babenko@gmail.com (M. Babenko), gawry@mimuw.edu.pl (P. Gawrychowski), kociumaka@mimuw.edu.pl (T. Kociumaka), ignat@yandex-team.ru (I. Kolesnichenko), tat.starikovskaya@gmail.com (T. Starikovskaya).

[1] Currently holding a post-doc position at Warsaw Center of Mathematics and Computer Science.

[2] Supported by Polish budget funds for science in 2013–2017 as a research project under the 'Diamond Grant' program (Ministry of Science and Higher Education, Republic of Poland, grant number DI2012 01794).

http://dx.doi.org/10.1016/j.tcs.2015.08.023
0304-3975/

suffixes of a query string. This seems to be a very natural setting whenever one thinks of storing large collections of static text data.

Let n be the length of T. We first show that for every τ, $1 \le \tau \le \log n$, there exists a linear-space data structure solving the minimal suffix problem with $\mathcal{O}(\tau)$ query time and $\mathcal{O}(\frac{n \log n}{\tau})$ preprocessing time. Secondly, we describe a linear-space data structure for the maximal suffix problem with $\mathcal{O}(1)$ query time which can be constructed in linear time. As a particular application, we show how to compute the Lyndon decomposition [5] of a substring of T in $\mathcal{O}(k\tau)$ time, where k is the number of distinct factors in the decomposition.

The key idea of our solution is to select, for each position j of the text T, a set of $\mathcal{O}(\log n)$ *canonical substrings* — substrings of T that end at j such that the lengths of two consecutive canonical substrings differ by a factor of at most 2. Note that for substrings with a fixed end-position, the maximal suffix becomes larger as the length of a substring increases, while the minimal suffix becomes smaller. Thus, for a query $x = T[i..j]$ we know that either the answer is the same as for the longest canonical suffix of x, or the resulting suffix is longer than $|x|/2$. For the latter case we develop a subroutine which exploits periodicities to compute the maximal (resp. minimal) suffix given its approximate length (within a factor of 2). The answers for canonical substrings are stored in $\mathcal{O}(\log n)$ bits for each position j. These bits let us to retrieve approximate lengths only; the exact answers are computed as in the previous case.

The basic $\mathcal{O}(n \log n)$-time construction algorithm computes the answers for all canonical substrings. However, for maximal suffixes we develop a linear-time construction algorithm. This is possible mainly due to the following fact: the length of the maximal suffix of a string cannot increase by more than one when a single letter is appended at the end. Minimal suffixes do not enjoy such a property, e.g., when `aa` is extended to `aab` the length of the minimal suffix increases from 1 to 3.

Related work. Text indexes that support various substring queries have been extensively studied in the literature. The study dates back to the invention of the suffix tree. Augmented properly, the suffix tree can be used to answer the substrings equality and the longest common prefix queries in constant time and linear space [6].

Cormode and Muthukrishnan [7] initiated a study on substring compression problems, where the goal is to quickly find the compressed representation or the compressed size for a given substring of the text. Some of their results were later improved in [8] and [9].

Recently, substring queries gained more attention. It has been shown that various periodicity-related queries can be answered in logarithmic or constant time [10,11,9]. Some of these results apply a linear-space data structure for internal pattern matching queries, which are to find all occurrences of one substring of the text in another substring [9]. Yet another type of substring queries is range LCP queries studied in [12,13].

Queries asking for the k-th lexicographically smallest suffix of a substring, more general than both the minimal and the maximal suffix queries, have also been studied. They can be answered in $\mathcal{O}(\log n)$-time by a wavelet suffix tree, a linear space data structure which admits an $\mathcal{O}(n\sqrt{\log n})$-time construction algorithm [14]. However, wavelet suffix trees are less efficient and much more involved than the data structures we specifically design for minimal and maximal suffix queries.

2. Preliminaries

We start by introducing some standard notation and definitions. Let Σ be a finite non-empty set (called an *alphabet*). The elements of Σ are *letters*. A finite ordered sequence of letters (possibly empty) is called a *string*. Letters in a string are numbered starting from 1, that is, a string T of *length* k consists of letters $T[1], T[2], \ldots, T[k]$. The length of T is denoted by $|T|$. For $i \le j$, $T[i..j]$ denotes the *substring* of T from position i to position j (inclusive). If $i = 1$ or $j = |T|$, then we omit these indices and we write $T[..j]$ and $T[i..]$. Substring $T[..j]$ is called a *prefix* of T, and $T[i..]$ is called a *suffix* of T.

A *border* of a string T is a string that is both a prefix and a suffix of T but differs from T. A string T is called *periodic with period* ρ if $T = \rho^s \rho'$ for an integer $s \ge 1$ and a (possibly empty) proper prefix ρ' of ρ. Borders and periods are dual notions: if T has period ρ, then it has a border of length $|T| - |\rho|$, and vice versa; see, e.g., [15].

Fact 1. *(See [16].) If a string T has periods ρ and γ such that $|\rho| + |\gamma| \le |T|$, then T has a period of length* $\gcd(|\rho|, |\gamma|)$*, the greatest common divisor of ρ and γ.*

Lemma 2. *If a string T has a proper border, then its shortest border has length at most $|T|/2$.*

Proof. Suppose that the shortest non-empty border of T has length larger than $|T|/2$, then by border-period duality T has a period ρ smaller than $|T|/2$. Since 2ρ is also a period and $2\rho < |T|$, we get another (shorter) border of T, a contradiction. □

We assume the word RAM model of computation [17] with word size $\Omega(\log n)$. Letters are treated as integers in range $\{1, \ldots, |\Sigma|\}$, so a pair of letters can be compared in $O(1)$ time. We also assume $\Sigma = n^{\mathcal{O}(1)}$ so that all letters of the input text T can be sorted in $\mathcal{O}(n)$ time. The natural linear order on Σ is extended in a standard way to the *lexicographic* order of strings over Σ. Namely, $T_1 \prec T_2$ if either