Discrete Applied Mathematics 163 (2014) 258–267

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

New simple efficient algorithms computing powers and runs in strings

M. Crochemore [a,c], C.S. Iliopoulos [a,d], M. Kubica [e], J. Radoszewski [e,*,1], W. Rytter [e,f], K. Stencel [e,f], T. Waleń [b,e]

[a] King's College London, London WC2R 2LS, UK

[b] Laboratory of Bioinformatics and Protein Engineering, International Institute of Molecular and Cell Biology in Warsaw, Ks. Trojdena 4, 02-109 Warsaw, Poland

[c] Université Paris-Est, France

[d] Digital Ecosystems & Business Intelligence Institute, Curtin University of Technology, Perth WA 6845, Australia

[e] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

[f] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland

ARTICLE INFO

Article history:
Received 10 December 2011
Received in revised form 17 February 2013
Accepted 20 May 2013
Available online 14 June 2013

Keywords:
Run in a string
Square in a string
Cube in a string
Dictionary of Basic Factors

ABSTRACT

Three new simple $O(n \log n)$ time algorithms related to repeating factors are presented in the paper. The first two algorithms employ only a basic textual data structure called the Dictionary of Basic Factors. Despite their simplicity these algorithms not only detect existence of powers (in particular, squares) in a string but also find all primitively rooted cubes (as well as higher powers) and all cubic runs. Our third $O(n \log n)$ time algorithm computes all runs and is probably the simplest known efficient algorithm for this problem. It uses additionally the Longest Common Extension function, however, due to relaxed running time constraints, a simple $O(n \log n)$ time implementation can be used. At the cost of logarithmic factor (in time complexity) we obtain novel algorithmic solutions for several classical string problems which are much simpler than (usually quite sophisticated) linear time algorithms.

1. Introduction

We present algorithms finding various types of repetitions in a string: powers (e.g., squares or cubes), cubic runs and runs. The k-th power of a string u, u^k, is simply composed of k juxtaposed occurrences of this string. The square and the cube are obviously the second and the third power. A string u is called periodic with the period p if $u_i = u_{i+p}$ holds for all i. A run is a periodic factor of the string u in which the shortest period repeats at least twice. A run must be maximal, i.e., if extend it by one symbol its period increases. A cubic run is similar, but is has to contain at least 3 occurrences of the period. Finding powers and runs in a given string is a fundamental problem in text processing and has numerous applications: it occurs frequently in pattern matching, text compression and computational biology. A more detailed explanation of the motivation and related topics can be found in the survey [8].

Multiple algorithms for finding various kinds of repetitions in a string have already been presented. The largest part of the related literature deals with different approaches to searching for squares in a string. Most of the existing algorithms

* Corresponding author. Tel.: +48 22 55 44 484; fax: +48 22 55 44 400.
E-mail addresses: maxime.crochemore@kcl.ac.uk (M. Crochemore), c.iliopoulos@kcl.ac.uk (C.S. Iliopoulos), kubica@mimuw.edu.pl (M. Kubica), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), stencel@mimuw.edu.pl (K. Stencel), walen@mimuw.edu.pl (T. Waleń).

[1] Some parts of this paper were written during the corresponding author's Erasmus exchange at King's College London.

0166-218X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dam.2013.05.009

are rather complex. The first approach is to check if a string contains any square factor at all, or, otherwise, is square-free. Denote by n the length of the considered string. $O(n \log n)$ time algorithms for square-free testing are presented in [24,25] (the latter one is randomized). The optimal $O(n)$ time algorithms are described in [5,24].

For the problem of finding all distinct squares, linear time algorithms are known [10,16,19]. It is also known that the maximal number of distinct squares in a string is linear with respect to the length of the string: this number never exceeds $2n$ [15].

Another approach is to find all occurrences of primitively rooted squares in a string, that is, factors of the string in which the shortest period occurs exactly twice. A number of $O(n \log n)$ time algorithms reporting all such occurrences can be found in [2,4,20,23,26].

Due to the lower bound shown in [6] these algorithms are optimal.

Yet another approach is to report simply all occurrences of squares in a string. Denote the number of such occurrences by z, note that z could be $\Theta(n^2)$. Both $O(n \log n + z)$ time algorithms [21,23,26] and $O(n + z)$ time algorithms [16,19] are known for this problem.

Finally, there are recent results related to on-line square detection (that is, when the symbols of the string are given one by one), improving the time complexity from $O(n \log^2 n)$ [22] to $O(n \log n)$ [18] and $O(n)$ [3].

Let u be a string of length n over a bounded alphabet. In Section 3 a very simple $O(n \log n)$ time algorithm checking whether u contains any k-th string power is presented. The algorithm utilizes a simple, yet powerful textual data structure called the Dictionary of Basic Factors. The algorithm also reports all occurrences of primitively rooted k-th powers for any $k \geq 3$, in particular, primitively rooted cubes. As a by-product we obtain an alternative, algorithmic proof of the fact [6] that the maximal number of such occurrences in a string of length n is $O(n \log n)$.

From the aforementioned literature, the papers [2,4,23] deal also with powers of arbitrary (integer) exponent, however the techniques used there (e.g., suffix trees, Hopcroft's factor partitioning) are much more sophisticated than the techniques applied in this paper. The $O(n \log n)$ time algorithm for a single square detection from [24] is in some sense similar to the algorithm presented in this paper. However, it is less versatile than ours. We see no simple modification adapting the algorithm from [24] to detect all occurrences of primitively rooted higher powers.

In Section 4 we present an application of our power-detecting algorithm to find all cubic runs in a string. This algorithm also works in $O(n \log n)$ time and it does not use any additional advanced techniques.

Finally, in Section 5 we give an algorithm reporting all runs in a string in $O(n \log n)$ time. It is significantly simpler than all known $O(n \log n)$ time algorithms present implicitly in [2,4,23] and than the optimal $O(n)$ time algorithm [19]. The only non-trivial technique used in our algorithm is the Longest Common Extension function. It can be either implemented as described in [12,17] – very efficiently, but using quite sophisticated machinery – or in a much simpler way, using the Dictionary of Basic Factors, which is sufficient to obtain $O(n \log n)$ time complexity.

This paper is a full version of the paper [9].

2. Preliminaries

We consider *strings* (words) over a bounded alphabet Σ. The empty string is denoted by ε. The positions in u are numbered from 1 to $|u|$. For $u = u_1 u_2 \dots u_n$, by $u[i..j]$ we denote a *factor* of u equal to $u_i \dots u_j$ (in particular $u[i] = u[i..i]$). Strings $u[1..i]$ are called prefixes of u, strings $u[i..n]$ are called suffixes of u, whereas strings that are both a prefix and a suffix of u are called *borders* of u.

We say that a positive integer p is a *period* of the string $u = u_1 \dots u_n$ if $u_i = u_{i+p}$ holds for all i, $1 \leq i \leq n - p$. Periods and borders correspond to each other, i.e., u has a period p if and only if it has a border of length $n - p$; see, e.g., [7,14].

Let us consider positions between consecutive letters of u. We say that a square w^2 *is centered* at inter-position between u_i and u_{i+1} if:

- w is a suffix of $u[1..i]$ or $u[1..i]$ is a suffix of w, and
- w is a prefix of $u[i + 1..n]$ or $u[i + 1..n]$ is a prefix of w.

The *local period* at inter-position between u_i and u_{i+1} is the length of the shortest nonempty string centered at this inter-position.

A *run* (also called a maximal repetition) in a string u is such an interval $[i..j]$, that:

- the shortest period p of the associated factor $u[i..j]$ satisfies $2p \leq j - i + 1$,
- the interval can be extended neither to the left nor to the right, without violating the above property, that is, $u[i - 1] \neq u[i + p - 1]$ and $u[j - p + 1] \neq u[j + 1]$, provided that the respective symbols exist.

We identify a run with a corresponding triple (i, j, p). $\frac{j-1+1}{p}$ is called the *exponent* of a run (i, j, p). A *cubic run* is a run $[i..j]$ for which the shortest period p satisfies $3p \leq j - i + 1$. Fig. 1 shows all the runs and cubic runs in a sample string.

If $w^k = u$, where u and w are nonempty strings and k is a positive integer, then we say that u is the k-th power of the string w. A *square* (*cube*) is the 2nd (3rd) power of a string. A string v is called *primitive* if there is no string w and integer $k \geq 2$ such that $v = w^k$. A k-th power w^k is *primitively rooted* if w is primitive.