Theoretical Computer Science 700 (2017) 45–62

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Locating maximal approximate runs in a string ☆

Mika Amit [a,*], Maxime Crochemore [c,d], Gad M. Landau [a,b], Dina Sokol [e]

[a] Department of Computer Science, University of Haifa, Mount Carmel, Haifa, Israel
[b] Department of Computer Science and Engineering, NYU Polytechnic School of Engineering, New York University, Brooklyn, NY, USA
[c] King's College London, Strand, London WC2R 2LS, UK
[d] Université Paris-Est, Institut Gaspard-Monge, 77454 Marne-la-Vallée Cedex 2, France
[e] Department of Computer and Information Science, Brooklyn College of the City University of New York, Brooklyn, NY, USA

ARTICLE INFO

Article history:
Received 23 January 2017
Received in revised form 24 June 2017
Accepted 29 July 2017
Available online 4 August 2017
Communicated by L.M. Kirousis

Keywords:
Algorithms on strings
Pattern matching
Repetitions
Tandem repeats
Runs

ABSTRACT

An exact run in a string T is a non-empty substring of T that is a repetition of a smaller substring possibly followed by a prefix of it. Finding maximal exact runs in strings is an important problem and therefore a well-studied one in the area of stringology. For a given string T of length n, finding all maximal exact runs in the string can be done in $O(n \log n)$ time on general ordered alphabets or $O(n)$ time on integer alphabets. In this paper, we investigate the maximal approximate runs problem: for a given string T and a number k, find non-empty substrings T' of T such that changing at most k letters in T' transforms them into a maximal exact run. We present an $O(nk^2 \log^2 k + occ)$ algorithm to solve this problem, where occ is the number of substrings found.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Periodicities and repetitions are ubiquitous in nature, and they play a central role in the field of stringology. They are used to obtain efficient algorithms for pattern matching problems, to conserve space via text compression, and to better analyze data, e.g. in biological sequences. The research of repetitions and their characteristics has been thoroughly investigated for both exact and approximate ones.

Exact repetitions: Several methods are available to detect all the occurrences of exact squares in strings, where a *square* is defined as exactly two consecutive copies of a pattern (see [6,2,25]). For a given string T, of length n, these algorithms run in $O(n \log n)$ time, which is optimal since it is possible for a string to contain $\Omega(n \log n)$ squares. Selecting some of their occurrences, or just *distinct* squares, regardless of their number of occurrences, paved the path to faster algorithms [21,14] (for constant alphabets) and [4] (for integer alphabets).

Runs have been introduced by Iliopoulos, Moore, and Smyth [15], and are defined as repetitions with two or more consecutive copies of a pattern. They showed that Fibonacci words contain only a linear number of maximal runs. Kolpakov and Kucherov [19] (see also [8], Chapter 8) proved that this property holds for any string. In [3], the authors provided a

☆ A preliminary version appeared in the proceeding of CPM 2013.

* Corresponding author.
E-mail addresses: mika.amit2@gmail.com (M. Amit), maxime.crochemore@kcl.ac.uk (M. Crochemore), landau@cs.haifa.ac.il (G.M. Landau), sokol@sci.brooklyn.cuny.edu (D. Sokol).

http://dx.doi.org/10.1016/j.tcs.2017.07.021
0304-3975/© 2017 Elsevier B.V. All rights reserved.

simple and elegant proof that the number of maximal runs in a string of length n is at most $n-3$. Recently, in [11], it was shown that for binary strings this number is bounded by $0.957n$.

In [19], the authors designed an algorithm to compute all maximal runs in a string of length n over an alphabet Σ. The time complexity of this algorithm is $\mathcal{O}(n \log |\Sigma|)$. Their algorithm extends Main's algorithm [24], which itself extends the method in [7] (see also [8]).

The design of a linear-time algorithm for building the Suffix Array of a string on an integer alphabet (see [16–18]) and the introduction of another related data structure (the Longest Previous Factor table in [9]) have eventually led to a linear-time solution (for integer alphabet) for computing all maximal runs in a string. This is a consequence of the linear-time computation of the Ziv–Lempel factorization on integer alphabets (see [1] and [5]), which removed the $O(n \log |\Sigma|)$ time bottleneck in the Kolpakov–Kucherov algorithms [19]. A recent algorithm by Bannai et al. [3] uses similar tools and also runs in linear time. On an ordered alphabet, namely where letters can be compared w.r.t. a linear order, the optimal computing time is $O(n \log n)$ [25,10].

Approximate repetitions: In many applications, finding *approximate* runs is more sensible than finding exact runs. A typical example is genetic sequence analysis. This problem was widely researched and many different measurements have been used in order to find such runs. In [27], a k-approximate run is defined as follows: a string x is an approximate run if there exists a consensus string u such that x can be divided into a number of adjacent occurrences of substrings $x = u_1 u_2 \cdots u_t$ where the distance between u and every u_i is not greater than k. In this version of the problem, the difference between two periods u_i and u_j can be as big as $2k$, for example: the string $x = bacd\ abdc\ cbad$ is a 2-approximate run, since the difference between the substring $u = abcd$ and each u_i, $1 \le i \le 3$ is exactly 2. [27] provide an $O(n^3)$-time algorithm for finding all such maximal repetitions in an input string of size n.

A different approach to the problem is defined as follows [13]: given a string x and an integer k, x is an approximate run if it can be divided into a number of adjacent substrings $x = u_1 u_2 \cdots u_t$, such that the sum over all distances between adjacent substrings, u_i and u_{i+1}, is not greater than k. In this version, the first period and the last period can be completely different from each other. For instance: the string $x = abcd\ dbcd\ dccd\ dcbd\ dcba$ is a 4-approximate run. [13] provide an $O(n^2)$-time algorithm for finding such maximal approximate runs in a string. This version of the problem can be extended to the problem where the sum over all distances between *every* two substrings u_i and u_j in x (for $1 \le i < j \le t$) cannot exceed k. In this case, all substrings u_i, $1 \le i \le t$ must be similar to each other, as one error between two substrings u_i and u_j may imply $O(t)$ errors. For example, the string $x = (abcd)^{t-1} abed$ is a $(t-1)$-approximate run according to this definition.

In [22] another definition of approximate run is given: a substring x is a k-approximate run if $x = u_1 u_2 \cdots u_t$ and the removal of the same k positions from each u_i will generate an exact run. According to this definition, any number of mismatches in the same column of the period is counted as 1 mismatch. For example, the string $x = abcd\ abdd\ abbd\ abad$ is a 1-approximate run. The algorithm for finding such repetitions [22] has time complexity $O(nka \log(n/k))$, where n is the length of the input string, k is the number of allowed error columns, and a is the maximum number of periods in any found repeat.

In this paper we introduce a novel, more global definition of an approximate run. Informally, in our problem we count the total number of letters that need to be replaced in order to generate an exact run. A k-approximate run can be transformed into an exact run through the modification of at most k letters. This definition is similar to the one presented in [27], as it finds a consensus string u that is similar to all substrings u_i of x. But, as opposed to the former version, this version sums the *total* number of differences between all u_i and u, which requires the substrings to be more similar to each other. The formal definition of the problem is given in Section 2. For example, the substring $x = bacd\ abdc\ cbad$, that contains periods that are very different from each other, is a 2-approximate run according to the former definition, and in our problem definition, it is a 6-approximate run. Note that the substring $x = abcd\ aadd\ abcd\ abcd$ is a 2-approximate run according to both definitions. In this paper we present an $O(nk^2 \log^2 k + occ)$-time algorithm to find such maximal approximate runs in a given input string of length n, where occ is the number of maximal approximate runs that are found.

Roadmap: We start in Section 2 with definitions and notations that will be used throughout the paper. In Section 3, we present the main procedure of our algorithm. Initially, in subsection 3.1, we describe a simple $O(n)$ algorithm for the main procedure, and then in subsection 3.4 we present an improved $O(k^3)$ algorithm for it. In Section 4, we describe the efficient $O(k^2 \log k)$-time algorithm for the main procedure. Finally, in Section 5, we present the entire algorithm for searching a given input string of length n for maximal approximate runs with k modifications.

2. Definitions and notation

Let $T = T[1]T[2]\cdots T[n]$ be a string of size n defined over the constant size alphabet Σ. We denote the substring of T that starts at position i and ends at position j as $T_{i,j} = T[i]T[i+1]\cdots T[j]$. A position h is *contained* in the substring $T_{i,j}$ if $i \le h \le j$. The following definitions are needed in order to formally define the problem we solve in the paper.

Exact run. An *exact run* is a non-empty string, x, that can be written as $x = u_1 u^t u_2$, where $t \ge 2$, the first substring u_1 is a (possibly empty) suffix of u, and the last substring u_2 is a (possibly empty) prefix of u. u is called a *period* and its length

Download English Version:

https://daneshyari.com/en/article/6875866

Download Persian Version:

https://daneshyari.com/article/6875866

Daneshyari.com