Repetitions in strings: algorithms and combinatorics
Maxime Crochemore, Lucian Ilie, Wojciech Rytter

To cite this version:

HAL Id: hal-00741884
https://hal-upec-upem.archives-ouvertes.fr/hal-00741884
Submitted on 13 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Repetitions in strings: algorithms and combinatorics

Maxime Crochemore\textsuperscript{a, \ast, 1}

\textsuperscript{a}Dept. of Computer Science, King’s College London, London WC2R 2LS, UK
and Université Paris-Est, France

Lucian Ilie\textsuperscript{b, 2}

\textsuperscript{b}Dept. of Computer Science, University of Western Ontario, N6A 5B7, London,
Ontario, Canada

Wojciech Rytter\textsuperscript{c, 3}

\textsuperscript{c}Institute of Informatics, Warsaw University, ul. Banacha 2, 02-097 Warszawa,
and Dept. of Math. and Informatics, Copernicus University, Torun, Poland

Abstract

The article is an overview of basic issues related to repetitions in strings, concentrating on algorithmic and combinatorial aspects. This area is important both from theoretical and practical point of view. Repetitions are highly periodic factors (substrings) in strings and are related to periodicities, regularities, and compression. The repetitive structure of strings leads to higher compression rates, and conversely, some compression techniques are at the core of fast algorithms for detecting repetitions. There are several types of repetitions in strings: squares, cubes, and maximal repetitions also called runs. For these repetitions, we distinguish between the factors (sometimes qualified as distinct) and their occurrences (also called positioned factors). The combinatorics of repetitions is a very intricate area, full of open problems. For example we know that the number of (distinct) primitively-rooted squares in a string of length $n$ is no more than $2n - \Theta(\log n)$, conjecture to be $n$, and that their number of occurrences can be $\Theta(n \log n)$. Similarly we know that there are at most $1.029 n$ and at least $0.944 n$ maximal repetitions and the conjecture is again that the exact bound is $n$. We know almost everything about the repetitions in Sturmian words, but despite the simplicity of these words, the results are nontrivial. One of the main motivations for writing this text is the development during the last couple of years of new techniques and results about repetitions. We report both the progress which has been achieved and which we expect to happen.

Keywords: Repetitions, squares, cubes, runs, consecutive repeats, tandem repeats, compression, factorisation, algorithms.
1 Introduction

Repetitions and periods in strings constitute one of the most fundamental areas of string combinatorics. They have been studied already in the papers of Axel Thue [46], considered as having founded stringology. While Thue was interested in finding long sequences with few repetitions, in recent times a lot of attention has been devoted to the algorithmic side of the problem.

Periods are ubiquitous in string and pattern matching algorithms. Knuth-Morris-Pratt string-matching algorithm uses the border table of the pattern, which is equivalent to using the periods of all its prefixes. Periods are implicitly computed when preprocessing the pattern in the as well famous Boyer-Moore algorithm (see [9,26]). The basic reason why periods show up in this question is that stuttering is likely to slow down any string-matching algorithm. The analysis of periods is even more important in constant-space optimal pattern matching algorithms because the only information on the patterns that is precomputed and stored is related to global and local periods of the pattern: perfect factorisation [23], critical factorisation [15], or sampling method [24]. By the way, the difficulties in extending string-matching techniques to image pattern matching methods are essentially due to different and more complex structures of 2D-periodicities.

Periodicities and repetitions in strings have been extensively studied and are important both in theory and practice. The strings of the type \(ww\) and \(www\), where \(w\) is a nonempty string, are called squares and cubes, respectively. They are well investigated objects in combinatorics of strings [33] and in string-matching with small memory [16].

Detecting repetitions in strings is an important element of several questions: pattern matching, text compression, and computational biology to quote a few. Pattern matching algorithms have to cope with repetitions to be efficient as these are likely to slow down the process; the large family of dictionary-based text compression methods (see [47]) use a weaker notion of repeats (like the software gzip); repetitions in genomes, called satellites or Simple Sequence Repeats, are intensively studied because, for example, some over-repeated short segments are related to genetic diseases [35]; some satellites are also used in forensic crime investigations.

* Corresponding author.

Email addresses: maxime.crochemore@kcl.ac.uk (Maxime Crochemore), ilie@csd.uwo.ca (Lucian Ilie), rytter@mimuw.edu.pl (Wojciech Rytter).

1 Research partially supported by CNRS.
2 Research partially supported by NSERC.
3 Research partially supported by the grant of the Polish Ministry of Science Higher Education N206 004 32/0806.
In this survey, we recall some of the most significant achievements in the area over the past three decades or so, as well as point out several central open questions. We focus on algorithms for finding repetitions and, as a key component, on counting various types of repetitions. The main results concern fast if not optimal algorithms for computing squares occurrences and runs, as well as combinatorial estimation on the number of the corresponding objects. Section 2 is devoted to properties of squares, Section 3 to that of runs, and finally the last two sections investigates repetitions in Fibonacci words and in Sturmian words.

2 Squares

Let $A$ be an alphabet of size $a$ and $A^*$ the set of all finite strings over $A$. We denote by $|w|$ the length of a string or word $w$, its $i$th letter by $w[i]$, and its factor (substring) $w[i]w[i+1]...w[j]$ by $w[i...j]$. Note that $w = w[1...|w|]$. We say that $w$ has period $p$ if $w[i] = w[i+p]$, for all $i$, $1 \leq i \leq |w| - p$. The period of $w$ is its smallest period and is denoted by $\text{period}(w)$. The ratio between the length and the period of $w$ is called the exponent of $w$. The string $u$ is said to be periodic if $\text{period}(u) \leq |u|/2$. A repetition in $w$ is an interval $[i...j] \subseteq [1...|w|]$ for which the associated factor $w[i...j]$ is periodic. It is an occurrence of a periodic string $w[i...j]$, sometimes called a positioned repetition in the literature. A string can contain many repetitions, see Figure 3.

In the following, we analyse squares in a string $x$ of length $n$.

The simplest but most investigated type of repetition is the square. A square is a string of the form $ww$, where $w$ is nonempty. Indeed, to avoid counting redundant elements, the root $w$ of the square is assumed to be primitive, that is, it is not itself the power of another string. This is equivalent to say that the exponent of $ww$ is 2. Note that the same square may appear several times in the same string and then we talk about square occurrences or equivalently positioned squares. As we shall see, counting distinct squares, i.e. squares that are distinct strings, or squares occurrences gives very different results.

2.1 Square occurrences

Initially people investigated mostly squares occurrences, but their number can be as high as $\Theta(n \log n)$ [6], hence algorithms computing all of them cannot run in linear time, due to the potential size of the output. Indeed the same result holds for any type of repetition having an integer exponent greater than 1 [8]. The optimal algorithms reporting all positioned squares or just a single
Fig. 1. Three squares that are prefixes of each other: \( u^2 < v^2 < w^2 \) with \( |w| \leq 2|u| \).

In this case none of the strings \( u, v, w \) is primitive.

square were designed in [6,1,37,7].

**Theorem 1 (Crochemore [6], Apostolico-Preparata [1], Main-Lorentz [37])**

There exists an \( O(n \log n) \) worst-case time algorithm for computing all the occurrences of primitively-rooted squares in a string of length \( n \).

Techniques used to design the algorithms are based on partitioning, suffix trees, and naming segments, respectively. A similar result has been obtained by Franek, Smyth, and Tang using suffix arrays [22]. The key component of the algorithm of Theorem 3 is the function described in the following lemma.

We say that an occurrence of a square \( ww \) in \( uv \) is centred in \( u \) (resp. \( v \)) if its position \( i \) satisfies \( i + |w| < |u| \) (resp. \( i + |w| \geq |u| \)).

**Lemma 2 (Main-Lorentz [37])** Given two square-free strings \( u \) and \( v \), reporting if \( uv \) contains a square centred in \( u \) can be done in worst-case time \( O(|u|) \).

Using suffix trees or suffix automata together with the function derived from the lemma, the following fact has been shown.

**Theorem 3 (Crochemore [7], Main-Lorentz [37])** Testing if a string of length \( n \) is square-free can be done in worst-case time \( O(n \log a) \), where \( a \) is the size of the alphabet of the string.

Another interesting result concerning periodicities is the following lemma and its fairly immediate corollary.

**Lemma 4 (Three Square Prefixes, Crochemore-Rytter [16])** If \( u, v, \) and \( w \) are three strings such that \( u \) is primitive, \( uu \) is a proper prefix of \( vv \), and \( vv \) is a proper prefix of \( ww \), then \( |u| + |v| \leq |w| \).

A couple of proofs of this important lemma, different from the original, were given. A short one appears in [33, page 281]. A very simple proof of a slightly weaker result, where \( 2|u| < |w| \), is given in [34, page 433]. We recall here the simplest such proof, due to Ilie [28], which yields two results: a weaker version of Lemma 4 \( (2|u| < |w| \) if any of the three strings is assumed primitive) and Corollary 2 below. For the former, assume \( 2|u| \geq |w| \); see Figure 1 where
$u_i = u$, $1 \leq i \leq 3$. Set $u^{-1}v = x^p$ with $x$ primitive. The overlap between $u_1$ and $u_2$ gives that $u = x'x'$, for a prefix $x'$ of $x$. Synchronisation of primitive powers of $x$ in $u_3$ with those in $u_2$ and the suffix $x^p$ of $v$ implies that $x'$ is empty and hence none of $u$, $v$, and $w$ is primitive, a contradiction.

A fairly immediate consequence of Lemma 4 is the next corollary.

**Corollary 1** Any nonempty string of length $n$ possesses less than $\log_\Phi n$ prefixes that are squares, where $\Phi$ is the golden mean $(1 + \sqrt{5})/2$.

### 2.2 Distinct squares

Unlike their number of occurrences discussed above, it is known that only $O(n)$ (distinct) squares can appear in a string of length $n$ [19].

In the configuration of Lemma 4, a second consequence is that $uu$ is a prefix of $w$. Therefore, a position in a string $x$ cannot be the largest (rightmost) position of more than two squares, which yields the next corollary. As mentioned earlier, a simple proof of it, bypassing Lemma 4 and due to Ilie [28], is illustrated by Figure 1. If three squares $u^2$, $v^2$, and $w^2$ start at the same position, then the shortest of those, $u^2$ appears again later. Indeed, this is obvious if $2|u| \leq |w|$. Otherwise, $u^2$ appears again $|x|$ positions later. Therefore, at most two squares can have their last (rightmost) occurrences starting at the same position. The claim follows.

**Corollary 2 (Fraenkel and Simpson [19])** Any string of length $n$ contains at most $2n$ (distinct) squares.

The structure of all squares and of unpositioned runs has been also computed within the running time $O(n \log a)$ in [36] and [27].

Based on numerical evidence, it has been conjectured that the number of (distinct) squares in a string of length $n$ is at most $n$. The best bound to date, $2n - \Theta(\log n)$, was given in [29].

Proving the conjecture is probably difficult due to the following example from [19]. Consider the family of strings $w_m = \bigcirc_{i=1}^m 0^{i+1}10^{i+1}1$, for all $m \geq 1$ ($\bigcirc$ denotes the concatenation). Then, the length of $w_m$ is $|w_m| = \frac{3}{2}m^2 + \frac{13}{2}m$ and the number of squares it contains is very close to it: $\frac{3}{2}m^2 + 4m - 3 + \frac{\text{odd}(m)}{2}$, that is, $|w_m| - o(|w_m|)$.

Figure 2 displays $w_4$ and the last occurrences of all its squares. It is interesting to note that, although the string has many squares, this bottom sequence contains very few 2’s.
Fig. 2. The string \( w_4 = 0^210^110^21^10^310^21^10^410^10^51 \) contains many squares. The rightmost occurrence of each square is displayed. The numbers in the bottom sequence give the number of squares whose rightmost occurrence starts at that position.

Fig. 3. The structure of runs in the string \( baababaababbabaababaab = b z^2(z^R)^2b \), where \( z = aabab \) and \( z^R = babaa \).

3 Runs

The concept of maximal repetitions, called runs in [30], has been introduced to represent all repetitions in a succinct manner. The crucial property of runs is that there are only \( O(n) \) many of them in a string of length \( n \) [32,42,10,40].

A run in a string \( w \) is an interval \([i \ldots j]\) such that both the associated string \( w[i \ldots j] \) has period \( p \leq (j - i + 1)/2 \), and the periodicity cannot be extended to the right nor to the left: \( w[i - 1] \neq w[x + p - 1] \) and \( w[j - p + 1] \neq w[j + 1] \) when these elements are defined. When the period \( p \) of a run is known, we call it a \( p \)-run. An example is displayed in Figure 3.

As a consequence of the algorithms and of the estimation on the number of squares, the most important result related to repetitions in strings can be formulated as follows.

**Theorem 5** (Kolpakov-Kucherov [32], Rytter [42], Crochemore-Ilie [10])

(i) All runs in a string of length \( n \) over an alphabet of size \( a \) can be computed in time \( O(n \log a) \).

(ii) The number of all runs is linear in the length of the string.

The point (ii) is very intricate and of purely combinatorial nature. The algorithm for (i) executes in time proportional to the number of runs (on a fixed-size alphabet) which, by (ii), is linear. Indeed, with an reasonable hypothesis on the alphabet, the running time of (i) can be reduced to \( O(n) \) as stated in Theorem 6 below.

Let \( \rho(n) \) be the maximal number of runs in a string of length \( n \). By item (ii) we have \( \rho(n) < cn \) for some constant \( c \). Based on the results in Table 1, Kolpakov and Kucherov [32] conjectured that \( c = 1 \) for binary alphabets. A stronger
Fig. 4. The $f$-factorisation of the string `baababaabababaababaab` of Figure 3 and the set of its internal runs; all other runs overlap factorisation points.

A conjecture was proposed in [21] where a family of strings is given with the number of runs equal to $\frac{3}{\sqrt{5}} = 0.927\ldots$ (Φ is the golden ratio), thus proving $c \geq 0.927\ldots$. The authors of [21] conjectured that this bound is optimal, but the best known lower bound for $c$ has been shown to be 0.944 more recently by Matsubara et al. [38]. Some reasons which might indicate that the optimal bound may be less than $n$ are discussed in Section 6.

| $n$  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| runs | 2  | 3  | 4  | 5  | 5  | 6  | 7  | 8  | 8  | 10 | 10 | 11 | 12 | 13 | 14 | 15 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

### 3.1 Computing runs

Next, we sketch shortly the basic components of the proof of the point (i) of Theorem 5. The main idea is to use, as for the previous Theorem 3, the $f$-factorisation of the input string (see [7]): a string $w$ is decomposed into factors $u_1, u_2, \ldots, u_k$, where $u_i$ is the longest segment which appears before its position in $w$, i.e. in $u_1 u_2 \ldots u_i A^{-1}$, possibly with overlapping the present occurrence of $u_i$; if the segment is empty $u_i$ is a single letter (see Figure 4).

The runs which fit in a single factor of the $f$-factorisation are called internal runs, other runs are called here overlapping runs. Figure 4 shows the $f$-factorisation and the internal runs of an example string.

There are three crucial facts:

- all overlapping runs can be computed in linear time,
- each internal run is a copy of an earlier overlapping run,
- the $f$-factorisation can be computed in linear time under some hypothesis on the alphabet of the string (see Theorem 6 below).

It follows easily from the definition of the $f$-factorisation that if a run overlaps two (consecutive) factors $u_{k-1}$ and $u_k$ then its size is at most twice the total size of these two factors.

---

4 This factorisation plays an important role in data compression algorithms and has many other applications. Its combinatorial properties have been investigated in [3] and [5]; see the latter for a number of open problems.
Fig. 5. If an overlapping run with period \( p \) starts in \( u_{k-1} \), ends in \( u_k \), and its part in \( u_{k-1} \) is of size at least \( p \), then it is easily detectable by computing continuations of the period \( p \) in the two directions, left and right.

Figure 5 shows the basic idea for computing runs that overlap \( u_{k-1} \) and \( u_k \) in time \( O(|u_{k-1}| + |u_k|) \). Using similar tables as in the Morris-Pratt algorithm (border and prefix tables, see [9,17]) we can test the continuation of a period \( p \), to the left and to the right. The corresponding tables can be constructed in linear time in a preprocessing phase.

After computing all overlapping runs the internal runs can be copied from their earlier occurrences by processing the string from left to right. Recall that, by (ii), there are only linearly many.

The above process is offline and computing all runs in linear time online, i.e. sequentially while reading the input string, is an open question. This might be of great interest when processing streams of data.

The \( f \)-factorisation of a string is commonly computed with the suffix tree or the suffix automaton of the string. When the alphabet of the string has a fixed size thanks to the efficient algorithms for building these data structures, the whole process can be carried on in linear time. Two recent algorithms, due to [12] and [4] (see also [13]), use the suffix array of the string to provide linear-time algorithms for integer alphabets. The hypothesis means that the alphabet of the string of length \( n \) is in the interval \([0, n^d]\), for some constant \( d \), which implies that letters can be sorted in linear time.

**Theorem 6 (Crochemore-Ilie [12], Chen-Puglisi-Smyth [4])** On an integer alphabet, the \( f \)-factorisation of a string and its runs can be computed in linear time.

### 3.2 Counting runs

The most intriguing question remains the asymptotically tight bound for the maximum number of runs \( \rho(n) \) in a string of length \( n \). The first proof (by painful induction) was quite difficult and has not produced any concrete constant coefficient in the \( O(n) \) notation. This subject has been studied in [21,20,44,45]. The exact number of runs has been considered for special types of strings (see Sections 4 and 5): Fibonacci strings and more generally Stur-
Fig. 6. This string of length 116 contains 99 runs (99/116 > 0.85). It has 27 1-runs, 26 2-runs, 27 3-runs, 6 5-runs, 5 8-runs, 6 13-runs, 1 21-run, and the whole string is a 55-run.

Fig. 7. Several runs starting at the same position. Their periods grow exponentially.

Fig. 8. Several runs with the same centre. Their periods grow only linearly. Above the string, strong local periodicities are shown.

mian strings [18,30,42]. The best-known lower bound of approximately 0.944n is from [38]. Figure 6 gives a sample of string containing many runs.

The first explicit upper bound for general strings was given by Rytter [42], that is, \( \rho(n) \leq 5n \), and improved in a structural and intricate manner in [43], \( \rho(n) \leq 3.44n \), by introducing a sparse-neighbour technique. Another improvement of the ideas of [42] was done in [40] where the bound 3.48n is obtained. The neighbours are runs for which both the distance between their starting positions is small and the difference between their periods is also proportionally small according to some fixed coefficient of proportionality. The occurrences of neighbours satisfy certain sparsity properties which imply the linear upper bound. Several variations for the definitions of neighbours and sparsity are possible. Considering runs having close centres (the beginning position of the second period) the bound has been lowered to 1.6n in [10,11], improved to 1.52n in [25], and further to 1.029n as a result of computations (see [14]).

It is interesting to note that the approach of [10,11] is somewhat counterintuitive. On the one hand, Corollary 1 states that there can be only logarithmically many runs starting at the same position and this is how they are counted in [42]; see Figure 7 for an example. On the other hand, there can be linearly many runs with the same centre, see the example in Figure 8, and still counting them this way in [10,11] yields a better bound. This is essentially due to the fact that many runs with the same centre implies strong local periodicities in the string, thus eliminating many other potential runs.

---

5 See the Web page http://www.shino.ecei.tohoku.ac.jp/runs/.
6 See the Web page http://www.csd.uwo.ca/~ilie/runs.html for the results of latest computations.
4 The structure of runs in Fibonacci words

The structure of runs is well understood for the class of Fibonacci words. Let us denote by \( f_n \) the \( n \)-th Fibonacci word (\( f_0 \) is the empty word, \( f_1 = b, f_2 = a, \) and \( f_n = f_{n-1}f_{n-2} \) for \( n > 2 \)), by \( F_n = |f_n| \) the \( n \)-th Fibonacci number, and by \( \mathcal{F}_\infty \) the infinite Fibonacci word.

Kolpakov and Kucherov (see [34, Chapter 8]) have shown the following two properties of runs occurring in Fibonacci words.

**Theorem 7 (Kolpakov-Kucherov [34])**

(i) There are exactly \( 2F_{n-2} - 3 \) runs in the \( n \)-th Fibonacci word \( f_n \).

(ii) The sum of exponents of runs in Fibonacci words is \( hF_n + o(n) \), where \( 1.922 \leq h \leq 1.926 \).

Property (i) shows in particular that the asymptotic ratio of runs in Fibonacci words is \( (\Phi \) is the golden mean \((1 + \sqrt{5})/2)\)

\[
\lim_{n \to \infty} \frac{2F_{n-2} - 3}{|f_n|} = \frac{2}{1 + \Phi} \approx 0.76.
\]

In fact we can compute the number of \( p \)-runs for a specific period \( p \). We say that the \( p \)-run \( w \) is short if \( |w| < 3p \), and long otherwise. Let also \( g_n \) be the \( n \)-th Fibonacci word \( f_n \) with the last two letters removed. We have:

**Lemma 8** Every run of \( \mathcal{F}_\infty \) with a period larger than two is of one of the two types: either a short \( \beta \)-run, i.e. of the form \( \beta_k = f_{k-1}^2g_{k-2} \), or a long \( \alpha \)-run, i.e. of the form \( \alpha_k = f_{k-2}^3g_{k-3} \), for some integer \( k \).

An immediate corollary follows from the structure of runs.

**Corollary 3 (Karhumäki [31])** There is no nonempty factor of the form \( w^4 \) in the infinite Fibonacci word \( \mathcal{F}_\infty \).

Indeed, it appears that maximal-exponent repetitions in \( \mathcal{F}_\infty \) correspond to long \( \alpha \)-runs. Their exponent, for a given \( k \), is then:

\[
\frac{|\alpha_k|}{\text{period}(\alpha_k)} = \frac{|f_{k-2}^3g_{k-3}|}{|f_{k-2}|} = \frac{3F_{k-2} + F_{k-3} - 2}{F_{k-2}} = \frac{F_{k-1} - 2}{F_{k-2}} + 2,
\]

whose limit is:

\[
\lim_{k \to \infty} \frac{F_{k-1} - 2}{F_{k-2}} + 2 = \Phi + 2.
\]

Therefore, if we define the repetition order of the (finite or infinite) string \( x \),
Fig. 9. The structure of runs in the Fibonacci word $F_{11}$. The arrows show endpoints of its prefixes which are the Fibonacci words $f_4, f_5, \ldots, f_{11}$. The string contains 65 runs. There are 21 = $F_8$ 1-runs (occurrences of $aa$), 12 = $F_7$ 2-runs, 13 = $F_7$ 3-runs, 8 = $F_6$ 5-runs, 5 = $F_5$ 8-runs, 3 = $F_4$ 13-runs, 2 = $F_3$ 21-runs, and 1 = $F_2$ 34-run.

denoted by $\text{rep}(x)$, as

$$\text{rep}(x) = \sup\{|w|/\text{period}(w) | w \text{ is a finite factor of } x\},$$

we get the stronger following corollary (see [33, Chapter 8]).

**Corollary 4 (Mignosi-Pirillo [39])** The repetition order of the infinite Fibonacci word $F_{\infty}$ is $\text{rep}(F_{\infty}) = \Phi + 2$.

Counting runs occurring in Fibonacci words by their period yields the next result illustrated by Figure 9. This provides an alternative proof for the number of all runs in Fibonacci words.

**Theorem 9 (Rytter [41])** The Fibonacci word $f_n$, $n \geq 6$, contains $F_{n-3}$ 1-runs, $F_{n-4}$ 1 2-runs, and $F_{n-k}$ $F_k$-runs for any $k$ satisfying $4 \leq k \leq n-2$.

## 5 Maximal repetitions in Sturmian words

The standard Sturmian words are generalization of Fibonacci words and similarly as Fibonacci words are described by recurrences. We denote by $S$ the class of standard Sturmian words.

The recurrence for a standard word is related to its so-called **directive sequence**: an integer sequence of the form $\gamma = (\gamma_0, \gamma_1, \ldots, \gamma_n)$, where $\gamma_0 \geq 0$ and $\gamma_i > 0$ for $0 < i \leq n$. The standard word corresponding to $\gamma$, denoted by $S(\gamma) = x_{n+1}$, is defined by the recurrence relations:

$$x_{-1} = b, \; x_0 = a, \; x_1 = x_0^{\gamma_0}x_{-1}, \; x_2 = x_1^{\gamma_1}x_0,$$

$$x_3 = x_2^{\gamma_2}x_1, \ldots, \; x_n = x_{n-1}^{\gamma_{n-1}}x_{n-2}, \; x_{n+1} = x_n^{\gamma_n}x_{n-1}$$

For example, a Fibonacci word is generated by a directive sequence of the form $(1, 1, \ldots, 1)$. 

The number $N = |x_{n+1}|$ is the (real) length of the word, while $n$ can be thought as its compressed size. It happens that $N$ can be exponential with respect to $n$, and computations on the word in time $O(n)$ are often rather nontrivial.

To state the next result, we introduce a zero-one function, called unary, for testing if its argument equals 1:

$$\text{if } x = 1 \text{ then } \text{unary}(x) = 1 \text{ else } \text{unary}(x) = 0.$$
maximal number of (distinct) squares occurring in a string and the maximal number of runs. We focused on algorithms for finding repetitions and, as a key component, on counting various types of repetitions.

Although the Kolpakov and Kucherov’s conjecture on the maximum number of runs in a string is still unsolved, from the practical point of view of the analysis of algorithms depending on this number, its very tight approximation is largely sufficient. A possible research track to attack the question is to study the compressibility of run-rich strings in addition to their combinatorial properties.

Aside from the above-mentioned open questions, we discuss here several other related problems.

**Distinct runs.** Inspired by the square problem, we may look at the strings associate with runs and count only the number of runs associated with different strings. Notice that the number of nonequivalent runs and that of squares do not seem to be obviously related to each other. The same run may contain several distinct squares (e.g., \textit{ababa} contains the squares \textit{abab} and \textit{baba}) but we can have also distinct runs corresponding to a single square (e.g., \textit{aa} and \textit{aaa} are distinct runs but only the square \textit{aa} is involved).

\[(2 + \varepsilon)^{+}\text{-repetitions.}\] A way to weaken the conjecture on the number of squares is to increase the exponent of the repetition. Given a non-negative \(\varepsilon\), one could count only the number of repetitions of exponent \(2 + \varepsilon\) or higher. We need first to make it precise what we are talking about. We count primitively-rooted repetitions of exponent at least \(2 + \varepsilon\) and having distinct roots. That is, \(x^{\alpha}\) and \(y^{\beta}\), \(x\) and \(y\) primitive, \(\alpha \geq 2 + \varepsilon\), \(\beta \geq 2 + \varepsilon\), are different if and only if \(x \neq y\).

This conjecture might be easier to prove. At least for \(2 + \varepsilon = 1 + \Phi\) (where \(\Phi\) is the golden ratio) we can prove it immediately. We count each square at the position where its rightmost occurrence starts and show that no two distinct squares can have the same rightmost starting position. Assume \(x^{1+\Phi}\) is a prefix of \(y^{1+\Phi}\) and denote \(|x| = p < q = |y|\). Then necessarily \(|x^{1+\Phi}| = (1 + \Phi)p > \Phi q = |y^{\Phi}|\) as otherwise \(x^{1+\Phi}\) would have another occurrence to the right. That means \(\Phi^{2}p = (1 + \Phi)p > \Phi q\), or \(\Phi p > q\). Therefore, the overlap between the two runs has the length \(|x^{1+\Phi}| = (1 + \Phi)p = p + \Phi p > p + q\). By Fine and Wilf’s lemma, this means \(x\) and \(y\) are powers of the same string and therefore not primitive, a contradiction.
(2 − ε)+-repetitions. This is similar to the previous problem except that now we consider repetitions of exponent 2 − ε or higher. Is the number of such maximal repetitions still linear? If this is false for any ε > 0, then 2 is the optimal threshold. Otherwise, the optimal threshold needs to be found.

References


