Two-Pattern Strings*

František Franěk[1], Jiandong Jiang[1,2], Weilin Lu[1,2], and W. F. Smyth[1,3]

[1] Algorithms Research Group, Department of Computing & Software
McMaster University, Hamilton, Ontario, Canada L8S 4K1
smyth@mcmaster.ca
www.cas.mcmaster.ca/cas/research/groups.shtml

[2] Toronto Laboratories, IBM Canada, 8200 Warden Avenue,
Markham, Ontario, Canada L6G 1C7

[3] School of Computing, Curtin University, GPO Box U-1987
Perth WA 6845, Australia

March 19, 2002

Abstract. This paper introduces a new class of strings on $\{a, b\}$, called ***two-pattern strings***, that constitute a substantial generalization of Sturmian strings while at the same time sharing many of their nice properties. In particular, we show that, in common with Sturmian strings, only time linear in the string length is required to recognize a two-pattern string as well as to compute all of its repetitions. We also show that two-pattern strings occur in some sense frequently in the class of all strings on $\{a, b\}$.

1 Introduction

In this paper we outline the results of an investigation of the properties of a new class of strings on $\{a, b\}$, derived by the successive action of a sequence of morphisms on the single letter a. All of the strings so determined are finite, and we deal with them from a computational point of view: initially, we are interested in efficient algorithms to recognize such strings and to compute the repetitions in them; then we go on to estimate their frequency of occurrence among all strings on $\{a, b\}$.

A previous paper [5] specified linear-time algorithms to recognize and compute repetitions in finite substrings of Sturmian strings; the class of strings discussed here significantly extends this work.

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ denote two distinct nonempty strings on $\{a, b\}$ such that $|\boldsymbol{p}| \leq \lambda$ and $|\boldsymbol{q}| \leq \lambda$, where λ is a finite integer called the ***scope***. We call

* Supported in part by grants from the Natural Sciences & Engineering Research Council of Canada.

$\boldsymbol{p}$ and $\boldsymbol{q}$ ***patterns of scope*** λ. For any pair of finite positive integers i and j such that $i < j$, consider the morphism σ that maps single letters into ***blocks***:

$$a \to \boldsymbol{p}^i \boldsymbol{q}, \quad b \to \boldsymbol{p}^j \boldsymbol{q}. \tag{1}$$

We call σ an ***expansion of scope*** λ and observe that it is specified by a 4-tuple $[\boldsymbol{p}, \boldsymbol{q}, i, j]$. Observe also that an expansion can be applied to any (finite or infinite) string on $\{a, b\}$ to yield an expanded string

$$\boldsymbol{y} = \sigma(\boldsymbol{x}).$$

Given any two morphisms σ_1 and σ_2, the composition $\sigma_1 \circ \sigma_2$ is therefore well defined: $\boldsymbol{z} = \sigma_1(\boldsymbol{y}) = \sigma_1\big(\sigma_2(\boldsymbol{x})\big) = (\sigma_1 \circ \sigma_2)(\boldsymbol{x})$.

Definition 1. *Suppose a positive integer λ and a finite sequence*

$$\sigma_1, \sigma_2, \ldots, \sigma_k$$

of expansions of scope λ are given, where

$$\sigma_r = [\boldsymbol{p_r}, \boldsymbol{q_r}, i_r, j_r]$$

for every $r = 1, 2, \ldots, k$. Then the string

$$\boldsymbol{x} = (\sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_k)(a)$$

is a ***complete two-pattern string of scope*** *λ if and only if every pair $(\boldsymbol{p_r}, \boldsymbol{q_r})$ of patterns is* ***suitable*** *(defined in Section 2).*

The definition of a suitable pair of patterns is deferred till Section 2 because it is necessarily somewhat technical. However, the main idea of a suitable pair is simple: $\boldsymbol{p}$ and $\boldsymbol{q}$ should be dissimilar enough that they can be efficiently distinguished from each other by an algorithm that recognizes complete two-pattern strings.

We can easily provide examples of complete two-pattern strings. If we suppose that $\lambda = 3$ and $\sigma_1 = [ab, ba, 2, 3]$, $\sigma_2 = [abb, aa, 1, 4]$, the following strings are all complete two-pattern strings of scope 3:

$$\begin{aligned}
\sigma_1(a) &= (ab)^2 ba; \\
(\sigma_1 \circ \sigma_1)(a) &= (ab)^2 ba (ab)^3 ba (ab)^2 ba (ab)^3 ba (ab)^3 ba (ab)^2 ba; \\
(\sigma_1 \circ \sigma_2)(a) &= (ab)^2 ba (ab)^3 ba (ab)^3 ba (ab)^2 ba (ab)^2 ba; \\
(\sigma_2 \circ \sigma_1)(a) &= (abb) aa (abb)^4 aa (abb) aa (abb)^4 aa (abb)^4 aa (abb) aa.
\end{aligned}$$

Observe further that when the scope $\lambda = 1$, the choice of $\boldsymbol{p}$ and $\boldsymbol{q}$ is restricted to

$$(\boldsymbol{p},\boldsymbol{q}) = (a,b) \text{ or } (\boldsymbol{p},\boldsymbol{q}) = (b,a). \tag{2}$$

If the further restriction is imposed that $j = i+1$, then all the strings generated by any finite sequence of expansions are finite substrings of Sturmian strings; in fact, in the terminology of [5], these strings are exactly the set of "block-complete" finite substrings of Sturmian strings. We note that every complete two-pattern string of scope λ is also a complete two-pattern string of scope $\lambda+1$; thus in particular every block-complete finite substring of a Sturmian string is a complete two-pattern string.

As noted above, our initial interest in complete two-pattern strings is computational, following similar studies of Fibonacci [7,4,8] and Sturmian [1,5] strings. We pose two sets of questions:

(Q1) What is the complexity of determining whether or not a given string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a fragment of a complete two-pattern string? Can an efficient algorithm be found to make this determination for every $\boldsymbol{x}$?

(Q2) Given a fragment $\boldsymbol{x}$ of a complete two-pattern string, can an algorithm be found that computes all the repetitions in $\boldsymbol{x}$ in linear time?

Since complete two-pattern strings constitute a much more general class of strings than block-complete finite substrings of Sturmian strings, the following questions also become of interest:

(Q3) What is the frequency of occurrence of fragments $\boldsymbol{x}$ of complete two-pattern strings among all strings on $\{a,b\}$ of length n? What is the asymptotic frequency of occurrence of complete two-pattern strings among all infinite strings on $\{a,b\}$?

In this paper we provide a partial answer to (Q1) by outlining an algorithm that in $\Theta(n)$ time determines whether or not a given string $\boldsymbol{x}[1..n]$ is complete two-pattern. Similar to the recognition algorithm in [5], this algorithm outputs the sequence of expansions (1) by which a is transformed into $\boldsymbol{x}$ — or more precisely, the sequence of ***reductions***

$$\boldsymbol{p}^i\boldsymbol{q} \rightarrow a, \quad \boldsymbol{p}^j\boldsymbol{q} \rightarrow b \tag{3}$$

by which $\boldsymbol{x}$ is reduced to a. This sequence provides a complete specification of $\boldsymbol{x}$. Since by (1) each reduction decreases string length by a factor that exceeds

$$i|\boldsymbol{p}|+|\boldsymbol{q}| \geq 2, \tag{4}$$

the recognition algorithm thus yields as a byproduct a potential data compression technique for complete two-pattern strings $\boldsymbol{x}$.

The reduction sequence is then used to provide partial answers to (Q2) and (Q3). Before going on to discuss these questions in more detail, we pause to provide an introduction and context for them, as well as an outline of the main results.

In dealing with (Q1), we need to cope with the possibility that at any stage of the reduction of $\boldsymbol{x}$, there may be more than one reduction satisfying (3): it then becomes possible that one of these reductions is a part of a sequence that reduces $\boldsymbol{x}$ to a, while another one is not. As long as this possibility exists, any recognition algorithm would be obliged to include provision for backtracking, leading possibly to an execution time exponential in the number of reductions. Our main result in this connection is to show however that backtracking is not required, and that therefore the algorithm that recognizes complete two-pattern strings requires only $\Theta(n)$ time.

Of course this does not yet fully solve (Q1). We conjecture that, just as for Sturmian strings [5], there exists a $\Theta(n)$-time algorithm to determine whether or not a string is a *fragment* of a complete two-pattern string of scope λ. If such an algorithm were found, it would greatly extend the class of strings that could be efficiently compressed using reductions, or whose repetitions could be efficiently computed.

The view may be taken that interest in (Q2) has been superseded by other work. It has recently become clear that, as a result of research extending over a period of a quarter-century, the repetitions in any string $\boldsymbol{x}[1..n]$ on an ***indexed*** alphabet — that is, an alphabet of size $\alpha \in O(n)$ that maps onto the integers $1..\alpha$ — can be computed in $\Theta(n)$ time. The main steps in this development are as follows:

- an algorithm to compute the suffix tree of $\boldsymbol{x}$ in $\Theta(n)$ time [3];
- an algorithm to compute the s-factorization of $\boldsymbol{x}$, given the suffix tree of $\boldsymbol{x}$, in $\Theta(n)$ time [9, 11];
- the identification of "maximal periodicities" or "runs" as a suitable encoding of repetitions in strings, and the computation of the leftmost occurrence of every distinct run in $\boldsymbol{x}$ in $\Theta(n)$ time, based on the s-factorization [10];
- the proof that the number of runs in any string is $O(n)$, and the extension of the algorithm [10] to compute all occurrences of every run in $\boldsymbol{x}$ in $\Theta(n)$ time, still based on the s-factorization [8].

Impressive as this intellectual edifice is, it nevertheless appears, at least in the context of strings on the alphabet $\{a, b\}$, to be rather indirect in its approach, perhaps involving more sophistication than is really required. Indeed, it is not clear that the $\Theta(n)$-time algorithm given in [3] is preferable in practice to classical $O(n \log n)$-time algorithms for suffix-tree construction. Further, the very long and technical proof that number of runs is linear in string length shows that a constant of proportionality exists, but provides no information about its size; at the same time, computer experiments described in [8] provide convincing evidence that the maximum number of runs in any string is at most n, and that this maximum occurs in strings on $\{a, b\}$! Thus, in a sense, the existing theory serves to remind us of how little, rather than how much, we know of periodicity in strings, perhaps especially those on $\{a, b\}$.

For (Q2) we adopt a more direct approach, an extension of the methodology used in [5] for Sturmian strings. Making use of the reduction sequence computed by the recognition algorithm, we show how to compute all the runs in complete two-pattern strings $\boldsymbol{x}[1..n]$ in $\Theta(n)$ time. Essentially, we show that if $\boldsymbol{y}$ is derived from $\boldsymbol{x}$ by a reduction (3), then the nontrivial runs in $\boldsymbol{x}$ can be computed directly from certain special configurations occurring in $\boldsymbol{y}$; thus, over the whole reduction sequence, the runs in $\boldsymbol{x}$ can be computed on a step-by-step basis, from one reduction to the next. It is the special configurations that are of interest here, since they provide insight into the way in which repetitions are formed.

Finally, we report on progress with (Q3), in estimating the frequency of occurrence of complete two-pattern strings among all strings on $\{a, b\}$. We claim that for λ sufficiently large with respect to n, complete two-pattern strings are dense in the set of all strings. We claim also that for some values of k and fixed λ, the number of distinct strings of length k (the ***complexity***) can exceed $2k$ — can in fact even be exponential in k.

Sections 2-4 deal with questions (Q1)-(Q3) respectively.

2 Recognizing Two-Pattern Strings in Linear Time

Before proceeding with our development, we need to provide a definition of the term "suitable pair" mentioned in Section 1:

Definition 2. *A string* $\boldsymbol{q}$ *is said to be* $\boldsymbol{p}$**-regular** *if and only if there are strings* $\mathbf{u} \neq \boldsymbol{\varepsilon}$, $\mathbf{v}$ *together with nonnegative integers* $n_1, \ldots, n_k$, $k \geq 1$, *and* r *such that*

- $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\mathbf{u}$*;*
- $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\mathbf{v}$*;*
- *there are at most two integer values* m_1 *and* m_2 *such that for each* $i \in 1..k$, $n_i = m_1$ *or* $n_i = m_2$, *i.e.* $|\{n_i : i \in 1..k\}| \leq 2$*;*

- $\boldsymbol{q} = (\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_1})(\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_2})\cdots(\boldsymbol{u}\boldsymbol{p}^r\boldsymbol{v}\boldsymbol{p}^{n_k})\boldsymbol{u}$;
- *if* $r = 0$, *then* $\boldsymbol{v} = \boldsymbol{\varepsilon}$ *(the empty string).*

The next definition formalizes the notion that the two strings $\boldsymbol{p}$ and $\boldsymbol{q}$ are fundamentally distinct and can be used as "building blocks" for complete two-pattern strings.

Definition 3. *An ordered pair of nonempty strings* $(\boldsymbol{p}, \boldsymbol{q})$ *is said to be a* **suitable pair of patterns** *if and only if*

- $\boldsymbol{p}$ *is* **primitive**, *i.e. has no nonempty border;*
- $\boldsymbol{p}$ *is neither a prefix nor a suffix of* $\boldsymbol{q}$*;*
- $\boldsymbol{q}$ *is neither a prefix nor a suffix of* $\boldsymbol{p}$*;*
- $\boldsymbol{q}$ *is not* $\boldsymbol{p}$*-regular.*

Using these definitions, we can now show how to reduce a nontrivial complete two-pattern string. Let

$$\begin{aligned}\boldsymbol{x} = {} & abbabaaabbababbababbababbabaaabbababbabaaabbababbababbababbaba \\ & aabbababbababbababbabaaabbababbababbababbabaaabbababbabaaabbab.\end{aligned}$$

Consider the reduction $\rho_1 = [a, bbab, 1, 3]$ and observe that according to Definition 3, $(a, bbab)$ is a suitable pair. Then applying the reduction

$$a(bbab) \rightarrow a, \quad a^3(bbab) \rightarrow b,$$

we find that

$$\boldsymbol{x_1} = \rho_1(\boldsymbol{x}) = abaaababaaabaaabaaabab.$$

Similarly for $\rho_2 = [a, b, 1, 3]$, we find

$$\boldsymbol{x_2} = \rho_2(\boldsymbol{x_1}) = ababbba,$$

while for $\rho_3 = [ab, bba, 2, 3]$,

$$\boldsymbol{x_3} = \rho_3(\boldsymbol{x_2}) = a.$$

Thus $\boldsymbol{x} = \boldsymbol{x}[1..124]$ is completely described by the three-term reduction sequence

$$[a, bbab, 1, 3], \;\; [a, b, 1, 3], \;\; [ab, bba, 2, 3],$$

and so is a complete two-pattern string of scope 4.

In the context of possible applications to data compression, it is worth remarking that in general the number of terms in a reduction sequence is by (4) at most $\lceil \log_2 n \rceil$ and may be much smaller. In our example, $n = 124$ and sequence length $3 = \log_{4.39} 124$.

We now introduce formally an idea mentioned in the introduction: a canonical reduction that is identified by patterns that are somehow "shortest":

Definition 4. *A reduction* $\rho = [\boldsymbol{p}, \boldsymbol{q}, i, j]$ *of a binary string* $\boldsymbol{x}$ *using patterns of scope* λ *is* **$\boldsymbol{\lambda}$-canonical** *if and only if for every reduction* $\rho_1 = [\boldsymbol{p_1}, \boldsymbol{q_1}, i_1, j_1]$ *of* $\boldsymbol{x}$ *using patterns of scope* λ*:*

(a) *either* $|\boldsymbol{p}| < |\boldsymbol{p_1}|$, *or* $|\boldsymbol{p}| = |\boldsymbol{p_1}|$ *and* $|\boldsymbol{q}| < |\boldsymbol{q_1}|$, *or* $|\boldsymbol{p}| = |\boldsymbol{p_1}|$ *and* $|\boldsymbol{q}| = |\boldsymbol{q_1}|$.

(b) $\boldsymbol{x} = \boldsymbol{p_1}^{i_1}\boldsymbol{q_1}$ *implies* $\boldsymbol{x} = \boldsymbol{p}^i\boldsymbol{q}$.

It is then possible to prove that it suffices to reduce $\boldsymbol{x}$ using a sequence of canonical reductions:

Theorem 1. $\boldsymbol{x}$ *is a complete two-pattern string of scope* λ *if and only if there is a sequence of* λ*-canonical reductions* $\{\rho_1, \rho_2, \cdots, \rho_n\}$ *reducing* $\boldsymbol{x}$ *to a string a.* □

We omit the fairly predictable details of the algorithm REC that is based on this theorem. We state however the main result:

Theorem 2. *For any* $\lambda \geq 1$*, the recognition algorithm REC determines in* $O(2\lambda^8|\boldsymbol{x}|)$ *steps whether or not* $\boldsymbol{x}$ *is a complete two-pattern string of scope* λ*, and if so, the algorithm outputs the* λ*-canonical reduction sequence of* $\boldsymbol{x}$*.* □

3 Computing the Repetitions in Linear Time

We describe here the main ideas that permit the repetitions in a complete two-pattern string to be computed in linear time. Just as for Sturmian strings [5], it turns out that the repetitions that occur in an expansion $\boldsymbol{y} = \sigma(\boldsymbol{x})$ of a two-pattern string $\boldsymbol{x}$ are formed as a result of the application of σ to certain well-defined configurations in $\boldsymbol{x}$. Thus, with the help of the expansion (reduction) sequence that determines a complete two-pattern string, it is possible to track and output the repetitions as they are formed by each expansion in the sequence. As mentioned in the introduction, a crucial factor that ensures the efficiency of this process is the encoding of the repetitions as runs.

Definition 5. *A **repetition** in $\boldsymbol{x} = \boldsymbol{x}[1..n]$ [2] is a triple (i,p,r) of positive integers, where $i < n$, $r > 1$,*

$$\boldsymbol{x}[i..i{+}rp{-}1] = \boldsymbol{x}[i..i{+}p{-}1]^r,$$

*and $\boldsymbol{x}[i+rp..i{+}(r{+}1)p{-}1] \neq \boldsymbol{x}[i..i{+}p{-}1]$. The **period** of the repetition is p and its **generator** is $\boldsymbol{x}[i..i{+}p{-}1]$.*

Crochemore [2] showed that Fibonacci strings, a special case of two-pattern strings, contain $\Omega(n \log n)$ repetitions. To avoid $\Omega(n \log n)$ processing just for output, we therefore introduce:

Definition 6. *A **run** in $\boldsymbol{x}[1..n]$ [10] is a 4-tuple (i,p,r,t) where*

$$(i,p,r),\ (i{+}1,p,r),\ \ldots,\ (i{+}t,p,r)$$

*are all repetitions, while $(i{-}1,p,r)$ and $(i{+}t{+}1,p,r)$ are not. The **period** and **generator** are defined as for (i,p,r).*

It is easy to prove that for any constant κ, all the runs in $\boldsymbol{x}$ whose period $p \leq \kappa$ can be output in at most $c_\kappa n$ steps, where c_κ is a constant whose value depends only on κ. In particular, this result is true for the choice $\kappa = 3\lambda$. For $p > 3\lambda$, we require the following lemma, the main result of this section:

Lemma 1. *For $|\boldsymbol{p}| \leq \lambda$, $|\boldsymbol{q}| \leq \lambda$, let $\sigma = [\boldsymbol{p}, \boldsymbol{q}, i, j]$ be an expansion of $\boldsymbol{x}$, so that $\boldsymbol{y} = \sigma(\boldsymbol{x})$. Then for every run R in $\boldsymbol{y}$ whose period $p > 3\lambda$, one of the following holds:*

- *R is an expansion under σ of a run in $\boldsymbol{x}$,*
- *R is determined by a square $\boldsymbol{u}^2$ in $\boldsymbol{x}$ that is derived from a substring of $\boldsymbol{x}$ of one of the following forms:*

$$aa, ab, ba, bb, a\boldsymbol{v}b\boldsymbol{v}a, b\boldsymbol{v}a\boldsymbol{v}b, b\boldsymbol{v}aa\boldsymbol{v}b, a\boldsymbol{v}b\boldsymbol{v}, b\boldsymbol{v}a\boldsymbol{v},$$

for any nonempty substring $\boldsymbol{v}$. □

The details of "deriving" $\boldsymbol{u}^2$ and of using it to "determine" R are lengthy and complicated, to be found at web site

`http://www.cas.mcmaster.ca/~franek/`

The analysis found there enables us to claim that

Theorem 3. *There exists an algorithm RUN such that for every integer $\lambda \geq 1$, RUN computes all the runs in every complete two-pattern string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ of scope λ, based on the reduction sequence of $\boldsymbol{x}$, in at most $c_\lambda n$ steps, where c_λ is a constant whose value depends only on λ.* □

4 Frequency of Two-Pattern Strings

In this section we first present results showing that infinite two-pattern strings (that is, two-pattern strings formed from an infinite sequence of expansions) have complexity at least $k+1$, while for some values of k the complexity can even be exponential in k. Since Sturmian strings have complexity $k+1$, we can accordingly claim that two-pattern strings are in some sense more frequent among all strings on $\{a, b\}$ than Sturmian strings are. Nevertheless, for fixed λ the relative frequency of two-pattern strings approaches zero as string length approaches infinity.

Another point of view is also of interest. We find that if we consider only those values of n that are close to λ, then two-pattern strings occur frequently among all strings of length n. In other words, for sufficiently large λ, a large proportion of strings on $\{a, b\}$ turn out to be two-pattern strings.

We begin by recalling the notation $\mathrm{LCP}(\boldsymbol{u}, \boldsymbol{v})$ and $\mathrm{LCS}(\boldsymbol{u}, \boldsymbol{v})$ for arbitrary strings $\boldsymbol{u}$ and $\boldsymbol{v}$: longest common prefix and longest common suffix, respectively. It is then convenient to define, for any integer $m \geq 0$,

$$\Delta_m = \Delta_m(\boldsymbol{u}, \boldsymbol{v}) = m|\boldsymbol{p}| + |\mathrm{LCP}(\boldsymbol{u}, \boldsymbol{v})| + |\mathrm{LCS}(\boldsymbol{u}, \boldsymbol{v})|.$$

Using this notation, we state:

Theorem 4. *Let $\boldsymbol{x}$ be an infinite two-pattern string of scope λ reducible by $[\boldsymbol{p}, \boldsymbol{q}, i, j]$. Then the complexity $C_k = C_k(\boldsymbol{x})$ satisfies*

(a) $C_k \geq k+1$ *when* $|\boldsymbol{p}| \leq k \leq \Delta_i$;
(b) $C_k \geq 2k-\Delta_i$ *when* $\Delta_i+1 \leq k \leq \Delta_{j-1}+1$;
(c) $C_k \geq k+1+(j-i-1)|\boldsymbol{p}|$ *when* $k \geq \Delta_{j-1}+2$;

where $\Delta_m = \Delta_m(\boldsymbol{p}, \boldsymbol{q})$. □

This result provides lower bounds on the complexity C_k that are in fact sharp. We have also established upper bounds on C_k that are however not sharp. To show that the complexity can for some values of k be much larger than $k+1$, consider an infinite two-pattern string $\boldsymbol{x}$ with substring $\boldsymbol{pqp}$, where $\boldsymbol{p} = aaaabbbb$ and $\boldsymbol{q} = aababbab$ are a suitable pair of patterns. It is easy to check that in the substring $\boldsymbol{pqp}$ there are 2^4 substrings of length 4 — for $k = 4$ the complexity is exponential in k.

In order to state our final results, we introduce the constant [6]

$$\phi = \lim_{n \to \infty} P(n)/2^n \approx 0.26778684,$$

where $P(n)$ is the number of primitive strings (with no nonempty border) of length n on $\{a,b\}$. The frequency of occurrence of two-pattern strings for λ large with respect to n can then be estimated in terms of ϕ:

Theorem 5. *For $n \geq 2$, let $f(n)$ denote the frequency of occurrence among all strings of length n of two-pattern strings $\boldsymbol{x}[1..n] = \boldsymbol{pq}$, where $\boldsymbol{p}, \boldsymbol{q}$ is a suitable pair of scope $\lambda \geq \lceil n/2 \rceil$. Then $f(n) \geq \phi/2$.* □

Theorem 6. *For $n \geq 4$, let $f(n)$ denote the frequency of occurrence among all strings of length n of two-pattern strings $\boldsymbol{x}[1..n] = \boldsymbol{p}^{\boldsymbol{i}}\boldsymbol{q}$, where $i \geq 1$ and $\boldsymbol{p}, \boldsymbol{q}$ is a suitable pair of scope $\lambda \geq n-2$. Then $f(n) \geq 15\phi/16$.* □

References

1. M. Boshernitzan & Aviezri S. Fraenkel, **A linear algorithm for nonhomogeneous spectra of numbers**, *J. Algorithms 5* (1984) 187-198.
2. Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *IPL 12-5* (1981) 244-250.
3. Martin Farach, **Optimal suffix tree construction with large alphabets**, *Proc. 38th Annual IEEE Symp. FOCS* (1997) 137-143.
4. Aviezri S. Fraenkel & R. Jamie Simpson, **The exact number of squares in Fibonacci words**, *TCS 218-1* (1999) 83-94.
5. František Franěk, Ayşe Karaman & W. F. Smyth, **Repetitions in Sturmian strings**, *TCS 249-2* (2000) 289-303.
6. Leo J. Guibas & Andrew M. Odlyzko, **Periods in strings**, *J. Combinatorial Theory, Series A 30* (1981) 19-42.
7. Costas S. Iliopoulos, Dennis Moore & W. F. Smyth, **A characterization of the squares in a Fibonacci string**, *TCS 172* (1997) 281-291.
8. Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algorithms 1* (2000) 159-186.
9. Abraham Lempel & Jacob Ziv, **On the complexity of finite sequences**, *IEEE Trans. Information Theory 22* (1976) 75-81.
10. Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145-153.
11. Jacob Ziv & Abraham Lempel, **A universal algorithm for sequential data compression**, *IEEE Trans. Information Theory 23* (1977) 337-343.