Elsevier Editorial System(tm) for Theoretical Computer Science
Manuscript Draft

Manuscript Number: TCS-D-08-00419

Title: Compressed string matching in standard Sturmian words

Article Type: Special Issue: BIJK 2009 (Rozenberg)

Section/Category: A - Algorithms, automata, complexity and games

Keywords: string matching; Sturmian words; linear time; algorithm;
don't care symbols

Corresponding Author: Dr. W. Rytter,

Corresponding Author's Institution: Warsaw University

First Author: Pawel Baturo

Order of Authors: Pawel Baturo; W. Rytter

Compressed String Matching in Standard Sturmian Words [1]

Pawel Baturo [a] Wojciech Rytter [b,a]

[a] *Department of Mathematics and Informatics, Copernicus University, Torun*
[b] *Inst. of Informatics, Warsaw University, Warsaw, Poland*

Abstract

We present a simple algorithm which for an explicitely given input string v (a *pattern*) and a standard Sturmian word w described by the recurrences of size n computes in time $O(|v|+n)$ the set of all occurrences of v in w as a single arithmetic progression (modulo the length of w). For cyclic shifts of standard words the set of all occurrences does not need to be a single arithmetic progression, in this case our algorithm produces a linear set of arithmetic progressions. The algorithm can be extended to the case when some letters of the pattern are replaced by *don't care* symbols. It is an example of fast computations for the input given in a compressed form, in our special case the length N of the standard Sturmian word w is usually exponential with respect the size n of the input.

Key words: Sturmian words, string-matching, linear time, lexicographic numeration property

1 Introduction

The *standard Sturmian words* (*standard words*, in short) are generalization of Fibonacci words and have many interesting combinatorial properties, see for example [11,9,5,6,15,3,4,12,13,1,10]. In particular in [9] it has been introduced the *lexicographic shift property*: the lexicographic order of cyclic shifts corresponds to an arithmetic progression of shifts (modulo length of the word). We introduce and investigate a similar property of Sturmian words w:

[1] Supported by the grant KBN N206 004 32/0806 for the second author.
A preliminary version of the paper has been presented at LATA 2007

Occurrence shift property: the set of occurrences of any subword v in w is a **single** arithmetic progression (modulo the length of the word) and can be computed in linear time;

We fix the alphabet $\Sigma = \{a, b\}$. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. Standard words are described by the the recurrences corresponding to so called **directive sequences**

$$\gamma \ = \ (\gamma_0, \gamma_1, ..., \gamma_{n-1}),$$

where $\gamma_0 \geq 0$, $\gamma_i > 0$ for $0 < i < n$. The **standard word** corresponding to γ, denoted by $Word(\gamma) = x_n$, is defined:

$$x_{-1} = b, x_0 = a, \ x_1 = x_0^{\gamma_0} x_{-1}, \ x_2 = x_1^{\gamma_1} x_0, \ldots, x_n = x_{n-1}^{\gamma_{n-1}} x_{n-2}. \qquad (1)$$

We consider here standard words starting with the letter a, hence assume $\gamma_0 > 0$. The case $\gamma_0 = 0$ can be considered similarly. For even $n > 0$ a word x_n has suffix ba, and for odd $n > 1$ it has suffix ab. The number $N = |x_n|$ is the (real) size, while n can be thought as the compressed size.

Example. Consider the standard word

$$Word(1, 2, 1, 1, 1) = ababaababababaababaab$$

written below (together with numbering of positions) and corresponding sequence of the recurrences:

a	b	a	b	a	a	b	a	b	a	b	a	a	b	a	b	a	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$$x_{-1} = b; \ x_0 = a, \ x_1 = x_0^1 x_{-1}, \ x_2 = x_1^2 x_0,$$

$$x_3 = x_2^1 x_1, \ x_4 = x_3^1 x_2, \ x_5 = x_4^1 x_3.$$

A grammar-based compression is given be a context-free grammar generating a single string. Our recurrences can be treated as a special grammar-based compression. A very simple representation of standard Sturmian words has some special algorithmic consequences. There is an algorithm for compressed pattern-matching in standard words working in linear time. This is much better than the general compressed pattern-matching algorithms for texts given by a grammar-based compressed representation, see [8,14,7].

Denote by $Occ(y, x)$ the set of positions which start occurrences of a pattern y in text x. Assume the positions of the table and indexes of symbols in words

start with 0. We say that a word w has **occurrence shift property** iff for any word y the set $Occ(y, w)$ can be ordered as a sequence which is an arithmetic progression modulo the length of w.

Denote: $x \oplus y = (x+y) \bmod N$, where N is the length of the considered Sturmian word w. Similarly define the *circular* subtraction $\ominus$ and multiplication $\otimes$.

Example
(a) For $w = ababaababababaababaab = Word(1, 2, 1, 1, 1)$ we have:

$$Occ(abab,\ w) = \{\ 0, 5, 7, 12\ \}.$$

The set $\{\ 0, 5, 7, 12\ \}$ has *constant shift property* with respect to $N = 19$, since it can be ordered as an arithmetic progression:

$$7 \xrightarrow{\oplus\ 12} 0 \xrightarrow{\oplus\ 12} 12 \xrightarrow{\oplus\ 12} 5.$$

(b) For $w = ababaababababaababa = Word(1, 2, 1, 2)$ we have:

$$Occ(abab,\ w) = \{\ 14, 7, 0, 12, 5\ \}.$$

Throughout this paper we use in examples the $Word(1, 2, 1, 2)$ for the case when directive sequence is even and the $Word(1, 2, 1, 1, 1)$ for the case when directive sequence is odd

A *compressed representation* of an arithmetic progression consists of 3 numbers: the starting element, the difference, and the last element. We show:

for given sequence γ and an explicitly given word x we can compute a compressed representation of $Occ(x, Word(\gamma))$ in time $O(|x| + |\gamma|)$.

2 Compressed string matching in Sturmian words

There is a very useful *circular* interpretation of standard words, in which only the numbers of letters matter. Let $|w|_a$ denote the number of occurrences of the symbol a in the standard word w of size N, $q = |w|_a$, $p = |w|_b$. Define

$$I_a = [0..q-1],\ I_b = [q..N-1]$$

We recall an equivalent definition of standard words (see [11] for details) by defining symbols at positions $0 \le i < N$ in the following way, see Figure 1. For $0 \le j < N$, define:

$$R_{p,j}[i] = a \;\Leftrightarrow\; j \oplus_N i \cdot p \;\in I_a,$$

$$R_{p,j}[i] = b \;\Leftrightarrow\; j \oplus_N i \cdot p \;\in I_b$$

In other words we generate the word $R_{p,j}$ in the following way. Draw consecutively on the circle q symbols a and after them p symbols b. Then start from the j-the letter a (counting from 0) and go around the circle clockwise with the step p. Each time we meet a we output a, otherwise we output b. After outputting N letters we stop. $R_{p,j}$ is the generated word.

Example. In Figure 1 we have: $x = R_{8,7} = ababaababababaababaab$, $N = 19$. This word is *mechanically* generated starting at (inner) position 7 (marked by an arrow) in Figure 1 and moving around with the step 8. The small squares correspond to symbols a, the circles correspond to b's. If we start at other point then we generate a cyclic shift of x.

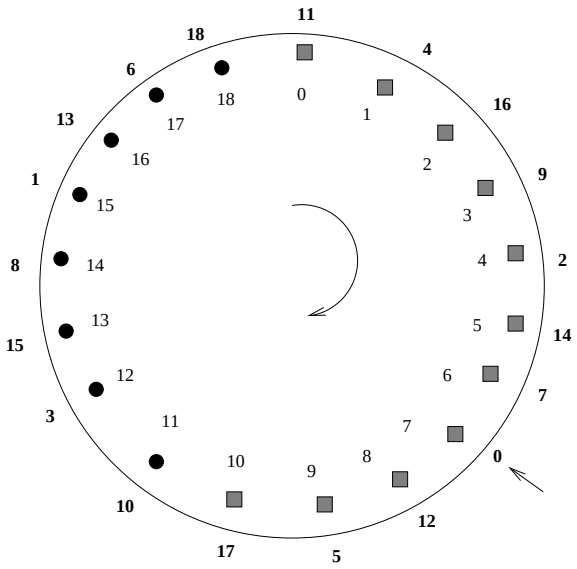

Fig. 1. The mechanical generation of the word $x = R_{8,7}$. The *jump* is $p = 8$ and we start at $j = 7$. The outer numbering corresponds to the table ROT, it is an arithmetic progression with the difference $r = p^{-1} = 12$ ($8 \times 12 = 96 = 5 \times 19 + 1$).

Let ROT be the table of cyclic shifts of a given word x which are listed in lexicographic order, the cyclic shift is identified with the size of the shift. For a given word w of length N define $shift(i, w) = w[i..N-1]w[0..i-1]$, where $w[0..-1]$ is the empty word. The following facts have been shown in [11,9].

Lemma 1
(a) *If x is a standard word starting with a then $x = R_{p,p}$ or $x = R_{p,p-1}$, in the former case x ends with ba, otherwise it ends with ab.*

Let $r = p^{-1}$ (modulo N), then
(b) *The values of the lexicographically sorted rotations for the word $R_{p,p-1}$ are given by the formula:*

$$ROT[i] \;=\; (N-1) \oplus_N (i+1) \cdot r.$$

(c) *The values of the lexicographically sorted rotations for the word $R_{p,p}$ are given by the formula:*

$$ROT[i] \;=\; (N-1) \oplus_N i \cdot r.$$

There are two numberings on the circle, the inner numbering corresponds to positions in the table $ROT[*]$, the outer numbering to the values of $ROT[*]$. In our example the outer numbering is given by formula $11 + 12 \cdot i$, since $8^{-1} = 12$ (modulo 19).

Theorem 1 [String matching in Sturmian Words]
For a standard word w given by a sequence $\gamma = (\gamma_0, \ldots, \gamma_{n-1})$ and a pattern x of the length m we can compute $Occ(x, w)$ in $O(n+m)$ time. The output is produced as a single arithmetic progression given by three numbers (the starting and ending positions, and the difference).

Proof
The table ROT has a special property responsible for the fact that the output is a single arithmetic progression. The following claim is a direct consequence of Lemma 1.

Claim 1. If w ends with the letter b then $ROT[N-1] = N-1$, otherwise $ROT[0] = N-1$. In other words the cyclic shifts starting at position $N-1$ of w is lexicographically the least or the last.

For $0 \le s \le t \le N-1$ an interval $\mathcal{I} = [s, t]$ is the set of positions $\{s, s+1, \ldots, t\}$, where s, t are the starting and terminating positions. We extend it to the case $s > t$, in this situation define $\mathcal{I}$ as the set $\{s, s+1, \ldots, N-1, 0, \ldots, t\}$.s s s s

For an interval $\mathcal{I} = [s, t]$ denote

$$SHIFT(\mathcal{I}) \;=\; [s \oplus p,\; t \oplus p].$$

Recall that

$$I_a \;=\; [0..q-1],\; I_b \;=\; [q..N-1].$$

ALGORITHM *String-Matching*(x, w) ;

Input: $x = x_0 x_1 \ldots x_{m-1}$, $w = Word(\gamma_0, \gamma_1, \ldots \gamma_{n-1})$,

$N := |w|$; $p := |w|_b$; $q := |w|_a$;

if n is odd **then** $\delta := N - 1$ **else** $\delta := 0$;

if $x_0 = a$ **then** $\mathcal{I} := \mathcal{I}_a$ **else** $\mathcal{I} := \mathcal{I}_b$;

Forward Phase:

for $j = 1$ **to** $m - 1$ **do**

Invariant: if $\delta \in \mathcal{I}$ then $\delta = s$ or $\delta = t$;

(1) $\mathcal{I} := \mathcal{I} - \{\delta\}$;

(2) $\mathcal{I} := SHIFT(\mathcal{I})$;

(3) **if** $x_j = a$ **then** $\mathcal{I} := \mathcal{I} \cap \mathcal{I}_a$ **else** $\mathcal{I} := \mathcal{I} \cap \mathcal{I}_b$;

Backward Phase:

let $\mathcal{I} = [s, t]$; $s := ROT[s] - m + 1$; $t := ROT[t] - m + 1$;

return

the arithmetic progression starting in s and ending in t, with difference $r = p^{-1}$ (modulo N) ;

If $\mathcal{I} = [s, t]$ is an interval then:

$$ROT(\mathcal{I}) \;=\; \{\; ROT[s], ROT[s+1], \ldots, ROT[t] \;\}.$$

Observe that it is a singe arithmetic progression with the difference p^{-1}.

An *end-occurrence* of x in w is a position i such that $w[i - m + 1, \ldots, i] = x$. Observe that a set of end-occurrences is an arithmetic progression if the set of the (starting) occurrences is. Denote by $End - Occ(y, w)$ the set of end-occurrences of y in w.

The next claim follows directly from the circular interpretation of the word w implied by Lemma 1. Each time we extend the prefix of x we have to remove the position $N-1$, since the next position would go outside the word w. Hence we remove the position δ from the interval $\mathcal{I}$ since $ROT[\delta] = N - 1$.

Correctness of the values of δ chosen by the algorithm follows from Claim 1.

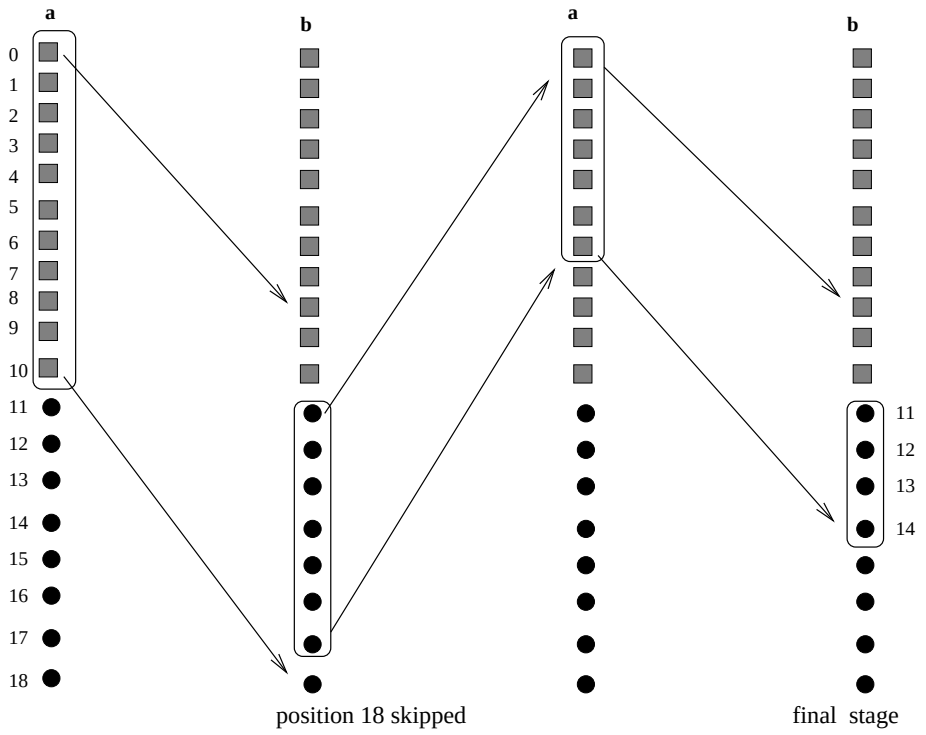

Fig. 2. Computation of $Occ(abab, x)$, the forward phase.

Hence we have:

Claim 2.

(a) If $x_0 = a$ then $End\text{-}Occ(x_0, w) = ROT(\mathcal{I}_a)$ else $End\text{-}Occ(x_0, w) = ROT(\mathcal{I}_b)$.

(b) Assume $ROT(\mathcal{I}) = End\text{-}Occ(x_0 x_1 \ldots x_{j-1}, w)$.

If $x_j = a$ then

$$End\text{-}Occ(x_0 x_1 \ldots x_{j-1} x_j, w) \;=\; ROT(SHIFT((\mathcal{I} - \{\delta\}) \cap \mathcal{I}_a))$$

else

$$End\text{-}Occ(x_0 x_1 \ldots x_{j-1} x_j, w) \;=\; ROT(SHIFT((\mathcal{I} - \{\delta\}) \cap \mathcal{I}_b))$$

Claim 2 implies that the returned set is the set of all occurrences. We have still to show the following fact.

Claim 3. The returned set of occurrences is a single arithmetic progression.

Proof (of the claim)

The point δ is at the beginning of $\mathcal{I}_a$ or at the end of $\mathcal{I}_b$ due to Claim 1. Hence in each iteration when we remove δ it will be the first or the last in the current arithmetic progression. Consequently the final set of returned values is a single arithmetic progression. □

We discuss now the complexity issues. We can compute the size of the word $Word(\gamma_0, \ldots, \gamma_{n-1})$ in $O(n)$ time, as well as the number of letters in w. Once we have the numbers N, p, q the final interval $\mathcal{I}$ can be computed in $O(m)$ time. The remaining parameter is the number $r = p^{-1}$ (modulo) N). It can be computed in the following way due to [9]:

- if n is odd then $r = |x_{n-1}|$;
- otherwise $r = N - |x_{n-1}|$.

Hence this parameter can be also computed in linear time. This completes the proof of the theorem. □

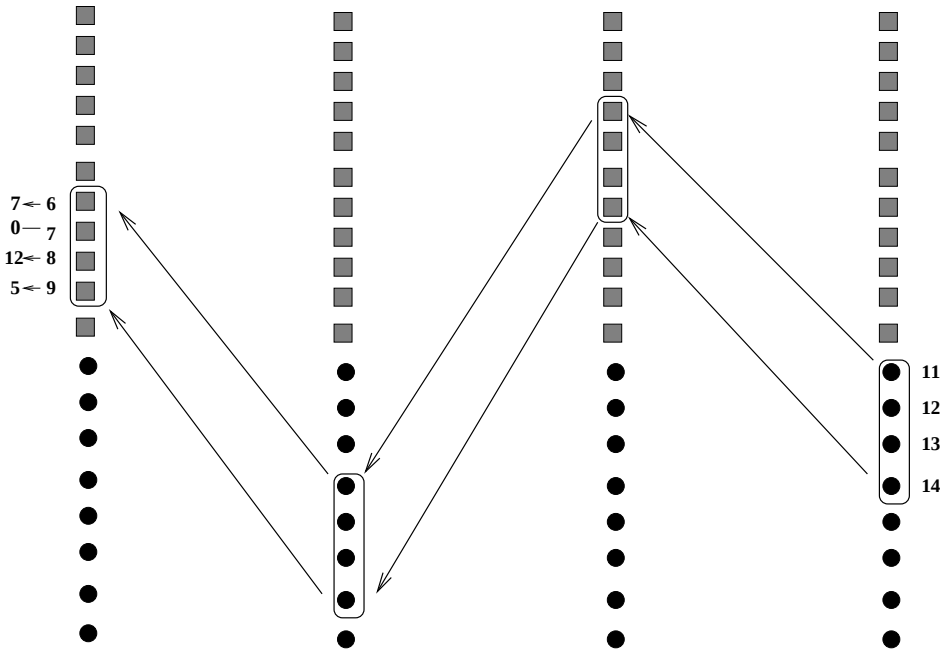

Fig. 3. The computation of $Occ(abab, x)$, the backward phase. We have: $Occ(abab, x) = \{\ ROT[6], ROT[7], ROT[8], ROT[9]\ \} = \{7, 0, 12, 5\}$. It is a part of the arithmetic progression describing the sequence $ROT[*]$.

We explain now the main ideas of the proof using illustrations in Figure 2. In the forward phase, we move the interval [0..10] by shifting it by 8 (modulo 19), each time we take only positions which include the required symbol. The sequence of symbols is: a, b, a, b.

If we encounter the last position we do not take it, unless it is the final stage. Observe that we omit the number $\delta = 18$ in the first move.

Remark. For cyclic shifts of standard words the occurrence shift property is no longer valid. For example let $x' = x[9..18]x[0..8] = abaababaababababaabab$, then $Occ(ba, x)$ is not a single arithmetic progression.

Although cyclic shifts of standard words do not necessarily have Occurrence Shift Property the set of occurrences in such words can be also described in a compressed way using only few arithmetic progressions.

Theorem 2 [Cyclic shifts and don't care symbols]
Assume w is a cyclic shift of a standard word given by a directive sequence of size n and some letters (possibly none) of the pattern x are replaced by a don't care symbol. Then we can compute compressed representation of $Occ(x, w)$ in time $O(m + n)$, where $m = |x|$. This compressed representation consists of at most m arithmetic progressions.

Proof We can apply essentially the same algorithm *String-Matching*(x, w) to extend Theorem 1. There are however several technical differences.

When we process the next symbol of the pattern equal to the *don't care* symbol then we do not intersect the current set $\mathcal{I}$ with $\mathcal{I}_a$ nor $\mathcal{I}_b$.

Instead of a single interval representing the current set of positions $\mathcal{I}$ we need to keep (at most linear) set of subintervals. The current set $\mathcal{I}$ is implemented as a single interval $\mathcal{I}' = [s, t]$ together with the set $\mathcal{H}$ of the *holes* corresponding to shifts of the removed positions δ. The final set of positions is: $\mathcal{I} = \mathcal{I}' - \mathcal{H}$.

The interval $\mathcal{I}' = [s, t]$ bahaves exactly as the interval $\mathcal{I}$ in the previous algorithm, it is processed in each iteration by manipulating the numbers s, t in constant time as before.

The set of the hole positions is changing differently compared with the algorithm *Compressed-Matching*, since the invariant related to δ is not preserved now, possibly $\delta \neq s$ and $\delta \neq t$. Each time we remove the position δ from $\mathcal{I}$ we are adding it to the set $\mathcal{H}$.

However there is a problem with shifting (by a function $SHIFT$) too many positions in $\mathcal{H}$. Shifting a linear number of holes a linear number of times would give a quadratic algorithm. Therefore we perform the *lazy* shifts of the hole positions. When the position h is inserted into $\mathcal{H}$ for the first time we record h together with the number of the iteration when it happens. Then we do not shift h until the final iteration, when we shift h by the number of shifts multiplied by the the total number of iterations when it was to be shifted.

In this way the processing of all the hole positions is done by a simple arithmetics in linear time only at the final iteration. In the earlier iteration each udpdate of H takes only a constant time.
The asympthotic time complexity of the whole algorithm remains linear. $\square$

Remark. Theorem 2 subsumes Theorem 1 and we could start from Theorem 2. However the algorithm related to Theorem 1 is much simpler and could be compactly written. We have chosen to start with an easily understandable algorithm for a simpler problem, and then present some obscuring details needed to extend this algorithm to cyclic shifts and don't care symbols. The extension is technical, the main structure of the algorithm is preserved.

Example.
Denote by $*$ the *don't care* symbol. For the Fibonacci word

$$w \;=\; abaababaabaababaababa$$

we have

$$p = 8 \text{ and } p^{-1} \; modulo \; N \;=\; 8.$$

For the pattern $x = a**a$ the set $Occ(x, w)$ cannot be ordered as a single arithmetic progression modulo $N = 21$, however it is the union of two arithmetic progressions (where addition is modulo 21):

$$7 \xrightarrow{\oplus\, 8} 15 \xrightarrow{\oplus\, 8} 2 \xrightarrow{\oplus\, 8} 10 \quad \text{and} \quad 5 \xrightarrow{\oplus\, 8} 13 \xrightarrow{\oplus\, 8} 0 \xrightarrow{\oplus\, 8} 8.$$

Remark.
A similar approach to pattern-matching in Sturmian words has been described in [6] for the case of infinite words. In case of finite words the situation is more subtle, in particular after each shift possibly one position is removed (such a situation has no place in infinite case). The other difference is that for infinite case the efficiency is not an issue, there are considered subintervals of [0..1], with endpoints which are not necessarily rational. Also the occurrence shift property is not considered there at all.

References

[1] J.-P. Allouche, J. Shallit, Automatic Sequences: Theory, Applications, Generalizations, Cambridge University Press, UK, 2003.

[2] Automata and Number Theory (CANT 2006), 2006.

[3] P. Baturo, M. Piatkowski, W. Rytter, The Number of Runs in Sturmian Words. In CIAA 2008, 252-261

[4] P. Baturo, M. Piatkowski, W. Rytter, Usefulness of Directed Acyclic Subword Graphs in Problems Related to Standard Sturmian Words, presented at Praque Stringology Conference 2008.

[5] J. Berstel, J. Karhumäki, Combinatorics on words - a tutorial. Bull. EATCS 79 (2003), pp 178-228.

[6] J. Berstel, P. Séébold, Sturmian words, in: M. Lothaire, Algebraic combinatorics on words, (Chapter 2), vol. 90 of *Encyclopedia of Mathematics and its Applications*, Cambridge University Press, Cambridge (2002) 45-110

[7] M. Crochemore, W. Rytter, Jewels of stringology, World Scientific (2003)

[8] Martin Farach, Mikkel Thorup: String matching in Lempel-Ziv compressed strings. STOC 1995: 703-712

[9] O. Jenkinson, L. Zamboni, Characterizations of balanced words via orderings, TCS 2004

[10] A. de Luca, Sturmian words: structure, combinatorics and their arithmetics, Theoret. Comput. Sci. 183 (1997), 4582,

[11] S. Mantaci, A. Restivo, M. Sciortino, Burrows-Wheeler transform and Sturmian words, IPL 86 (2003) 241-246

[12] G. Pirillo, Inequalities characterizing standard Sturmian and episturmian words, Theoret. Comput. Sci. 341 (13) (2005), 276292

[13] G. Pirillo, A new characteristic property of the palindrome prefixes of a standard Sturmian word, Sem. Lothar. Combin. 43 (1999), pp. 3

[14] W. Rytter, Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input. ICALP 2004: 15-27

[15] M. Sciortino, L. Zamboni, Suffix Automata and Standard Sturmian Words. Developments in Language Theory 2007, 382-398