Compressed string-matching in standard Sturmian words☆

Paweł Baturo [a], Wojciech Rytter [b,a,*]

[a] Faculty of Mathematics and Informatics, Copernicus University, Torun
[b] Institute of Informatics, Warsaw University, Warsaw, Poland

ARTICLE INFO

ABSTRACT

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

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

The *standard Sturmian words* (*standard words*, in short) are generalization of Fibonacci words and have many interesting combinatorial properties, see for example [11,9,4,5,15,2,3,12,13,1,10]. 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 the length of the word). We introduce and investigate a similar property of Sturmian words – the *occurrence shift property*.

We fix the alphabet $\Sigma = \{a, b\}$. The length of a word $x \in \Sigma^*$ is denoted by $|x|$. The letter on the ith position in the word x is denoted by $x[i]$. We assume that the first letter is at the position 0.

The subword v of a word x starting at the position i and ending at the position j is denoted by $x[i..j]$.

Standard words are described by the recurrences corresponding to the so-called *directive sequences*

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

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

$$x_{-1} = b, \qquad x_0 = a, \qquad x_{i+1} = x_i^{\gamma_i} x_{i-1} \quad \text{for } 0 \leq i < n. \tag{1}$$

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 $n > 0$ the word x_n has the suffix ba, and for odd $n > 1$ it has the 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 $x = SW(1, 2, 1, 1, 1) = ababaababababaababaab$ written below together with the positions of the letters in x and the corresponding sequence of 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

$$x_{-1} = b, \qquad x_0 = a, \qquad x_1 = x_0^1 x_{-1}, \qquad x_2 = x_1^2 x_0,$$
$$x_3 = x_2^1 x_1, \qquad x_4 = x_3^1 x_2, \qquad x_5 = x_4^1 x_3.$$

A grammar-based compression is given by 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. In fact in this paper we show an algorithm for the 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 [7,14,6].

In the paper [8] the authors present an algorithm for finding the number of occurrences of the pattern pat in an index structure of an arbitrary text x in time $O(|pat| + occ \cdot log^{1+\epsilon} N)$, where occ is the number of occurrences of a pattern pat in x, $0 < \epsilon < 1$ and $N = |x|$. Their data structure uses at most $5N\ H_k(T) + o(n)$ bits of storage, where $H_k(T)$ is the kth order empirical entropy of T.

In our special case of standard words we have $O(|pat| + n)$ searching time for all occurrences of the pattern pat in a standard word x, where n is the size of the directive sequence, while N is usually exponential with respect to the length n of the directive sequence. Our algorithm works in constant space.

Denote by $Occ(pat, x)$ the set of positions where an occurrence of the pattern pat starts in the text x.

We say that a word x has the *occurrence shift property* iff, for any pattern pat, the set $Occ(pat, x)$ can be ordered as a sequence which is an arithmetic progression modulo the length of x. Denote:

$$i \oplus j = (i + j) \bmod N,$$

where N is the length of the Sturmian word x.

Example. (a) For $x = ababaababababaababaab = SW(1, 2, 1, 1, 1)$ we have:

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

The set $\{ 0, 5, 7, 12 \}$ has *occurrence 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 $x = ababaabababaababababa = SW(1, 2, 1, 2)$ we have:

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

since the sequence $(14, 7, 0, 12, 5)$ is the arithmetic progression modulo 19 with the difference 12.

Throughout the examples of this paper we use the word $SW(1, 2, 1, 2)$ for the case of an even n and the word $SW(1, 2, 1, 1, 1)$ for an odd n.

A *compressed representation* of an arithmetic progression consists of 3 numbers: the starting element, the difference, and the last element. We show: for a given sequence γ and an explicitly given pattern pat we can compute a compressed representation of $Occ(pat, SW(\gamma))$ in time $O(|pat| + |\gamma|)$.

2. Compressed string-matching in Sturmian words

There is a useful *circular* interpretation of a standard word x. Let $q = |x|_a$ denote the number of occurrences of a in x and analogously let $p = |x|_b$ denote the number of b's.

Define intervals:

$$\mathcal{l}_a = [0, q - 1], \qquad \mathcal{l}_b = [q, N - 1]$$

For $0 \le j < N$ we define the word $R_{p,j}$ over the alphabet $\{a, b\}$ as follows:

$$R_{p,j}[i] = a \iff j \oplus i \cdot p \in \mathcal{l}_a, \quad \text{for } 0 \le i < N$$

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 jth 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. Then $R_{p,j}$ is the generated word.
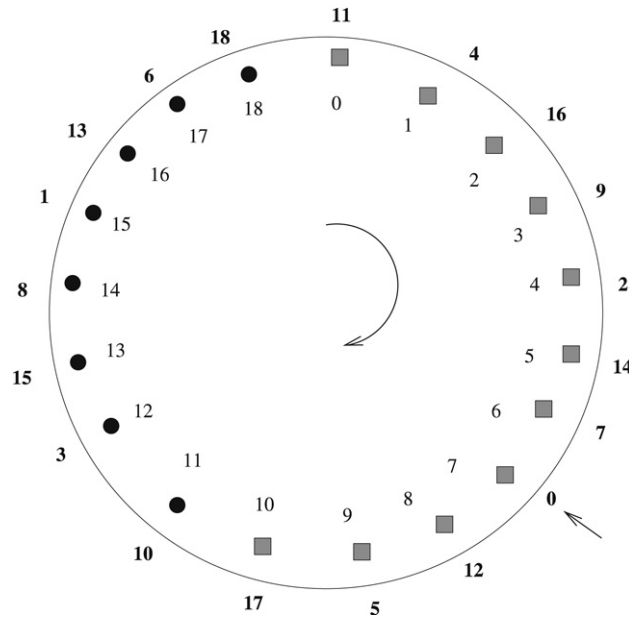
Fig. 1. The mechanical generation of the word $x = R_{8,7}$. The *jump* is $p = 8$ and we start at $j = 7$. The outer numbering corresponds to the sequence *ROT* of start positions of lexicographically sorted cyclic shifts, it is an arithmetic progression with the difference $r = p^{-1} = 12$ (since $8 \cdot 12 = 96 = 5 \cdot 19 + 1$).

Example. In Fig. 1 we have:

$$x = R_{8,7} = ababaabababaababaab, \qquad q = 11, \qquad p = 8, \qquad N = 19.$$

This word is *mechanically* generated starting at (inner) position 7 (marked by the arrow) in Fig. 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 another point then we generate a cyclic shift of *x*.

Let *ROT* be the sequence of start positions of cyclic shifts of a given word *x* listed in the lexicographic order. The cyclic shift is identified with the size of the shift. For a given word *x* of length *N* define

$$shift(i, x) = x[i..N − 1]x[0..i − 1],$$

where $x[0.. − 1]$ 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 $r = p^{-1} \bmod N$, 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:*

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

We can use the following easy fact for the computation of the values of *r* from the point (c).

Fact 1. *Let* $i, m \in \mathbb{Z}$ *and* $gcd(i, m) = 1$, *then there exist* $j \in \mathbb{Z}$ *such that* $i \cdot j = 1 \ (mod \ m)$, *equivalently* $i^{-1} = j \ (mod \ m)$.

Notice that $gcd(p, N) = gcd(q, N) = gcd(p, q) = 1$ (see [11]).

There are two numberings on the circle (see Fig. 1), the inner numbering corresponds to indices in the array *ROT*[∗], the outer numbering to the values of *ROT*[∗].

In our example $8^{-1} = 12$ (*modulo* 19) and the outer numbering is given by formula $(N − 1) \oplus (i + 1) \cdot r = 18 \oplus (i + 1) \cdot 12 = 11 \oplus 12 \cdot i$.

For $0 \leq s \leq t \leq N − 1$ we define the interval $\mathit{l} = [s, t]$ as the set

$$\{s, s + 1, \ldots, t\}.$$

If $s > t$ then define $\mathit{l} = [s, t]$ as the set $\{s, s + 1, \ldots, N − 1, 0, \ldots, t\}$.

For an interval $\mathit{l} = [s, t]$ denote

$$SHIFT_p(\mathit{l}) = [s \oplus p, t \oplus p].$$

Recall that

$$\mathit{l}_a = [0, q - 1], \qquad \mathit{l}_b = [q, N - 1].$$

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

  Input: A pattern *pat* and a directive sequence $(\gamma_0, \gamma_1, \ldots \gamma_{n-1})$;

    Let $x = SW(\gamma_0, \gamma_1, \ldots \gamma_{n-1})$,

    $N := |x|$; $p := |x|_b$; $q := |x|_a$; $m := |pat|$;

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

    if $pat[0] = a$ **then** $\mathit{l} := \mathit{l}_a$ **else** $\mathit{l} := \mathit{l}_b$;

  Forward phase:

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

      Invariant:

        if $\delta \in \mathit{l}$ then δ is the start or the end position of l

      (F1) $\mathit{l} := \mathit{l} - \{\delta\}$;
      (F2) $\mathit{l} := SHIFT_p(\mathit{l})$;
      (F3) **if** $pat[j] = a$ **then** $\mathit{l} := \mathit{l} \cap \mathit{l}_a$ **else** $\mathit{l} := \mathit{l} \cap \mathit{l}_b$;

  Backward phase:

    let s, t be the start and the end position of the interval l;

    $i := ROT[s] - m + 1$; $j := ROT[t] - m + 1$;

    return the arithmetic progression starting in i and ending
    in j, with the difference $r = p^{-1} \bmod N$;

Theorem 1 (*String-Matching in Sturmian Words*). *For a standard word x given by a sequence $\gamma = (\gamma_0, \ldots, \gamma_{n-1})$ and a pattern pat of length m, we can compute $Occ(pat, x)$ 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. We show that the algorithm *String-Matching*(pat, x) computes the set of occurrences within the required complexities.

An *end-occurrence* of the pattern *pat* in x is a position i such that the subword $x[i - m + 1..i]$ equals *pat*. Observe that the set of end-occurrences is an arithmetic progression iff the set of the start positions of the occurrences is. Denote by *End-Occ*(pat, x) the set of end-occurrences of *pat* in x.

If $\mathit{l} = [s, t]$ is an interval then define:

$$ROT(\mathit{l}) = \{ ROT[s], ROT[s + 1], \ldots, ROT[t] \}.$$

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

The next claim follows directly from the circular interpretation of the word x implied by Lemma 1.

Claim 1. **(a)** *If $x[0] = a$ then End-Occ$(pat[0], x) = ROT(\mathit{l}_a)$*

$$\textit{else} \ \ \textit{End-Occ}(pat[0], x) = ROT(\mathit{l}_b).$$

(b) *Assume $ROT(\mathit{l}) = End\text{-}Occ(pat[0..j - 1], x)$.*

  If $x[j] = a$ then End-Occ$(pat[0..j], x) = ROT(SHIFT_p((\mathit{l} - \{\delta\}) \cap \mathit{l}_a))$

$$\textit{else} \ \ \textit{End-Occ}(pat[0..j], x) = ROT(SHIFT_p((\mathit{l} - \{\delta\}) \cap \mathit{l}_b)).$$
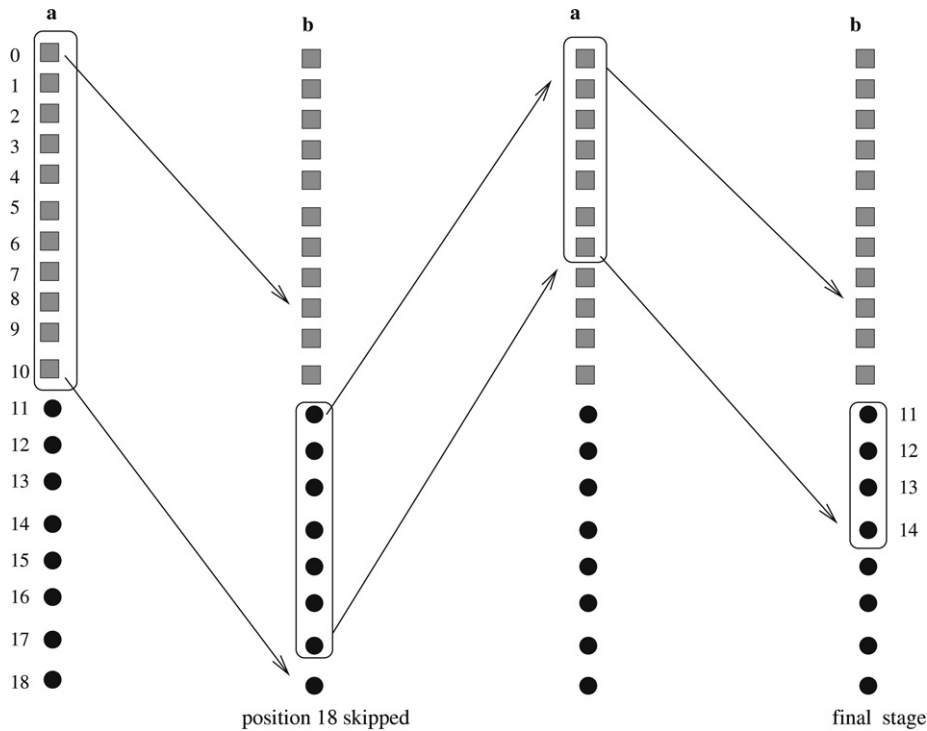
Fig. 2. Computation of *Occ*(*abab*, *x*), the forward phase.

The sequence *ROT* has a special property (expressed by the next claim) responsible for the fact that the output is a single arithmetic progression. Denote by δ the position of the word *shift*(*N* − 1, *x*) in the list of all cyclic shift of the word *x* sorted lexicographically. The following claim is a direct consequence of Lemma 1.

Claim 2. *If x ends with the letter b then ROT*[*N* − 1] = *N* − 1 *and* δ = *N* − 1, *otherwise ROT*[0] = *N* − 1 *and* δ = 0. *In other words the cyclic shift starting at position N* − 1 *of x is either lexicographically the least, or the last.*

Correctness of the value of δ chosen by the algorithm follows from Claim 2.
Claim 1 implies that the returned set is the set of all start-occurrences. In the backward phase end-occurrences become start-occurrences, since we are subtracting from their values the number *m* − 1 (*m* = |*pat*|).
We have still to show the following fact.

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

Each time we extend the prefix of *pat* (processing the next value of *j* < *m* − 1 in the forward phase) we correctly remove the position *N* − 1, since the next position would go outside the word *x*. In this situation we remove the position δ from the interval ℓ since *ROT*[δ] = *N* − 1.

The point δ is at the beginning of ℓ_a or at the end of ℓ_b due to Claim 1. Hence in each iteration when we remove δ then δ is the first or the last in the current arithmetic progression. Consequently the final set of returned values is a single arithmetic progression. This completes the proof of the claim.

Claim 4. *The algorithm works in time O*(|*pat*| + *n*) *and in O*(1) *additional space.*

The size of the word can be computed in *O*(*n*) time, as well as the numbers of letters *a*, *b* in *x*. Once we have the numbers *N*, *p*, *q* the final interval ℓ can be computed in *O*(*m*) time. The remaining parameter is the number *r* = *p*$^{-1}$ mod *N*. It can be computed in the following way (due to [9]):

if *n* is odd then *r* = |*SW*(γ_0, . . . , γ_{n-2})|, otherwise *r* = *N* − |*SW*(γ_0, . . . , γ_{n-2})|.

Hence this parameter can be also computed in linear time.

Consequently the overall complexity is linear. This completes the proof of the claim and the theorem. □

Example. We explain now the main ideas of the proof using illustrations in Figs. 2 and 3 for *pat* = *abab*. In the forward phase, we move the interval [0..10] by shifting it clockwise 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 meet the last position we do not take it, unless it is the final stage. Observe that we omit the number δ = 18 in the first move.
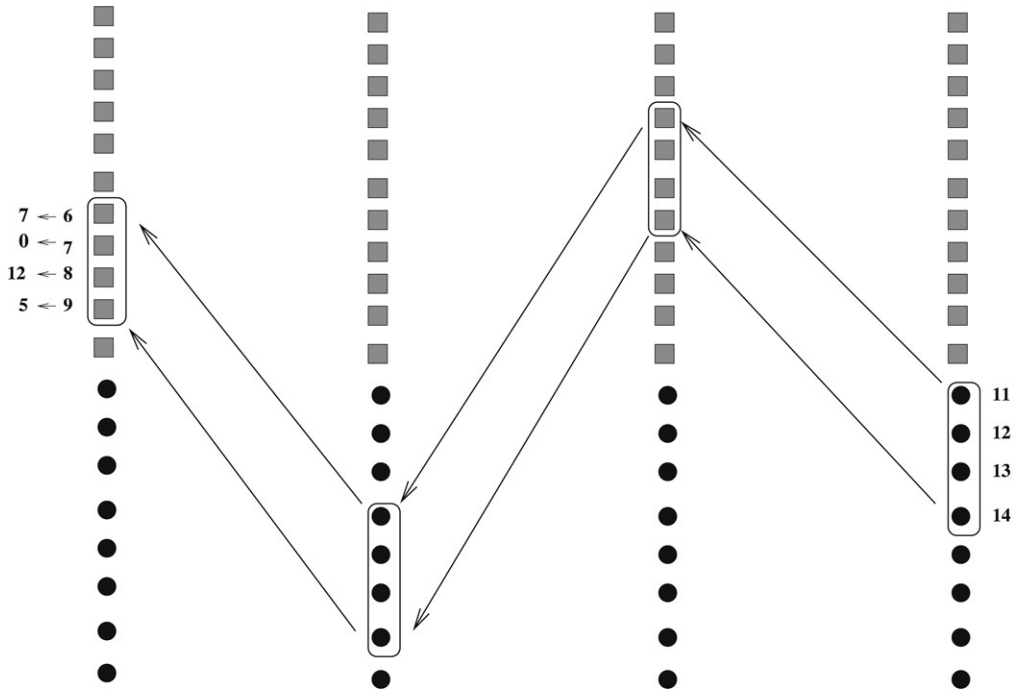
Fig. 3. The computation of $Occ(abab, x)$, the backward phase. We have: $Occ(abab, x) = \{ ROT[6], ROT[7], ROT[8], ROT[9] \} = \{7, 0, 12, 5\}$. It is a part of the arithmetic progression describing the sequence $ROT[*]$.

3. The patterns with a don't care symbol

Let $*$ denotes the *don't care* symbol. This symbol is matching any other symbol, it is a kind of a universal symbol.

For the patterns with the *don't care* symbol the set of occurrences is not necessarily a single arithmetic progression. For example consider the Fibonacci word of length 21 (which is a special case of a Sturmian word) and the pattern $pat = a**a$. The set

$$Occ(a**a, abaababaabaabababaababa) = \{0, 2, 5, 7, 8, 10, 13, 17\}$$

cannot be ordered as a single arithmetic progression modulo $N = 21$.

However it is the union of two arithmetic progressions with difference 8 (where addition is modulo 21):

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

It happens that in general for the pattern pat containing don't care symbols the set $Occ(pat, x)$ consists of at most $|pat|$ arithmetic progressions. The algorithm computing $Occ(pat, x)$ in the presence of don't cares is almost identical to the algorithm described earlier for the case without don't cares.

The next theorem partially subsumes Theorem 1 as a special case. Then we could have started directly from Theorem 2. However there are some reasons for a separate theorem.

An important nontrivial part of Theorem 1 and its proof is the occurrence shift property: the set of occurrences is a single arithmetic progression. This is not a part of Theorem 2. In fact this property does not hold in the presence of don't care symbols.

Secondly, the main result is Theorem 1 and the algorithm related to Theorem 1 is much simpler. We have chosen to start with an easily understandable algorithm for the main result, and then present in Theorem 2 some obscure details needed to extend this algorithm to patterns with don't care symbols.

Theorem 2. *Assume x is a standard word given by a directive sequence of size n and some letters (possibly none) of the pattern pat are replaced by a don't care symbol. Then we can compute a compressed representation of $Occ(pat, x)$ in time $O(|pat| + n)$. The computed representation consists of at most $|pat|$ arithmetic progressions (modulo N).*

Proof. We can apply essentially the same algorithm *String-Matching*(pat, x) to extend Theorem 1. There are 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 l with l_a nor l_b, the current interval l remains unchanged. This means that we change the statement (F3) in the forward phase to:

(F3) **if** $pat[j] \neq *$ **then**
      if $pat[j] = a$ **then** $\mathit{l} := \mathit{l} \cap \mathit{l}_a$ **else** $\mathit{l} := \mathit{l} \cap \mathit{l}_b$.

The invariant written in the forward phase does not hold now. Consequently, there is a problem with the positions δ removed in statement (F1), since they can be internal positions in $\mathcal{I}$ and the number of them can grow linearly. It would be too expensive to process them in each individual *SHIFT* in the forward phase.

Therefore we remove the positions indirectly. The current set $\mathcal{I}$ is implemented as a single interval and instead of removing the actual position δ we insert it into the set $\mathcal{H}$ of positions which should be removed. The positions in $\mathcal{H}$ are called here the *holes*. We know that after the iteration j there are still $(m - 1 - j)$ iterations and the positions are to be shifted $(m - 1 - j)$ times, each shift results in the addition (modulo N) of the number p. Hence in a given iteration j we postpone the removal of δ, and finally (after the forward phase) this removal corresponds to the deletion of the position

$$rem(j, \delta) = \delta \oplus [(m - 1 - j) \cdot p].$$

The statement (F1) is now changed into:

(F1)   insert $rem(j, \delta)$ into $\mathcal{H}$.

Then at the end of the forward phase we execute:

$$\mathcal{I} := \mathcal{I} - \mathcal{H}.$$

Such deletions can be interpreted as *lazy deletions*. In this way the processing of all the holes is done by simple arithmetics in linear time after the final iteration in the forward phase.

After the forward phase the set $\mathcal{I}$ is an interval with linear number of removed elements, in other words it is a set of linearly many disjoint intervals

$$[s_1, t_1], [s_2, t_2], \dots [s_k, t_k].$$

Then the whole (modified) algorithm returns k arithmetic progressions, the ith progression starts in $ROT(s_i) - m + 1$ and ends in $ROT(t_i) - m + 1$.

All these computations can be done in $O(|pat| + n)$ time. Therefore the time complexity of the whole (modified) algorithm remains linear. $\square$

4. Final remarks

In the case of cyclic shifts of standard Sturmian words the occurrence shift property is no longer valid. For example consider our example word $x = SW(1, 2, 1, 1, 1) = ababaabababababaabababaab$ and let

$$x' = x[9..18]x[0..8] = abaababaabababababaabab.$$

Then $Occ(ba, x')$ is not a single arithmetic progression. Although the cyclic shifts of standard words do not necessarily have the occurrence shift property, the set $Occ(pat, x)$ of occurrences in such words can be also described in a compressed way using at most $|pat|$ arithmetic progressions.

A similar algorithm as the one presented in the proof of Theorem 2 can be applied to cyclic shifts, again the main point is the processing of the set of *holes* (positions to be removed). The solution does not need new ideas, it is enough to change the numbering of positions due to the cyclic shift of the text, such a renumbering would technically complicate the algorithm but the structure remains the same.

A similar approach to the pattern-matching in Sturmian words has been described in [5] for the case of infinite words. In the 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 efficiency is not an issue. 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] P. Baturo, M. Piatkowski, W. Rytter, The number of runs in Sturmian words, CIAA, 2008, pp. 252–261.
[3] 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.
[4] J. Berstel, J. Karhumäki, Combinatorics on words - a tutorial, Bull. EATCS 79 (2003) 178–228.
[5] J. Berstel, P. Séébold, Sturmian words, in: M. Lothaire, Algebraic Combinatorics on Words, in: Encyclopedia of Mathematics and its Applications, vol. 90, Cambridge University Press, Cambridge, 2002, pp. 45–110. (chapter 2).
[6] M. Crochemore, W. Rytter, Jewels of Stringology, World Scientific, 2003.
[7] Martin Farach, Mikkel Thorup, String matching in Lempel-Ziv compressed strings, STOC 1995, pp. 703–712.
[8] Paolo Ferragina, Giovanni Manzini, Indexing compressed text, J. ACM 52 (2005) 552–581.
[9] O. Jenkinson, L. Zamboni, Characterizations of balanced words via orderings, Theoret. Comput. Sci. 310 (1–3) (2004) 247–271.
[10] A. de Luca, Sturmian words: Structure, combinatorics and their arithmetics, Theoret. Comput. Sci. 183 (1) (1997) 45–82.
[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 (1) (2005) 276–292.
[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, pp. 15–27.
[15] M. Sciortino, L. Zamboni, Suffix automata and standard Sturmian words, in: Developments in Language Theory, 2007, pp. 382–398.