Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Two fast constructions of compact representations of binary words with given set of periods

Wojciech Rytter [1]

Department of Mathematics, Computer Science and Mechanics, Warsaw University, Warsaw, Poland

ARTICLE INFO

Article history:
Received 7 August 2015
Received in revised form 12 April 2016
Accepted 25 April 2016
Available online xxxx

Keywords:
Periodicity
Collage system
Algorithm
Borders

ABSTRACT

Assume we are given a sorted set $\mathcal{P}$ of size n of all periods of an unknown string of size N. Our main result is an algorithm generating in $O(n)$ time and $O(1)$ space a compressed representation of size $O(n)$ of the lexicographically-first binary string having $\mathcal{P}$ as the set of its periods. The input is read-only and the output is write-only. We also present a very simple preliminary algorithm generating some binary string (not necessarily lexicographically-first one) with the same set of periods. The *explicit* size N of the produced string can be exponential with respect to n. We assume that a given set of periods is valid: there *exists* some unknown string realizing this set.

1. Introduction

Periodicity plays central role in combinatorics and algorithms of words. A period of a string w is an integer $0 < p \leq |w|$ such that $w[i+p] = w[i]$, whenever both sides are defined. Periodicities are extensively used as crucial combinatorial tools in construction of many efficient algorithms on strings, see [3] and [5].

One of natural questions about periodicities is how to construct strings with a given periodicity structure and small alphabet. This helps to understand better the periodicities in string, see [2,6]. Many important properties of the family of all sets of periods for strings of a given length are investigated in [2].

We say that a word w realizes a given set $\mathcal{P}$ of periods iff $\mathcal{P}$ is the set of all periods of w. A set $\mathcal{P}$ is a *valid* set of periods iff there is a string realizing it as the set of its periods. In this paper for each considered set of periods we assume that it is valid: there *exists* some unknown string realizing this set.

It is known (and rather surprising and quite nontrivial) that binary alphabet is sufficient in the following sense: any valid set of periods (no restriction on the alphabet) is realized by a *binary* string.

The first algorithm to construct a binary string w realizing a given set of periods was given in [7]. Then, using a different and simpler approach, it has been shown in [4] that w can be constructed in $O(|w|)$ time.

The interesting point here was that the constructed string is binary, though an unknown string can be over arbitrary alphabet. However there is another interesting point: the construction given in [4] implicitly shows that the produced string $|w|$ is highly compressible.

In this paper we give two fast and simple algorithms which produce compact representations of highly compressible binary strings w realizing a given set of periods in time linear with respect to the size of the representation, which is $O(n)$. The explicit size N of w can be exponential with respect to n.

E-mail address: rytter@mimuw.edu.pl.

[1] The author is supported by grant no. NCN2014/13/B/ST6/00770 of the National Science Centre.

http://dx.doi.org/10.1016/j.tcs.2016.04.027
0304-3975/

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

A string-period of a word w is any string $z = uv$ such that $w = (uv)^k u$, where u is a proper prefix of z (possibly empty) and k is a positive integer. Hence an integer p is a period of w iff it is the size of some string-period of w. For example the word *ababa* has 3 string-periods $\{ab, abab, ababa\}$. The periodic structure of a string is given by the set $\mathcal{P}(w)$ of integers which are its periods.

A border is an integer $1 \le q \le |w|$ such that $|w| - q$ is a period of w or $q = |w|$. A border-string of w is a string which is simultaneously a prefix and a suffix of w. We distinguish between borders (integers) and string-borders. Consequently borders are sizes of string-borders for a given string. In this paper we consider equivalent structure: integer sequences $\mathcal{B}$ of borders, since they are more convenient in the description of algorithms. Instead of periodic structure of a given string w we use the border structure given by the increasing sequence $\mathcal{B}(w) = (q_1, q_2, \ldots q_n)$ of sizes of all string-borders of w, including $q_n = |w| = N$.

Example 1.1. The sequence of periods of the string $w = abaaba$ is $\mathcal{P} = (3, 5, 6)$ and its sorted sequence of borders is $\mathcal{B} = (1, 3, 6)$. In this case the length of an unknown string is $N = 6$, and the length of the border sequence in $n = 3$.

A string v is said *to realize* the border sequence $\mathcal{R}$ iff $\mathcal{R} = \mathcal{B}(v)$ is the border sequence of v. In the whole paper we assume the input border $\mathcal{R}$ is *valid*, which means that there exists some (possibly unknown) string w realizing $\mathcal{R}$, in other words $\mathcal{B}(w) = \mathcal{R}$.

There are many types of compressed representation of words, see [9]. One such representation is in terms of *collage systems*, see [1]. A collage system defines larger strings in terms of concatenations of parts of smaller, already defined, strings. Such system is an extension of *grammar-based compression*, see [9]. The collage system defines the sequence of string variables, where we can use prefixes/suffixes of previously defined values of smaller variables to define the values of larger ones.

For a string x denote by $Pref(X, j)$ the prefix of X of size j. A collage system is formally defined as a sequence of assignments:

$$X_1 = expr_1; X_2 = expr_2; \ldots; X_n = expr_n,$$

where each X_k is a variable and $expr_k$ is of one of the forms:

- a single letter,
- $X_i \cdot X_j$ for $i; j < k$,
- $Pref(X_i, j)$ for $i < k$ and an integer j,
- $(X_i)^j$ for $i < k$ and an integer j.

Assume we know $\mathcal{B}$ of some unknown string, but we don't know the string w. In the reconstruction process we construct two binary strings realizing $\mathcal{B}$.

1. A binary string, denoted by $Alternating(\mathcal{B})$, highly compressible but without particular structure otherwise. It is interesting since it gives an especially simple reconstruction algorithm.
2. The string $LexFirst(\mathcal{B})$ which is the lexicographically-first binary string realizing $\mathcal{B}$. This string has also a short compressed representation which will be computed in $O(n)$ time.

Example 1.2. Assume the "unknown" string is the Fibonacci word *abaababaabaab*, its border sequence is $\mathcal{B} = (2, 5, 13)$. The first algorithm gives as the output the string 0110100001101 and the second one will give the lexicographically-first string 0100100001001. Both of them realize the given border sequence $(2, 5, 13)$.

Both our algorithms have similar structure.

$X_1 =$ a border-free string of size q_1;
for $i = 1$ **to** $n - 1$ **do**
  if $q_{i+1} - q_i \le q_i$ **then** $write(\text{“}X_{i+1} = Pref(X_i, q_{i+1} - q_i) \cdot X_i\text{”})$;
  else
    $\sigma_i := q_{i+1} - 2 \cdot q_i$;
    $write(\text{“}X_{i+1} = X_i \cdot Y_i \cdot X_i\text{”})$ where Y_i is a binary string of length σ_i
    such that $X_i \cdot Y_i \cdot X_i$ has no proper string-border longer than $|X_i|$;

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

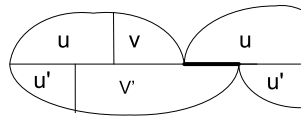

Fig. 1. Illustration of Lemma 2.4. The substring w distinguished by the thick line is a string-period of u and is also a suffix of v'. If v' is unary, then w is unary, hence the whole word is unary – a contradiction.

It is a general scheme of an algorithm, depending on the specific selection of Y_i (which fills the "gap"). This string will be here highly compressible: $Y_i \in 0^+ \cup 0^*1$. The case when $q_{i+1} - q_i \leq q_i$ (the gap is empty) is easy. In this case correctness follows from the following fact which is a reformulation of Lemma 5 in [4]:

Lemma 1.3. *If $(q_1, ..., q_i)$ is the border sequence of X_i and $(q_1, ..., q_i, q_{i+1})$, with $q_{i+1} - q_i \leq q_i$, is the border sequence of some string, then $(q_1, ..., q_i, q_{i+1})$ is the border sequence of $Pref(x, q_{i+1} - q_i) \cdot X_i$.*

The algorithm from [4] is essentially of the same form. *The main difference* between the algorithm from [4] and our two next algorithms is in choice of Y_i in case when $q_{i+1} - q_i > q_i$ In [4] a different binary string Y_i is computed: it is used a version of Knuth–Morris–Pratt algorithm, see [8], on the *explicit* uncompressed string X_iY_i for two possible candidates of Y_i. Consequently in [4] they have an algorithm of time $O(N)$, where $N = |X_n|$. In this paper we do it differently: each choice of Y_i is done in constant time due to several new simple combinatorial properties of unary extensions of non-unary words. Another difference is that our second algorithm gives a very special string: the lexicographically-first binary string with a given set of periods.

2. Several useful combinatorial properties of words

The crucial role in correctness of our algorithms play combinatorial properties of periodicities, primitive words and non-unary words.

The word u is said to be **unary** if $u \in 0^+ \cup 1^+$ (all letters of u are the same).

The length of the shortest string-period of x is denoted by $per(x)$. If uv is a shortest string-period of a nonempty word x, and $x = (uv)^k u$, where k is a positive integer, and u is a proper prefix of uv, then we denote $tail(x) = v$.

Example 2.1. $tail(abcd) = abcd$, $tail(abaab) = a$, $tail(abaababa) = ab$.
If Fib_n is the n-th Fibonacci word then $tail(Fib_n) = Fib_{n-2}$.

The following lemma, see [5,6], gives a basic property of the set of periods. Denote by $gcd(p, q)$ the greatest common divisor of p, q.

Lemma 2.2 *(Periodicity lemma). If p, q are periods of a word w and $|w| \geq p + q - r$ then r is also a period of w, where $r = gcd(p, q)$.*

The crucial role in the algorithm in [4] is played by primitive words. A word w is said to be *primitive* if its is not of a form $w = z^k$, where k is an integer larger than one.

Corollary 2.3. *Assume that for nonempty words u, z the word uz is primitive. Then the word uzu has no proper string-border of the form $uz\alpha$, where α is a proper prefix of u (in other words uzu has no proper period smaller than $|uz|$).*

Proof. Assume uzu has such a string-border, then it has two periods $p = |uz|$, $q = |uzu - |uz\alpha| = |u| - |\alpha|$, where $p > q$. We have $|uzu| \geq p + q$, so due to the periodicity lemma uzu has also the period $gcd(p, q)$ which is smaller than $|uz|$ and divides $|uz|$. Consequently, uz is nonprimitive. We get a contradiction. □

Lemma 2.4. *If x is non-unary, and $x = (uv)^k u = u'v'u'$, where uv is the shortest string-period of x, $k \geq 1$ and $|u'| < |u|$, then v' is non-unary.*

Proof. Observe that $|v'| > 0$. If $k > 1$ then v' contains as a subword the string-period uv of x, which is non-unary since x is.

Hence it is enough to consider the case $k = 1, |u| > 0$ illustrated in Fig. 1. Then v' and u have a nonempty overlap (thick segment in the figure), which is a string-period of u. Then if v' is unary then u is also, consequently the whole x is unary – a contradiction. Therefore v' is non-unary. □

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

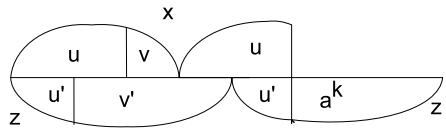

Fig. 2. Case 1: $|u'| < |u|$. Assume uv is the shortest string-period of x. If the unary extension $v' = a^k$ is longer than $tail(x) = v$ and xv' is a square, then, due to Lemma 2.4, v' is non-unary, a contradiction. Consequently xa^k cannot be of the form z^2.

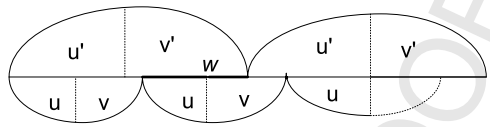

Fig. 3. Case 2: $|u'| > |u|$. We have: $x = (uv)^2u = u'v'u'$. Thick segment represents the string $w \in a^+$ such that $uvw = u'v'$. w is a string-period of uvu.

Lemma 2.5. *Assume x is non-unary, $a \in \{0, 1\}$ and $k > 0$, then*

$$xa^k \text{ is nonprimitive} \quad \Leftrightarrow \quad a^k = tail(x).$$

Proof. Assume $x = (uv)^iu$, where uv is the shortest string-period of x and $v = tail(x)$.
If $a^k = tail(x) = v$ then $xa^k = (uv)^iu \cdot v = (uv)^{i+1}$, hence xa^k is nonprimitive.
Now we show the implication:

$$\text{if } a^k \neq tail(x) \text{ then } xa^k \text{ is primitive.}$$

The proof is by contradiction. Assume that xa^k is nonprimitive, we have:

$$(*)\ a^k \neq tail(x) \text{ and } xa^k = z^j,\ j \geq 2,\ |z| = per(xa^k).$$

It is sufficient to show that conditions (*) lead to a contradiction.
The conditions (*) imply directly the following fact.

Claim 2.6. $per(x) \neq per(xa^k)$.

If $j > 2$ then the whole two string-periods of xa^k would be inside x and the shortest-string period of x will be equal to z but it is not possible due to Claim 2.6. Also we cannot have $|u'v'| < |uv|$, since uv is the shortest string-period of x and z is a different period of x.
Consequently the only remaining case in this situation is $j = 2$ and we have:

$$xa^k = z^2 = u'v'u'v',\ \ x = (uv)^iu = u'v'u',\ \ v' = a^k,$$

where $v' \neq v = tail(x)$, uv is the shortest string-period of x and $u'v'$ is the shortest string-period of xa^k.

Consider two cases:

Case 1: ($|u'| < |u|$) We have a situation illustrated in Fig. 2. We use now Lemma 2.4 which implies that $v' = a^k$ is non-unary, which is a contradiction.

Case 2: ($|u'| > |u|$) Let us consider only the case $i = 2$, i.e. $x = (uv)^2u$, the general case $i > 1$ can be considered similarly. We have a situation illustrated in Fig. 3 for $i = 2$.

Claim 2.7. *If conditions $(*)$ hold, $x = (uv)^2u$ and $per(x) = uv$ then $|u'| < |uv|$.*

Proof of the claim. The claim is proved by contradiction. If $|u'| \geq |uv|$ then the string x has the periods $p = |uv|$, $q = |x| - |u'| = |u'v'|$, where $p + q \leq |x|$. The periodicity lemma implies that p (as a smallest period) is a divisor of q. Hence p is also a period of xa^k (which has a period q). Consequently x and xa^k have the same shortest period, which is impossible due to Claim 2.6. □

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

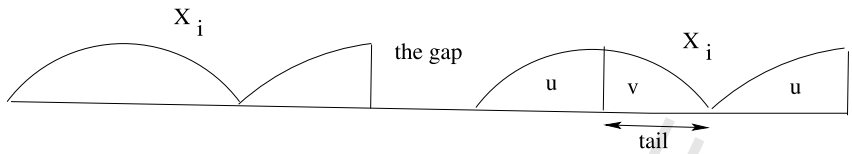

Fig. 4. The structure of X_{i+1} when the size σ_i of the gap between two copies of X_i is nonzero. The gap is filled in the alternating algorithm by a unary string from a^+, where a alternates between 0 and 1 in each move of this type. If the gap is nonempty than $tail(X_{i+1})$ becomes the value of the gap.

Let w be the string such that $u'v' = uvw$. Due to the claim above we have that w is a suffix of v', consequently w is unary. We have the situation illustrated in Fig. 3. The string u' is a prefix of uvu (as a prefix of x). This implies that w is a string-period of uvu. w is unary, hence uv is also unary and the whole string x is unary – a contradiction.

In both cases we have a contradiction. Therefore the word xa^k is primitive. This completes the proof. □

Corollary 2.8. *Assume u is non-unary, z is unary, $z \neq tail(u)$ and z is nonempty. Then uzu has no proper (additional) string-border longer than $|u|$*

Proof. Lemma 2.5 implies that uz is nonprimitive, consequently due to Corollary 2.3 uzu cannot have a proper string-border longer than uz.

Assume now that uzu has a proper string-border longer than u and shorter than uz. Then for some proper nonempty suffix v of z the word vu is a border-string on uzu of uzu. Hence v is a string-period of vu, since u is a prefix of vu (both are prefixes of uzu). Consequently, the word u is unary, since its string-period is unary. We get a contradiction.

Hence there is no proper string-border longer than u. □

Observation 2.9. *If we reverse the roles, u is unary, z is non-unary, then it can happen that uz is primitive but uzu has a proper string-border longer than u. For example take $u = 0$, $z = 010$.*

The next lemma was crucial in [4].

Lemma 2.10. [4] *For any word x the word $x0$ or the word $x1$ is primitive.*

3. A simple algorithm

Denote $\sigma_i = q_{i+1} - 2q_i$. If $q_{i+1} - q_i > q_i$ then σ_i is the length of some string Y_i which fills the gap between two copies of X_i, see Fig. 4.

In our first algorithm we alternately fill nonempty gaps with ones, zeros, ones, zeros etc.

```
ALGORITHM Alternating(B); { B = (q_1, q_2, ... q_n) }
a := 0;  write("X_1 = 01^{q_1 - 1}");
for i = 1 to n - 1 do
    if q_{i+1} - q_i ≤ q_i then
      write("X_{i+1} = Pref(X_i, q_{i+1} - q_i) · X_i") ;
    else
      a := negation(a);  σ_i := q_{i+1} - 2 · q_i;
      write("X_{i+1} = X_i · a^{σ_i} · X_i");
```

Example 3.1. The string generated for the border sequence given below has 6364 symbols, but its compressed representation computed by the alternating algorithm is very short. Let $q_1 = 1$, $q_{i+1} = 2q_i + 100$ for $1 \leq i < 7$. For $\mathcal{B} = (q_1, \ldots q_7)$ the input and output are:

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

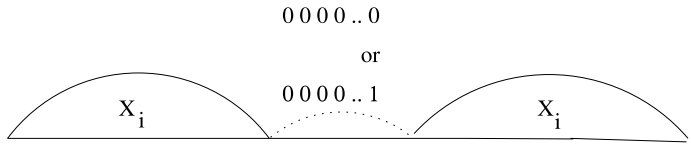

Fig. 5. Assume $q_{i+1} - q_i > q_i$ (the gap is nonempty). In the algorithm LexFirst we fill the gap with zeros (if $tail(X_i) \notin 0^+$ or $|tail(X_i)| \neq \sigma_i$) or with 00..01 (otherwise). We need only to remember the size of $tail(X_i)$ and information whether $tail(X_i) \notin 0^+$ (one bit). The size of the gap is σ_i and the gap (after filling) becomes $tail(X_{i+1})$.

Input: $\mathcal{B} = (1, 102, 304, 708, 1516, 3132, 6364)$

Output: $X_1 = 0;\ X_2 = X_1 1^{100} X_1,\ X_3 = X_2 0^{100} X_2;\ X_4 = X_3 1^{100} X_3$

$$X_5 = X_4 0^{100} X_4,\ X_6 = X_5 1^{100} X_5,\ X_7 = X_6 0^{100} X_6.$$

Theorem 3.2. *The alternating algorithm computes a compressed representation of a binary string with a given sequence $\mathcal{B}$ of borders. It works in $O(n)$ time and constant space without counting the space of the input (read-only) and the output (write-only).*

Proof. It is clear that all numbers q_i, for $i \leq n$, are borders of the produced string. We need only to prove that no additional border appears.

The algorithm works correctly if the input consists only of ones. Hence assume it is not the case.

Let X_t be the first X_t not consisting only of ones. It is clear that until this moment the algorithm works correctly. Then whenever we execute the instruction $X_{i+1} := X_i \cdot a^{\sigma_i} \cdot X_i$ for $i \geq t$ we have two possibilities:

(1) $tail(X_i)$ is unary and consists of symbols $negate(a)$ hence $tail(X_i) \neq a^{\sigma_i}$, and according to Corollary 2.8 we have not created any additional string-border in X_{i+1}.
(2) $tail(X_i)$ is non-unary. In this case also Corollary 2.8 can be applied.

When we execute $X_{i+1} := Pref(X_i, q_{i+1} - q_i) \cdot X_i$ the tail is not changing if the same periodicity continues. In the case when $q_{i+1} - q_i \leq q_i$, X_i is non-unary and $per(X_i) \neq q_{i+1} - q_i$ the tail of X_{i+1} becomes non-unary due to Lemma 2.4. In the latter case we can fill the next gap with any unary string (zeros or ones).

There are n iterations and each of them works in $O(1)$ time. Hence the algorithm works in $O(n)$ time. □

4. Computing the compressed representation of *LexFirst*($\mathcal{B}$)

We consider now computing *LexFirst*($\mathcal{B}$). The algorithm is a greedy one. Whenever it needs to fill the gap it considers only two candidate-strings: of the form 0^+ or 0^*1, preferring the first one, see Fig. 5. The decision is based upon two pieces of information (constant memory):

1. $AllZero = (tail(X_i) \in 0^+)$ (one bit);
2. $TailSize = |tail(X_i)|$ (one integer).

We do not need to remember the value of $tail(X_i)$ if $tail(X_i) \notin 0^+$.

Informal description of the algorithm. We initialize X_1 and parameters *TailSize*, *AllZero*. Then we compute string-borders $X_2, X_3, .., X_n$ in this order. When we compute X_{i+1} there are three cases depending on the length of the value of the gap.

- If the gap is empty and $q_{i+1} - q_i = q_i - q_{i-1}$, which is equivalent to $per(X_{i+1}) = per(X_i)$, then parameters *TailSize*, *AllZero* do not change.
- If the gap is empty and $q_i - q_{i-1} < q_{i+1} - q_i$ then *AllZero* becomes *false* ($tail(X_{i+1})$ is not unary) and we do not care about the *TailSize* value.
- If the gap is nonempty ($q_{i+1} - q_i > q_i$) we fill the gap with zeros if $tail(X_i) \notin 0^+$ or $|tail(X_i)| \neq \sigma_i$, otherwise we fill it with 00..01. We update the value of *AllZero*.

The pseudocode of our algorithm for $q_1 > 1$ is presented below.

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

```
ALGORITHM LexFirst(B); { Assume q_1 > 1 }

write("X_1 = 0^{q_1 - 1} 1");

TailSize := q_1; AllZero := false; q_0 := 0;

for i = 1 to n − 1 do
    if q_{i+1} − q_i ≤ q_i then
        write("X_{i+1} = Pref(X_i, q_{i+1} − q_i) · X_i");
        if q_i − q_{i−1} < q_{i+1} − q_i then AllZero := false;
    else

        σ_i := q_{i+1} − 2 · q_i;
        if AllZero & (σ_i = TailSize) then
            write("X_{i+1} = X_i · 0^{σ_i − 1} · 1 · X_i"); AllZero := false;
        else
            write("X_{i+1} = X_i · 0^{σ_i} · X_i"); AllZero := true;
        TailSize := σ_i;
```

We consider here only the case when $q_1 > 1$ to simplify presentation. If $q_1 = 1$ then we find the first $t > 0$ with $q_{t+1} - q_t > 1$, we write at the beginning $X_{t+1} = 0^t\, 0^{\sigma_t - 1} 1\, 0^t$ instead of X_1 and we start with $i = t + 1$ instead of $i = 1$. The point here is to start with a non-unary word (to simplify the algorithm). Recall that we also assumed that the input border sequence is valid for some string.

Example 4.1. The input–output for an example border sequence is:

Input: $\mathcal{B} = (2, 7, 17, 32, 70)$.
Output: $X_1 = 01;\ X_2 = X_1 0^3 X_1,\ X_3 = X_2 0^2 1 X_2;$

$$X_4 = \mathit{Pref}(X_3, 15) \cdot X_3),\ X_5 = X_4 0^6 X_4.$$

The history of the algorithm for this example looks as follows:

(Initialization) We start with $X_1 = 01$, $\mathit{AllZero} = \mathit{false}$.

(After 1st iteration) $X_2 = 0100001$, $\mathit{TailSize} = 3$, $\mathit{AllZero} = \mathit{true}$;

(Second iteration) We have $\sigma_2 = 17 - 2 \cdot 7 = 3 = \mathit{TailSize}$ so we cannot fill the gap only with zero. If we did it then the obtained string 01000 01000 01000 01 could have a additional border 12. Hence in this iteration we fill the gap with 001.

(After third iteration) We set $X_4 = \mathit{Pref}(X_3, 15) \cdot X_3)$. The algorithm sets $\mathit{AllZero} = \mathit{false}$ since $10 = q_3 - q_2 < q_4 - q_3 = 15 < q_3$.

(The last iteration) Since $\mathit{AllZero} = \mathit{false}$ we do not care about the value of *TailSize* which now is not equal $|\mathit{tail}(X_4)|$. The gap is filled by zeros.

Theorem 4.2. *The algorithm LexFirst computes a compressed representation of the lexicographically-first binary string with a given sequence $\mathcal{B}$ of borders. It works in $O(n)$ time and constant space without counting the space of the input (read-only) and the output (write-only).*

Proof. First we prove that the algorithm produces the string realizing a given border sequence.

If $\mathit{tail}(X_i) \neq 0^{\sigma_i}$ and the gap is nonempty ($q_{i+1} - q_i > q_i$) then we fill it with a string z consisting of zeros. Hence, due to Corollary 2.8 $X_i \cdot z \cdot X_i$ has no proper border longer than $|X_i|$. We do not create any additional border.

We still have to consider the second case, when $\mathit{tail}(X_i) = 0^{\sigma_i}$. The following fact is useful.

Claim 4.3. *If uz is primitive, z ends with the letter 1 and u has a factor consisting of $|z|$ zeros then uzu has no proper string-border longer than u.*

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

Proof. Due to Corollary 2.3 the string uzu has no proper string-border longer then uz. Assume now that uzu has a proper string-border longer than u and shorter than uz. Then, the same argument as in the proof of Corollary 2.8 implies that there is a nonempty suffix v of z which is a string-period of vu. However u has a factor consisting of $|z|$ zeros, so it cannot have a string period of length at most $|z|$ which ends with the letter 1. This contradiction shows that uzu has no proper string-border longer than u. □

If $tail(X_i) = 0^{\sigma_i}$ then $X_i \cdot 0^{\sigma_i - 1}1$ is primitive due to Lemma 2.10, since the string $X_i \cdot 0^{\sigma_i}$ is nonprimitive.

Consequently, $X_i \cdot 0^{\sigma_i - 1}1 \cdot X_i$ has no additional string-border longer than X_i due to the claim above with $u = X_i$, $z = 0^{\sigma_i - 1}1$.

In both cases the algorithm behaves correctly: it does not create any additional border and the produced string has the required sequence of borders.

We have still to show that the produced output is lexicographically-first. It follows from the following fact.

Claim 4.4. *The string X_i produced by the algorithm is the lexicographically-first binary string realizing the border sequence $\mathcal{B}_i = (q_1, q_2, \ldots q_i)$ for $i \leq n$.*

The proof is by induction on i. The thesis is certainly true for $i = 1$, due to initialization. Assume now that the thesis holds for $i < n$. We show that it holds for $i + 1$. Let X be the lexicographically-first binary string realizing the border sequence $\mathcal{B}_{i+1}$. It is enough to show $X = X_{i+1}$. The string X_i should be the prefix of X, since it has to realize $\mathcal{B}_i$ and is the lexicographically-first one which is doing that. Hence X_i is a string-border of X.

If $q_{i+1} - q_i \leq q_i$ then X_i completely determines X as well as X_{i+1}, so both strings are the same.

Otherwise $X = X_{i+1} \cdot Y \cdot X_{i+1}$ for some string Y of size σ_i. The algorithm first tries to insert only zeros between two X_i's. If it succeeds then the produced string is lexicographically-first, since the string consisting only of zeros is lexicographically-first among binary strings of the same length. If it is not possible then Y cannot consist only of zeros. Then our algorithm inserts successfully the lexicographicaly next string, which is of the form 0^*1. Hence in this case Y is the same as the string inserted by our algorithm between two copies of X_i. In both cases $X_{i+1} = X$. This completes the proof of the claim.

Claim 4.5. *The algorithm works in $O(n)$ time and uses $O(1)$ space, not counting the (read-only) input and (write-only) output.*

The complexity bounds follow directly from the algorithm description. There are n iterations and each of them works in $O(1)$ time. Also we use only a constant number of additional variables. This completes the proof. □

5. Final remarks

The period structure of a word would have even more compact representation. The string of size N has periods (borders) which can be grouped in $O(\log N)$ arithmetic progressions. The input to our algorithms can be in the form of logarithmically many such arithmetic progressions. Then the compact representation (in terms of collage systems) can be computed in linear time with respect to the size of such compacted input. No new ideas are needed.

We assumed throughout the paper that the input sequence is valid: it is realized existentially by some unknown string. However checking if the border or period sequence is valid is quite easy and can be independently done by a linear time algorithm (folklore).

The periods (borders) string reconstruction was interesting mostly because the binary alphabet is sufficient (which was rather surprising). We showed here that it is also interesting in a compressed setting.

References

[1] Takuya Kida, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, Setsuo Arikawa, A unifying framework for compressed pattern matching, Theoret. Comput. Sci. 1 (2003) 253–272.
[2] Sven Rahmann, Eric Rivals, Combinatorics of periods in strings, J. Combin. Theory Ser. A 104 (2003) 95–113.
[3] Dan Gusfield, Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.
[4] Vesa Halava, Tero Harju, Lucian Ilie, Periods and binary words, J. Combin. Theory Ser. A 89 (2000) 298–303.
[5] Maxime Crochemore, Christophe Hancart, Thierry Lecroq, Algorithms on Strings, Cambridge University Press, Cambridge, 2007.
[6] M. Lothaire, Combinatorics on Words, Cambridge University Press, Cambridge, 1997.
[7] Leonidas J. Guibas, Andrew M. Odlyzko, Periods in strings, J. Combin. Theory Ser. A 30 (1981) 19–42.
[8] Donald E. Knuth, James H. Morris Jr., Vaughan R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1977) 323–350.
[9] W. Rytter, Grammar compression lz encodings and string algorithms with implicit input, in: ICALP 2004, in: LNCS, vol. 5805, Springer, 2009, pp. 15–27.

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027

Sponsor names

Do not correct this page. Please mark corrections to sponsor names and grant numbers in the main text.

National Science Centre, *country*=Poland, *grants*=NCN2014/13/B/ST6/00770

Please cite this article in press as: W. Rytter, Two fast constructions of compact representations of binary words with given set of periods, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.04.027