# Computing $k$-th Lyndon Word and Decoding Lexicographically Minimal de Bruijn Sequence

Tomasz Kociumaka[1]*, Jakub Radoszewski[1], and Wojciech Rytter[1,2]

[1] Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Warsaw, Poland
`[kociumaka,jrad,rytter]@mimuw.edu.pl`
[2] Faculty of Mathematics and Computer Science,
Copernicus University, Toruń, Poland

**Abstract.** Let $\Sigma$ be a finite ordered alphabet. We present polynomial-time algorithms for computing the $k$-th in the lexicographic order Lyndon word of a given length $n$ over $\Sigma$ and counting Lyndon words of length $n$ that are smaller than a given word. We also use the connections between Lyndon words and minimal de Bruijn sequences (theorem of Fredricksen and Maiorana) to develop the first polynomial time algorithm for *decoding* minimal de Bruijn sequence of any rank $n$ (it determines the position of an arbitrary word of length $n$ within the de Bruijn sequence). Our tools mostly rely on combinatorics on words and automata theory.

## 1 Introduction

We consider finite words over a finite ordered alphabet $\Sigma$. A *Lyndon word* over $\Sigma$ is a word that is strictly smaller than all its nontrivial cyclic rotations. Lyndon words have a number of combinatorial properties (see, e.g., [10]) including the famous Lyndon factorization theorem which states that every word can be uniquely written as a concatenation of a lexicographically non increasing sequence of Lyndon words (due to this theorem Lyndon words are also called prime words, see [9]). They are also related to necklaces of $n$ beads in $k$ colors, that is, equivalence classes of $k$-ary $n$-tuples under rotation [6, 7]. Lyndon words have a number of applications in the field of text algorithms, see e.g. [1–3, 12].

A *de Bruijn sequence of rank n* is a cyclic sequence of length $|\Sigma|^n$ in which every possible word of length $n$ appears as a subword exactly once. De Bruijn sequences are present in a variety of contexts, such as digital fault testing, pseudo-random number generation, and modern public-key cryptographic schemes; there are numerous algorithms for generating such sequences and their generalizations to other combinatorial structures have been investigated, see [5, 9]. Fredricksen and Maiorana [7] have shown a surprising deep connection between de Bruijn sequences and Lyndon words: the lexicographically minimal de Bruijn sequence

* Supported by Polish budget funds for science in 2013-2017 as a research project under the 'Diamond Grant' program.

over $\Sigma$ is a concatenation, in lexicographic order, of all Lyndon words over $\Sigma$ whose length is a divisor of $n$.

All Lyndon words of length at most $n$ can be generated in lexicographic order by algorithm of Fredricksen, Kessler and Maiorana (FKM) [6, 7] (another algorithm was developed by Duval in [4]). The analysis from [14] shows that the FKM algorithm generates the subsequent Lyndon words in constant amortized time. We give the first polynomial time algorithm for generating Lyndon words of arbitrary rank in lexicographic order. We also generalize the known simple formula for the number of Lyndon words of length $n$ over $\Sigma$ (see [9, 10]) by showing a polynomial time algorithm that computes the number of Lyndon words of length $n$ smaller than a given word.

For several de Bruijn sequences *decoding* algorithms exist which find the position of an arbitrary word of length $n$ in a given de Bruijn sequence in polynomial time [11, 15]. Such algorithms prove useful in certain types of position sensing applications of de Bruijn sequences [15]. We obtain the first decoding algorithm for the lexicographically minimal de Bruijn sequence by exploiting its connections with Lyndon words. We also obtain a polynomial-time algorithm for random access of symbols in this sequence and in a related sequence defined in [8]. Note that the FKM algorithm can be used to compute the subsequent symbols of the lexicographically minimal de Bruijn sequence with $\mathcal{O}(n^2)$ time delay (or even with worst-case $\mathcal{O}(1)$ time delay [13]), however it does it only *in order*.

We denote by $\mathcal{L}$ and $\mathcal{L}_n$ the set of all Lyndon words and all Lyndon words of length $n$, respectively, and define

$$Lynd(w) \;=\; \{x \in \mathcal{L}_{|w|} \;:\; x \leq w\}.$$

*Example 1.* For $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ we have $|Lynd(\mathtt{abbaba})| = 8$ since we have the following Lyndon words of length 6 smaller than `abbaba` (note that `abbaba` itself is not a Lyndon word):

`aaaaab`, `aaaabb`, `aaabab`, `aaabbb`, `aababb`, `aabbab`, `aabbbb`, `ababbb`.

Let $\mathcal{L}^{(n)} = \bigcup_{d|n} \mathcal{L}_d$. By $dB_n$ we denote the lexicographically first de Bruijn sequence of rank $n$ over the given alphabet $\Sigma$. It is the concatenation of all Lyndon words in $\mathcal{L}^{(n)}$ in lexicographic order.

*Example 2.* For $n = 6$ and binary alphabet we have the following decomposition of $dB_6$ into Lyndon words:

0 000001 000011 000101 000111 001 001011 001101 001111 01 010111 011 011111 1.

Recently a variant of de Bruijn words was introduced in [8]. Let $dB'_n$ be the concatenation in lexicographic order of Lyndon words of length $n$ over $\Sigma$. Then $dB'_n$ is a cyclic sequence containing all primitive words of length $n$.

*Example 3.* For $n = 6$ and binary alphabet we have the following decomposition of $dB'_6$:

000001 000011 000101 000111 001011 001101 001111 010111 011111.

**Our results.** We assume that $|\Sigma|$ fits in a single machine word. We present an $\mathcal{O}(n^3)$-time algorithm for computing $|Lynd(w)|$ for a word $w$ of length $n$. Using binary search this algorithm implies an $\mathcal{O}(n^4 \log |\Sigma|)$-time algorithm for computing the $k$-th Lyndon word of length $n$ (in the lexicographic order) for a given $k$. Next we show an $\mathcal{O}(n^3)$-time decoding algorithm that finds the position of an arbitrary $w \in \Sigma^n$ in $dB_n$. We also obtain $\mathcal{O}(n^4 \log |\Sigma|)$-time algorithms computing the $k$-th symbol of $dB_n$ and $dB'_n$ for a given $k$.

## 2 Preliminaries

Let $\Sigma$ be a finite ordered alphabet. By $\Sigma^*$ and $\Sigma^n$ we denote the set of all words over $\Sigma$ and the set of all such words of length $n$. If $w$ is a word then $|w|$ denotes its length, $w[i]$ its $i$-th letter (for $1 \le i \le |w|$), $w[i,j]$ its factor $w[i]w[i+1]\dots w[j]$ and $w_{(i)}$ its prefix $w[1,i]$. Additionally $w^k$ is a concatenation of $k$ copies of $w$ and $w^\infty$ is an infinite word composed of an infinite number of copies of $w$.

By $rot(w,c)$ let us denote a *cyclic rotation* of $w$ obtained by moving ($c \bmod n$) first letters of $w$ to its end (preserving the order of the letters). We say that the words $w$ and $rot(w,c)$ are *cyclically equivalent* (sometimes called *conjugates*). By $\langle w \rangle$ we denote the lexicographically minimal cyclic rotation of $w$. We say that $w$ is *primitive* if $w = u^k$ for $k \in \mathbb{Z}_+$ implies that $u = w$, otherwise $w$ is called non-primitive. We say that $\lambda \in \Sigma^*$ is a *Lyndon word* if it is primitive and $\langle \lambda \rangle = \lambda$. All cyclic rotations of a Lyndon word are different primitive words [10]. The technical proofs of the following lemmas are left for the full version.

**Lemma 4.** *Let $x, y \in \Sigma^*$. Assume $x = \langle x \rangle$ and $x \ge y$. Then $x^\infty \ge y^\infty$.*

**Lemma 5.** *For a given word $w \in \Sigma^n$ we can compute in $\mathcal{O}(n^2)$ time the lexicographically largest word $w' \in \Sigma^n$ such that $\langle w' \rangle = w' \le w$.*

## 3 Combinatorial Tools

Our basic goal is to compute $|Lynd(w)|$, that is, the number of Lyndon words in $\Sigma^n$ not exceeding $w$ ($n = |w|$). It suffices to compute $|Lynd(w)|$ for words $w$ such that $\langle w \rangle = w$. We show how to reduce it to the computation of the cardinality of the following set:

$$CS(v) = \{x \in \Sigma^{|v|} : \langle x \rangle \le v\}$$

for some prefixes $v$ of $w$. Computation of $|CS(v)|$ is also of independent interest, we apply it in the decoding scheme for minimal de Bruijn sequences.

Let us introduce the following auxiliary sets:

$$CS_\ell(v) = \{x \in \Sigma^\ell : \langle x \rangle^\infty \le v^\infty\}$$
$$CS'_\ell(v) = \{x \in \Sigma^\ell : x \text{ is primitive}, \langle x \rangle^\infty \le v^\infty\}.$$

Note that if $|x| = |v|$ then $\langle x \rangle^\infty \le v^\infty$ is simply equivalent to $x \le v$. Thus $CS(v) = CS_{|v|}(v)$.

**Observation 6.** $|Lynd(w)| = \frac{1}{n}|CS'_n(w)|$

*Proof.* Observe that $CS'_n(w)$ is the set of all primitive words of length $n$ that have a cyclic rotation not exceeding $w$. Each Lyndon word of length $n$ not exceeding $w$ corresponds to $n$ such words: all its cyclic rotations. □

**Observation 7.** $|CS_\ell(w)| = \sum_{d|\ell} |CS'_d(w)|$.

*Proof.* For a word $x$ of length $\ell$ there exists exactly one primitive word $y$ such that $y^k = x$ where $k \in \mathbb{Z}_+$. Thus:

$$CS_\ell(w) = \bigcup_{d|\ell} \left\{ y \in \Sigma^d : y \text{ is primitive}, \left\langle y^{\ell/d} \right\rangle^\infty \leq w^\infty \right\},$$

and the sum is disjoint. Now $\left\langle y^{\ell/d} \right\rangle^\infty = \langle y \rangle^\infty$ implies the requested formula. □

From Observation 7, using Möbius inversion formula, we obtain:

$$|CS'_\ell(w)| = \sum_{d|\ell} \mu(\tfrac{\ell}{d}) |CS_d(w)|.$$

**Observation 8.** *Let $w \in \Sigma^n$ satisfy $w = \langle w \rangle$. Then $CS_d(w) = CS(w_{(d)})$.*

*Proof.* If $d = n$ the equality is trivial. Assume $d < n$. Let $y \in \Sigma^d$ be a word. By Lemma 4, $w_{(d)}^\infty \leq w^\infty$, so $y^\infty \leq w_{(d)}^\infty$ implies $y^\infty \leq w^\infty$. On the other hand, if $y^\infty \leq w^\infty$, then $y \leq w_{(d)}$, so $y^\infty \leq w_{(d)}^\infty$. Applying for $y = \langle x \rangle$ we conclude that $\langle x \rangle^\infty \leq w_{(d)}^\infty$ if and only if $\langle x \rangle^\infty \leq w^\infty$, which proves the claim. □

We conclude with a simple formula for $|Lynd(w)|$ that combines the results of this section.

**Lemma 9.** *Let $w \in \Sigma^n$ satisfy $\langle w \rangle = w$. Then*

$$|Lynd(w)| = \tfrac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d}) \left| CS(w_{(d)}) \right|.$$

*Example 10.* Let $w =$ `ababbb`. We have $w_{(1)} =$ `a`, $w_{(2)} =$ `ab`, $w_{(3)} =$ `aba` and

$$CS(w_{(1)}) = \{\texttt{a}\}, \qquad CS(w_{(2)}) = \{\texttt{aa}, \texttt{ab}, \texttt{ba}\},$$
$$CS(w_{(3)}) = \{\texttt{aaa}, \texttt{aab}, \texttt{aba}, \texttt{baa}\}, \qquad |CS(w)| = 54,$$

$$|Lynd(w)| = \tfrac{1}{6} \cdot \big(\mu(1)\,|CS(w)| + \mu(2)\left|CS(w_{(3)})\right| + \mu(3)\left|CS(w_{(2)})\right| + \mu(6)\left|CS(w_{(1)})\right|\big) = \tfrac{1}{6} \cdot (54 - 4 - 3 + 1) = 8.$$

The set $Lynd(w)$ contains the following words:

`aaaaab`, `aaaabb`, `aaabab`, `aaabbb`, `aababb`, `aabbab`, `aabbbb`, `ababbb`.

## 4 Automata-Theoretic Tools: Computing *CS*

In this section we design an algorithm computing $|CS(w)|$ for a word $w \in \Sigma^n$. Note that we may assume that $\langle w \rangle = w$, since $CS(w) = CS(w')$ where $w' \in \Sigma^n$ is the largest word such that $\langle w' \rangle = w' \leq w$.

Let $\mathrm{Pref}_-(w) = \{w_{(i)}s : i \in [0, n-1], s \in \Sigma, s < w[i+1]\} \cup \{w\}$. Consider a language $L(w)$ containing words that have a factor $y \in \mathrm{Pref}_-(w)$. Equivalently, $x \in L(w)$ if there exists a factor of $x$ which is smaller than or equal to $w$, but is not a proper prefix of $w$. For a language $L \subseteq \Sigma^*$ let $\sqrt{L} = \{x : x^2 \in L\}$.

**Fact 11.** $CS(w) = \sqrt{L(w)} \cap \Sigma^n$

*Proof.* Consider a word $x \in \Sigma^n$. If $x \in CS(w)$ then $\langle x\rangle \le w$. Take $y = \langle x\rangle$, which is a factor of $x^2$. Note that $y \le w$, so some prefix of $y$ belongs to $\mathrm{Pref}_-(w)$. This prefix is a factor of $x^2$, so $x^2 \in L(w)$. Consequently, $x \in \sqrt{L(w)}$.

On the other hand, assume $x \in \sqrt{L(w)}$, so $x^2$ contains a factor $y \in \mathrm{Pref}_-(w)$. Let us fix the first occurrence of $y$ in $x^2$. Observe that $y$ can be extended to a cyclic rotation $x'$ of $x$. Note that $y \in \mathrm{Pref}_-(w)$ implies that $x' \le w$, hence $\langle x\rangle \le x' \le w$ and $x \in CS(w)$. $\square$

We construct a deterministic finite automaton $A$ recognizing $L(w)$. It has $n+1$ states: one for each proper prefix of $w$, and an auxiliary accepting state $AC$. The transitions are defined as follows: we set $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

$$\delta(w_{(i)}, c) = \begin{cases} w_{(0)} & \text{if } c > w[i+1], \\ w_{(i+1)} & \text{if } c = w[i+1] \text{ and } i \neq n-1, \\ AC & \text{otherwise.} \end{cases}$$

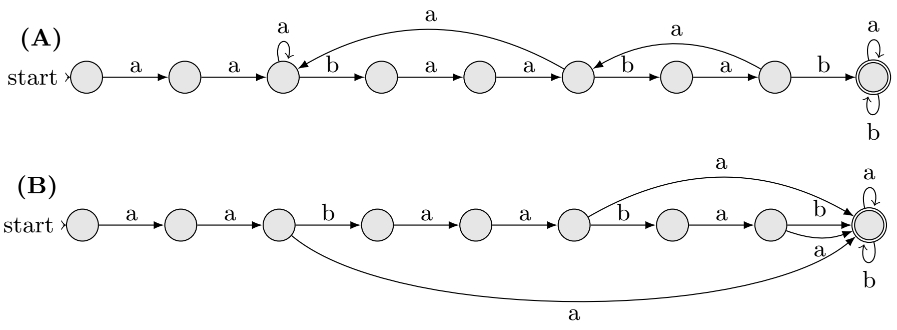

Fig. 1: Automata for a word $w = \texttt{aabaabab}$: **(A)** accepts all words containing $w$ as a factor, **(B)** accepts all words in $L(w)$, i.e. containing a factor $y \in \mathrm{Pref}_-(w)$. Missing links lead to the initial state.

The following fact implies that $L(A) = L(w)$.

**Fact 12.** *Let $x \in \Sigma^*$ and let $q$ be the state of $A$ after reading $x$. If $x \in L(w)$ then $q = AC$. Otherwise $q$ corresponds to the longest prefix of $w$ which is a suffix of $x$.*

*Proof.* The proof goes by induction on $|x|$. If $|x| = 0$ the statement is clear. Consider a word $x$ of length $|x| \geq 1$. Let $x = x'c$ where $c \in \Sigma$. If $x' \in L(w)$ then clearly $x \in L(w)$. By inductive assumption after reading $x'$ the automaton is in $AC$, and $A$ is constructed so that it stays in $AC$ once it gets there. Thus the conclusion holds in this case. From now on we assume that $x' \notin L(w)$.

Let $w_{(i)}$ be the state of $A$ after reading $x'$. If $c < w[i+1]$, clearly $x \in L(w)$ ($y = w_{(i)}c \in \mathrm{Pref}_{-}(w)$), and the automaton proceeds to $AC$ as desired. Similarly, it behaves correctly if $i = n-1$ and $c = w[i+1]$. Consequently we may assume that $c \geq w[i+1]$ and that $w$ is not a suffix of $x$.

Take any $j$ such that $w_{(j)}$ is a suffix of $x'$ (possibly empty). Note that then $w_{(j)}$ is a border of $w_{(i)}$. Consequently $w_{(j)}w[i+1,n]w_{(i-j)}$ is a cyclic rotation of $w$, so $w_{(j)}w[i+1,n]w_{(i-j)} \geq \langle w \rangle = w = w_{(j)}w[j+1,n]$, hence $c \geq w[i+1] \geq w[j+1]$. This implies that $w_{(j)}c$ could be a prefix of $w$ only if $c = w[i+1] = w[j+1]$. In particular, $A$ indeed shifts to the longest prefix of $w$ being a suffix of $x$. Now we only need to prove that $x \notin L(w)$. For a proof by contradiction, choose a factor $y$ of $x$ such that $y \in \mathrm{Pref}_{-}(w)$ and $|y|$ is minimal. Note that $y$ is a suffix of $x$ (since $x' \notin L(w)$). We have $y = w_{(j)}c$ for some $j \leq n-1$ and $c < w[j+1]$. As we have already noticed, such a word cannot be a suffix of $x$. □

We say that an automaton with the set of states $Q$ is *sparse* if the underlying directed graph has $\mathcal{O}(|Q|)$ edges counting parallel edges as one. Note that the transitions from any state $q$ of $A$ lead to at most 3 distinct states, so $A$ is sparse.

The following corollary summarizes the construction of $A$.

**Corollary 13.** *Let $w \in \Sigma^n$ satisfy $\langle w \rangle = w$. One can construct a sparse automaton $A$ with $\mathcal{O}(n)$ states recognizing $L(w)$.*

For a deterministic automaton $A = (Q, q_0, F, \delta)$ and $q, q' \in Q$ let us define $L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}$. Then $L_A(q, q')$ is recognized by $(Q, q, \{q'\}, \delta)$. Note that $L(A) = \bigcup_{q \in F} L_A(q_0, q)$ where the sum is disjoint. The proof of the following lemma is based on matrix multiplication, we omit it in this version.

**Lemma 14.** *Let $A = (Q, q_0, F, \delta)$ be a deterministic automaton with $n$ states, and let $m \in \mathbb{Z}_{\geq 0}$.*

*(a) In poly$(n+m)$ time, one can compute $|L_A(q, q') \cap \Sigma^k|$ for all $q, q' \in Q$ and $k \leq m$.*
*(b) If $A$ is sparse, it takes $\mathcal{O}(m^2 n)$ time to compute all values $|L_A(q, q') \cap \Sigma^k|$, $k \leq m$, for a fixed state $q$ or $q'$.*

**Observation 15.** *Let $A = (Q, q_0, F, \delta)$ be a deterministic automaton. Then*

$$\sqrt{L(A)} = \bigcup_{q \in Q, q' \in F} L_A(q_0, q) \cap L_A(q, q')$$

*and the sum is disjoint.*

*Proof.* Consider a word $x \in \Sigma^*$. Note that $x \in L_A(q_0, q) \cap L_A(q, q')$ for $q = \delta(q_0, x)$ and $q' = \delta(q, x)$ and no other pair of states $q, q'$. Clearly $x \in \sqrt{L(A)}$ if and only if $q' \in F$. □

**Corollary 16.** *If $w \in \Sigma^n$ satisfies $\langle w \rangle = w$, then one can compute $|CS(w)|$ in poly$(n)$ time.*

*Proof.* We construct the automaton $A$ with $L(A) = L(w)$ as in Corollary 13. By Fact 11, it suffices to compute $|\sqrt{L(A)} \cap \Sigma^n|$. Observation 15 reduces this to computing $|L_A(w_{(0)}, q) \cap L_A(q, AC) \cap \Sigma^n|$ for all states $q$ of $A$. One can construct an automaton with $\mathcal{O}(n^2)$ states representing pairs of states of $A$ that recognizes $L_A(w_{(0)}, q) \cap L_A(q, AC)$. Now it is enough to apply Lemma 14(a) to determine $|L_A(w_{(0)}, q) \cap L_A(q, AC) \cap \Sigma^n|$. □

Corollary 16 already gives a polynomial-time algorithm to compute $|CS(w)|$, however, the approach it takes is rather inefficient. Below, we give an algorithm working with a more reasonable $\mathcal{O}(n^3)$ bound for the running time, which exploits the structure of both the automaton $A$ and the language $L(w)$.

**Lemma 17.** *If $w \in \Sigma^n$ satisfies $\langle w \rangle = w$, then one can compute $|CS(w)|$ in $\mathcal{O}(n^3)$ time.*

*Proof.* As before we apply Fact 11 with Corollary 13 and actually compute $|\{x \in \Sigma^n : x^2 \in L(A)\}|$. If $x \in L(A)$, then obviously $x^2 \in L(A)$. Moreover, Lemma 14(b) lets us compute $|\Sigma^n \cap L(A)|$ in $\mathcal{O}(n^3)$ time. Thus, it suffices to count $x \in \Sigma^n$ such that $x^2 \in L(A)$ but $x \notin L(A)$. This is done based on the following claim, see Fig. 2.

*Claim.* Assume $|x| = n$ and $x^2 \in L(A)$ but $x \notin L(A)$. Then there is a unique decomposition $x = x_1x_2x_3$ such that $x_1, x_3 \neq \varepsilon$, $x_3x_1 \in \text{Pref}_-(w)$ and $x_1x_2 \in L_A(w_{(0)}, w_{(0)})$.

*Proof (of the claim).* Let $va$ (for $v \in \Sigma^*, a \in \Sigma$) be the shortest prefix of $x^2$ which belongs to $L(A)$. Let $w_{(i)} = \delta(w_{(0)}, v)$ be the state of $A$ after reading $v$. Also, let $u$ be the prefix of $v$ of length $|v| - i$. The structure of the automaton implies that $\delta(w_{(0)}, u) = w_{(0)}$, actually $u$ is the longest prefix of $x^2$ which belongs to $L_A(w_{(0)}, w_{(0)})$. Note that $v = uw_{(i)}$ and $w_{(i)}a \in \text{Pref}_-(w)$, so $x \notin L(A)$ implies $|u| < n \leq |v|$. We set the decomposition so that $x_1x_2 = u$ and $x_3x_1 = w_{(i)}a$. Uniqueness follows from deterministic behaviour of the automaton. □

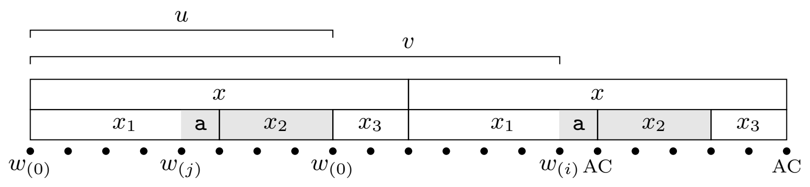

Fig. 2: Illustration of the claim. Both lines represent different factorizations of the same word $x^2$. Black circles represent states of the automaton. Only shaded letters are not necessarily uniquely determined by $|x_3|$ and $|x_1|$ for a fixed $w$.

The algorithm considers all $n^2$ choices of $|x_1|$ and $|x_3|$, and counts the number of $x$'s conditioned on these values. Let $x_1 = x_1'a$ where $x_1' \in \Sigma^*$, $a \in \Sigma$. Note that

$x_3x_1 = x_3x'_1a \in \text{Pref}_-(w)$, so $x_3x'_1$ is a prefix $w_{(i)}$ of $w$ and $\delta(w_{(i)}, a) = AC$. Hence, $i$ is uniquely determined by $|x_1|$ and $|x_3|$. In particular $x_3$ and $x'_1$ are uniquely determined, the latter lets us compute $w_{(j)} = \delta(w_{(0)}, x'_1)$. If we obtain $\delta(w_{(0)}, x'_1) = AC$, we would have $x_1 \in \mathcal{L}(w)$, a contradiction, which implies that no $x$'s for our choice of $|x_1|$ and $|x_3|$ exist.

We need to count $ax_2$ such that $|x_2| = n - i - 1$, $ax_2 \in L_A(w_{(j)}, w_{(0)})$ and $\delta(w_{(i)}, a) = AC$. Note that $\delta(w_{(j)}, a) \in \{w_{(0)}, w_{(j+1)}\}$, since $\delta(w_{(j)}, a) = AC$ would imply that $ax_2 \in L_A(w_{(j)}, AC)$ rather than $ax_2 \in L_A(w_{(j)}, w_{(0)})$. Thus the number of words $ax_2$ is equal to $|L(q, w_{(0)}) \cap \Sigma^{n-i-1}|$ summed for $q \in \{w_{(0)}, w_{(j+1)}\}$ taken with coefficients $|\{a \in \Sigma : \delta(w_{(j)}, a) = q \wedge \delta(w_{(i)}, a) = AC\}|$. Lemma 14(b) lets us compute all the values of the type $|L(q, w_{(0)}) \cap \Sigma^{n-i-1}|$ that we may need altogether in $\mathcal{O}(n^3)$ time. The coefficient can be computed for any $j$, $i$ and $q$ in $\mathcal{O}(n)$ time, and in total we need to sum $\mathcal{O}(n^2)$ integers fitting into $\mathcal{O}(n)$ machine words each. This concludes the $\mathcal{O}(n^3)$-time algorithm. □

**Lemma 18.** *For an arbitrary word $w \in \Sigma^n$ one can compute $|CS(w)|$ in $\mathcal{O}(n^3)$ time.*

*Proof.* Let $w' \in \Sigma^n$ be the largest word such that $\langle w' \rangle = w' \leq w$. Note that $CS(w') = CS(w)$. By Lemma 5 we can compute $w'$ in $\mathcal{O}(n^2)$ time. □

## 5 Ranking Lyndon words and De Bruijn Sequences

**Fact 19.** *Let $\alpha > 1$ be a real number. Then $\sum_{d|n} d^\alpha = \mathcal{O}(n^\alpha)$.*

*Proof.* Recall that for $\alpha > 1$ we have $\sum_{n=1}^{\infty} \frac{1}{n^\alpha} = \mathcal{O}(1)$. Consequently

$$\sum_{d|n} d^\alpha = \sum_{d|n} \left(\tfrac{n}{d}\right)^\alpha \leq \sum_{d=1}^{\infty} \left(\tfrac{n}{d}\right)^\alpha = n^\alpha \sum_{d=1}^{\infty} \tfrac{1}{d^\alpha} = \mathcal{O}(n^\alpha). \qquad \square$$

**Theorem 20.** *We can compute $|Lynd(w)|$ in $\mathcal{O}(n^3)$ time.*

*Proof.* We use the formula given by Lemma 9 and the algorithm of Lemma 18. The time complexity is $\mathcal{O}(\sum_{d|n} d^3)$ which, by Fact 19, reduces to $\mathcal{O}(n^3)$. □

**Theorem 21.** *The $k$-th Lyndon word of length $n$ can be found in $\mathcal{O}(n^4 \log|\Sigma|)$ time.*

*Proof.* By definition we look for the smallest $w \in \Sigma^n$ such that $|Lynd(w)| \geq k$. We binary search $\Sigma^n$ with respect to the lexicographic order, using the algorithm of Theorem 20 to check whether $|Lynd(w)| \geq k$. The size of the search space is $|\Sigma|^n$, which gives an additional $n \log|\Sigma|$-time factor. □

The proof of the theorem of Fredricksen and Maiorana [7] is constructive, i.e. for any word $w$ of length $n$ it shows the concatenation of a constant number of consecutive Lyndon words of length dividing $n$ that contain $w$. This, together with the following lemma which relates $dB_n$ to $CS$, lets us compute the exact position where $w$ occurs in $dB_n$. Recall that $\mathcal{L}^{(n)}$ is the set of Lyndon words whose length is a divisor of $n$.

**Lemma 22.** *Let $w \in \Sigma^n$ and $\mathcal{L}(w) = \{\lambda \in \mathcal{L}^{(n)} : \lambda^\infty \le w^\infty\}$. Then the concatenation, in lexicographic order, of words $\lambda \in \mathcal{L}(w)$ forms a prefix of $dB_n$ and its length, $\sum_{\lambda\in\mathcal{L}(w)} |\lambda|$, is equal to $|CS(w)|$. Moreover, if $w = \lambda^d$ for some $\lambda \in \mathcal{L}$ and $d \in \mathbb{Z}_+$, then $\lambda$ is the lexicographically largest element of $\mathcal{L}(w)$.*

*Proof.* First, observe that by Lemma 4 the lexicographic order on $\mathcal{L}$, the set of all Lyndon words, coincides with the lexicographic order of the infinite powers of these words. In particular, this remains true in $\mathcal{L}^{(n)}$ and shows that concatenation of elements of $\mathcal{L}(w)$ indeed forms a prefix of $dB_n$, and that if $w = \lambda^d$, then $\lambda$ is the lexicographically largest element of $\mathcal{L}(w)$.

It remains to show that $\sum_{\lambda\in\mathcal{L}(w)} |\lambda| = |CS(w)|$. We shall build a mapping $\phi : \Sigma^n \to \mathcal{L}^{(n)}$ such that $|\phi^{-1}(\lambda)| = |\lambda|$ and $\langle x\rangle \le w$ if and only if $\phi(x) \in \mathcal{L}(w)$.

Let $x \in \Sigma^n$. There is a unique primitive word $y$ and a positive integer $k$ such that $x = y^k$. We set $\phi(x) = \langle y\rangle$, note that it indeed belongs to $\mathcal{L}^{(n)}$. Moreover, to each Lyndon word $\lambda$ of length $d \mid n$ we have assigned $v^{\frac{n}{d}}$ for each cyclic rotation $v$ of $\lambda$. Also, $\langle x\rangle = \langle y\rangle^{\frac{n}{d}}$, so $\langle x\rangle \le w$ if and only if $\phi(x)^{\frac{n}{d}} \le w$, i.e. $\phi(x)^\infty \le w^\infty$, i.e. $\phi(x) \in \mathcal{L}(w)$. □

**Theorem 23.** *Given a word $w \in \Sigma^n$, its position in the de Bruijn sequence $dB_n$ can be found in $\mathcal{O}(n^3)$ time.*

*Proof.* Let $\lambda_1 < \lambda_2 < \ldots < \lambda_p$ be all Lyndon words in $\mathcal{L}^{(n)}$ (we have $\lambda_1\lambda_2\ldots\lambda_p = dB_n$). The proof of theorem of Fredricksen and Maiorana [7, 9] describes the location of $w$ in $dB_n$ which can be stated succinctly as follows.

*Claim ([7, 9]).* Assume that $w = (\alpha\beta)^d$, where $d \in \mathbb{Z}_+$ and $\beta\alpha = \lambda_k \in \mathcal{L}^{(n)}$. Denote $a = \min\Sigma$ and $z = \max\Sigma$.

(a) If $w = z^i a^{n-i}$ for $i \ge 1$, then $w$ occurs at position $|\Sigma|^n - i$.
(b) If $\alpha \ne z^{|\alpha|}$ then $w$ is a factor of $\lambda_k\lambda_{k+1}$.
(c) If $\alpha = z^{|\alpha|}$ and $d > 1$ then $w$ is a factor of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.
(d) If $\alpha = z^{|\alpha|}$ and $d = 1$ then $w$ is a factor of $\lambda_{k'-1}\lambda_{k'}\lambda_{k'+1}$, where $\lambda_{k'}$ is the largest $\lambda \in \mathcal{L}^{(n)}$ such that $\lambda < \beta$.

In case (a) it is easy to locate $w$ in $dB_n$, we omit it from further considerations. Observe that $\lambda_k$ can be retrieved as the primitive root of $\langle w\rangle$. Also note that, by Lemma 4, $\lambda_{k'}$ is the primitive root of the largest $w' \in \Sigma^n$ such that $w' = \langle w'\rangle$ and $w' < \beta a^{|\alpha|}$, and thus it can be computed in $\mathcal{O}(n^2)$ time using Lemma 5.

Once we know $\lambda_{k'}$ and $\lambda_k$, depending on the case, we need to find the successor in $\mathcal{L}^{(n)}$ and possibly the predecessor in $\mathcal{L}^{(n)}$ of one of them. For any $\lambda \in \mathcal{L}^{(n)}$ the successor in $\mathcal{L}^{(n)}$ can be generated by iterating a single step of the FKM algorithm at most $(n-1)/2$ times [6], i.e. in $\mathcal{O}(n^2)$ time. For the predecessor in $\mathcal{L}^{(n)}$, a version of the FKM algorithm that visits the Lyndon words in reverse lexicographic order can be used [9], it also takes $\mathcal{O}(n^2)$ time to find the predecessor. In all cases we obtain in $\mathcal{O}(n^2)$ time the Lyndon words whose concatenation contains $w$.

Then we perform a pattern matching for $w$ in the concatenation. This gives us a relative position of $w$ in $dB_n$ with respect to the position of the canonical occurrence of $\lambda_k$ or $\lambda_{k'}$ in $dB_n$. Lemma 22 proves that such an occurrence of $\lambda \in \mathcal{L}^{(n)}$ *ends* at position $|CS(\lambda^{\lceil \frac{n}{|\lambda|} \rceil})|$, which can be computed in $\mathcal{O}(n^3)$ time by Lemma 18. Applied to $\lambda_k$ or $\lambda_{k'}$ this concludes the proof. □

To compute the $k$-th symbol of $dB_n$ we have to locate the Lyndon word from $\mathcal{L}^{(n)}$ containing the $k$-th position of $dB_n$. We apply binary search as in Theorem 21. The $k$-th symbol of $dB'_n$ is much easier to find due to a simpler structure of the sequence. This gives a rough idea of the proof of the following theorem. We omit the full proof in this version of the paper.

**Theorem 24.** *Given integers $n$ and $k$, the $k$-th symbol of $dB_n$ and $dB'_n$ can be computed in $\mathcal{O}(n^4 \log |\Sigma|)$ time.*

## References

1. S. Bonomo, S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. Suffixes, conjugates and Lyndon words. In M.-P. Béal and O. Carton, editors, *Developments in Language Theory*, volume 7907 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 2013.
2. M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, doi: 10.1016/j.tcs.2013.11.018, 2013.
3. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.
4. J.-P. Duval. Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. *Theor. Comput. Sci.*, 60:255–283, 1988.
5. R. G. F. Chung, P. Diaconis. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110:43–59, 1992.
6. H. Fredricksen and I. J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Mathematics*, 61(2-3):181–188, 1986.
7. H. Fredricksen and J. Maiorana. Necklaces of beads in $k$ colors and $k$-ary de Bruijn sequences. *Discrete Mathematics*, 23(3):207–210, 1978.
8. Y. Hin Au. Shortest sequences containing primitive words and powers. *ArXiv e-prints*, Apr. 2009.
9. D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2.* Addison-Wesley, 2005.
10. M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA., U.S.A., 1983.
11. C. J. Mitchell, T. Etzion, and K. G. Paterson. A method for constructing decodable de Bruijn sequences. *IEEE Transactions on Information Theory*, 42(5):1472–1478, 1996.
12. M. Mucha. Lyndon words and short superstrings. In S. Khanna, editor, *SODA*, pages 958–972. SIAM, 2013.
13. J. Radoszewski. Generation of lexicographically minimal de Bruijn sequences with prime words. Master's thesis, University of Warsaw, 2008 (in Polish).
14. F. Ruskey, C. D. Savage, and T. M. Y. Wang. Generating necklaces. *J. Algorithms*, 13(3):414–430, 1992.
15. J. Tuliani. De Bruijn sequences with efficient decoding algorithms. *Discrete Mathematics*, 226(1-3):313–336, 2001.