SIAM J. DISCRETE MATH.
Vol. 30, No. 4, pp. 2027–2046

EFFICIENT RANKING OF LYNDON WORDS AND DECODING LEXICOGRAPHICALLY MINIMAL DE BRUIJN SEQUENCE*

TOMASZ KOCIUMAKA†, JAKUB RADOSZEWSKI†, AND WOJCIECH RYTTER†

Abstract. We give efficient algorithms for ranking Lyndon words of length n over an alphabet of size σ. The rank of a Lyndon word is its position in the sequence of lexicographically ordered Lyndon words of the same length. The outputs are integers of exponential size, and complexity of arithmetic operations on such large integers cannot be ignored. Our model of computations is the word RAM, in which basic arithmetic operations on (large) numbers of size at most σ^n take $\mathcal{O}(n)$ time. Our algorithm for ranking Lyndon words makes $O(n^2)$ arithmetic operations (this would imply directly cubic time on word RAM). However, using an algebraic approach we are able to reduce the total time complexity on word RAM to $O(n^2 \log \sigma)$. We also present an $O(n^3 \log^2 \sigma)$-time algorithm that generates the Lyndon word of a given length and rank in lexicographic order. Finally we use the connections between Lyndon words and lexicographically minimal de Bruijn sequences (a theorem of Fredricksen and Maiorana) to develop the first polynomial-time algorithm for decoding the minimal de Bruijn sequence of any rank n (it determines the position of a given word of length n within the de Bruijn sequence).

Key words. Lyndon word, de Bruijn sequence, necklaces, deterministic finite automaton, efficient algorithm

AMS subject classifications. 68R15, 68W32, 68R05, 68Q45

DOI. 10.1137/15M1043248

1. Introduction.

We consider finite words over an ordered alphabet Σ of size $\sigma = |\Sigma|$. A *Lyndon word* [24, 5] over Σ is a word that is strictly smaller in the lexicographic order than all its nontrivial cyclic rotations. For example, for $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ where $\mathtt{a} < \mathtt{b}$, the word `aababb` is a Lyndon word, as it is smaller than its cyclic rotations: `ababba`, `babbaa`, `abbaab`, `bbaaba`, `baabab`. On the other hand, the word `abaab` is not a Lyndon word since its cyclic rotation `aabab` is smaller than it. Also the word `aabaab` is not a Lyndon word, as its cyclic rotation by three letters is equal to it. Lyndon words have a number of combinatorial properties (see, e.g., [23]), including the famous Lyndon factorization theorem [5], which states that every word can be uniquely written as a concatenation of a lexicographically nonincreasing sequence of Lyndon words (due to this theorem, Lyndon words are also called prime words; see [20]). They are also related to *necklaces* of n beads in k colors, that is, equivalence classes of k-ary n-tuples under rotation [15, 16]. In particular, a necklace can be identified with the lexicographically minimal tuple in its class, and thus it is often defined as a word of length n over an alphabet of size k that is smaller than or equal to all its cyclic rotations (or, equivalently, as a power of a Lyndon word of

*Received by the editors October 9, 2015; accepted for publication (in revised form) August 30, 2016; published electronically October 27, 2016. A preliminary version of this paper appeared in *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching* (CPM 2014), Lecture Notes in Comput. Sci. 8486, Springer, Cham, 2014, pp. 202–211. This extended version has complexities reduced by a $(\log \sigma)/n$ factor in the case of the word-RAM model.

http://www.siam.org/journals/sidma/30-4/M104324.html

Funding: The work of the first author was supported by Polish budget funds for science in 2013–2017 as a research project under the "Diamond Grant" program, grant DI2012 01794 of the Ministry of Science and Higher Education, Republic of Poland. The work of the third author was supported by the Polish National Science Center, grant 2014/13/B/ST6/00770.

†Institute of Informatics, University of Warsaw, Warsaw, Poland (kociumaka@mimuw.edu.pl, jrad@mimuw.edu.pl, rytter@mimuw.edu.pl). The second author is a Newton Fellow at King's College London.

a length that divides n). Lyndon words and necklaces have numerous applications in the field of text algorithms; see, e.g., [3, 8, 9, 27].

A *de Bruijn sequence of rank* n [10] is a cyclic sequence of length σ^n in which every possible word of length n occurs as a factor exactly once. For example, for $\Sigma = \{0, 1\}$ the following two sequences of length 16 are de Bruijn sequences of rank 4:

$$0000100110101111 \quad \text{and} \quad 0011110110010100.$$

De Bruijn sequences are present in a variety of contexts, such as digital fault testing, pseudorandom 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 [6, 20]. Fredricksen [13] and Fredricksen and Maiorana [16] have shown a surprising deep connection between de Bruijn sequences and Lyndon words: the lexicographically minimal de Bruijn sequence over Σ is a concatenation, in the lexicographic order, of all Lyndon words over Σ whose length is a divisor of n. For example, for $n = 6$ and the binary alphabet we have the following decomposition of the minimal de Bruijn sequence into Lyndon words:

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

Problem definitions and previous results. 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

$$\mathit{LynRank}(w) \ = \ |\{x \in \mathcal{L}_{|w|} \ : \ x \le w\}|.$$

The problem of ranking Lyndon words can be defined as follows.

Problem 1. Ranking Lyndon words

Given a Lyndon word λ, compute $\mathit{LynRank}(\lambda)$.

Example 1. For $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ we have $\mathit{LynRank}(\mathtt{ababbb}) = 8$ since there are eight Lyndon words of length 6 that are not greater than `ababbb`:

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

What was previously known is that all Lyndon words of length at most n can be generated in lexicographic order. The first solution is due to Fredricksen and Kessler [15] and Fredricksen and Maiorana [16]; later Duval developed an alternative algorithm [12]. The analysis by Ruskey, Savage, and Wang [31] shows that the Fredricksen–Kessler–Maiorana (FKM) algorithm generates the subsequent Lyndon words in constant amortized time; Berstel and Pocchiola [2] achieved an analogous result for Duval's algorithm. A different constant-amortized-time solution, based on recursion, was given by Cattell et al. [4]. However, there was no polynomial-time algorithm to generate a Lyndon word of an arbitrary rank or for ranking Lyndon words. Ruskey stated finding such an algorithm as a research problem in his book [30].

An intimately related task, of ranking and unranking *necklaces*, was explicitly stated as open by Martínez and Molinero [25]. As far as obtaining polynomial-time solutions is concerned, one can easily show equivalence of this problem with its counterpart for Lyndon words. Listing necklaces also resembles listing Lyndon words: two of the previously mentioned algorithms (FKM and that of Cattell et al.) can be

used for both tasks. Variants of the listing problem have also been considered, e.g., generating binary necklaces with a given number of zeroes and ones [32].

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 Σ. It is the concatenation of all Lyndon words in $\mathcal{L}^{(n)}$ in lexicographic order [13, 16]. For a word w of length n over Σ, by $\mathit{occ\text{-}pos}(w, dB_n)$ we denote the (1-based) position of its occurrence in dB_n. The problem of decoding the minimal de Bruijn sequence can be stated as follows.

Problem 2. Decoding minimal de Bruijn sequence

Given a word w over Σ^n, compute $\mathit{occ\text{-}pos}(w, dB_n)$.

Example 2. For $\Sigma = \{0, 1\}$ we have $dB_4 = 0000100110101111$. For this sequence,

$$\mathit{occ\text{-}pos}(1001, dB_4) = 5, \quad \mathit{occ\text{-}pos}(0101, dB_4) = 10, \quad \mathit{occ\text{-}pos}(1100, dB_4) = 15.$$

For several types of de Bruijn sequences, there exist polynomial-time decoding algorithms [26, 35]. They find the position of an arbitrary word of length n in a specific de Bruijn sequence, which proves useful in certain types of position sensing applications of de Bruijn sequences (see [35]). Nevertheless, no polynomial-time decoding algorithm for the lexicographically minimal de Bruijn sequence was known prior to our contribution. Note that the FKM algorithm can be used to compute the subsequent symbols of the lexicographically minimal de Bruijn sequence with worst-case $\mathcal{O}(n^2)$ delay [15] and amortized $\mathcal{O}(1)$ delay [31]. Alternative solutions achieve $\mathcal{O}(n)$ [14, 29] or even $\mathcal{O}(1)$ [28] worst-case delay. All these solutions only allow one to generate characters of dB_n *in order*, though.

Our model of computations. Our algorithms work in the *word-RAM* model; see [19]. In this model, we assume that σ and n fit in a single machine word; in other words, a single machine word has at least $\max(\log \sigma, \log n)$ bits and simple arithmetic operations on small numbers (i.e., numbers which fit in a constant number of machine words) are performed in constant time. Basic arithmetic operations (addition, subtraction, multiplication by a small number) on numbers of size at most σ^n take $\mathcal{O}(\frac{\log \sigma^n}{\max(\log n, \log \sigma)}) = \mathcal{O}(n)$ time.

Another model of computation is the *unit-cost RAM*, where each arithmetic operation takes constant time. This model is rather unrealistic if we deal with large numbers. However, it is a useful intermediate abstraction.

Our results. We present an $\mathcal{O}(n^2 \log \sigma)$-time solution for finding the rank of a Lyndon word (Problem 1). The algorithm actually computes $\mathit{LynRank}(w)$ for arbitrary w that are not necessarily Lyndon words. Using binary search, it yields an $\mathcal{O}(n^3 \log^2 \sigma)$-time algorithm for computing the kth Lyndon word of length n (in the lexicographic order) for a given k. Next, we show an $\mathcal{O}(n^2 \log \sigma)$-time solution for decoding minimal de Bruijn sequence dB_n (Problem 2). We also develop an $\mathcal{O}(n^3 \log^2 \sigma)$-time algorithm computing the kth symbol of dB_n for a given k. Additionally, we obtain analogous results for a variant dB'_n of the minimal de Bruijn sequence, recently introduced by Au [1], in which all factors of length n are primitive and every primitive word of length n occurs exactly once. All these algorithms work in the word-RAM model. In the unit-cost RAM, the time complexities reduce by a factor of $\log \sigma$.

Related work. A preliminary version of this paper appeared as [21]. At about the same time, similar results were published by Kopparty, Kumar, and Saks [22]. The

work in these two papers was done independently. The papers provide polynomial-time algorithms for ranking Lyndon words and necklaces, respectively, and these two problems can be easily reduced to each other. The authors in [22] put the results in a broader context and have some additional applications (indexing irreducible polynomials and explicit constructions of certain algebraic objects). On the other hand, we exercised more care in designing the algorithm to obtain a better polynomial running time. In particular, [21] contained an $\mathcal{O}(n^3)$-time algorithm for ranking Lyndon words in the word-RAM model, which works in $\mathcal{O}(n^2)$ time in the unit-cost RAM. We also obtained a cleaner approach to alphabets of size more than 2. An alternative $\mathcal{O}(n^2)$-time algorithm in the unit-cost RAM model was recently designed by Sawada and Williams [33].

Structure of the paper. Sections 2 to 5 (and 7) contain a full version of the paper [21]. Section 2 defines the notions of self-minimal words (necklaces) and Lyndon words and lists a number of their properties. In section 3 we use combinatorial tools to obtain a formula for $\mathit{LynRank}(w)$ in the case that w is self-minimal. The next three sections are devoted to efficient computation of the main ingredient of this formula. In section 4 we show that it is sufficient to count specific walks in an auxiliary automaton. Then in sections 5 and 6 we show efficient implementations of this technique under unit-cost RAM and word-RAM models, respectively. In section 7, we apply ranking of Lyndon words to obtain efficient decoding of the minimal de Bruijn sequence.

2. Preliminaries. Let Σ be an ordered alphabet of size $\sigma = |\Sigma|$. By Σ^* and Σ^n we denote the set of all finite words over Σ and the set of all such words of length n. The empty word is denoted as ε. If w is a word, then $|w|$ denotes its length, $w[i]$ its ith letter (for $1 \le i \le |w|$), $w[i,j]$ its factor $w[i]w[i+1]\cdots w[j]$, and $w_{(i)}$ its prefix $w[1,i]$. A suffix of w is a word of the form $w[i,n]$. A prefix or a suffix is called proper if it is shorter than w. By w^k we denote a concatenation of k copies of w. Any two words can be compared in the lexicographic order: u is smaller than v if u is a proper prefix of v or if the letter following the longest common prefix of u and v in u is smaller than in v.

By $\mathit{rot}(w,c)$ let us denote a *cyclic rotation* of w obtained by moving $c \bmod |w|$ first letters of w to its end (preserving the order of the letters). We say that the words w and $\mathit{rot}(w,c)$ are *cyclically equivalent* (sometimes called *conjugates*). By $\langle w \rangle$ we denote the lexicographically minimal cyclic rotation of w. A word w is called *self-minimal* (alternatively, a *necklace*) if $\langle w \rangle = w$. The following fact gives a simple property of self-minimal words.

FACT 3. *If $w \in \Sigma^n$ is self-minimal and $d \mid n$, then $(w_{(d)})^{n/d} \le w$.*

Proof. Assume to the contrary that $(w_{(d)})^{n/d} > w$. Let k be the index of the first letter where these two words differ. Then of course $(w_{(d)})^{n/d}[k] > w[k]$. Let j be an integer defined as $jd+1 \le k \le (j+1)d$. Then $w_{(d)} > w[jd+1,(j+1)d]$. Hence, $w > \mathit{rot}(w,jd)$, a contradiction. ∎

In the ranking algorithms that we design below, we make an assumption that the input word is self-minimal. Consequently, we often need to replace a given arbitrary word w with the lexicographically largest self-minimal word w' (of the same length) not exceeding w. In the construction of this routine, we use the following auxiliary facts.

FACT 4 (see [11]). *For a given word $x \in \Sigma^n$, $\langle x \rangle$ can be computed in $\mathcal{O}(n)$ time.*

FACT 5. *Let x and x', $x < x'$, be words of length n with the longest common prefix of length p. If x is self-minimal and $x'[i] = \max\Sigma$ for each $i > p+1$, then x' is self-minimal.*

Proof. First, note that $x[1] < z$, where $z = \max\Sigma$. Indeed, if $x[1] = z$, then we would have $x = z^n$ by self-minimality of x. However, this contradicts $x' > x$. Now, observe that $x'[1] = z$ implies $p = 0$ and consequently $x'[i] = z$ for $i > 1$. Thus, $x' = z^n$ is trivially self-minimal. Hence, from now on we may assume that $x'[1] < z$.

Since $x'[i] = z$ for $i > p+1$, this means that $x' < z < rot(x', i)$ for $i \in \{p+1, \ldots, n-1\}$. Thus, it suffices to show that $x' \leq rot(x', i)$ for $i \in \{1, \ldots, p\}$. For a proof by contradiction, suppose that $x' > rot(x', i)$. Consequently,

$$x' > rot(x', i) > rot(x, i) \geq x$$

since for $i \in \{1, \ldots, p\}$ we have $rot(x, i) < rot(x', i)$ with the longest common prefix of length exactly $p - i$. However, the obtained sequence of inequalities proves that $rot(x', i)$ and $rot(x, i)$ have a common prefix of length at least p due to such a common prefix of x and x'. The contradiction concludes the proof. ∎

We are now ready to implement the announced procedure.

LEMMA 6. *For a given word $w \in \Sigma^n$ we can compute in $\mathcal{O}(n^2)$ time the lexicographically largest self-minimal word $w' \in \Sigma^n$ such that $w' \leq w$.*

Proof. Fact 4 lets us check whether w is self-minimal. If so, we simply return $w' = w$. Consequently, we may assume that the sought word w' is strictly smaller than w. Assume the longest common prefix of w and w' is $w_{(k-1)}$ for some $k \leq n$. Then $b = w[k] > w'[k]$, so in particular $w[k] \neq \min\Sigma$ and one can choose $b' \in \Sigma$ as the letter preceding b. Additionally, let $z = \max\Sigma$. Consider a word

$$w'' = w_{(k-1)} b' z^{n-k}.$$

Note that $w' \leq w'' < w$. If $w' < w''$, then Fact 5 applied for $x = w'$ and $x' = w''$ would show that $\langle w'' \rangle = w''$, and this would contradict the definition of w'. Hence, $w' = w''$.

Consequently, it suffices to consider w and, for each $k \in \{1, \ldots, n\}$ such that $w[k] \neq \min\Sigma$, a word $w_{(k-1)} b' z^{n-k}$ where b' is the letter preceding $w[k]$ in Σ. Since w' is guaranteed to be one of the considered words, it suffices to output the largest of these candidates for which $\langle w' \rangle = w'$. This procedure can be implemented in $\mathcal{O}(n^2)$ time using Fact 4. ∎

The shortest word u such that $w = u^k$ for some positive integer k is called the *primitive root* of w. We say that w is *primitive* if its primitive root is w itself. Otherwise, w is called nonprimitive. The primitive root of a word of length n can be computed in $\mathcal{O}(n)$ time; see [7].

We say that $\lambda \in \Sigma^*$ is a *Lyndon word* if it is primitive and self-minimal. Equivalent definitions are that a Lyndon word is (strictly) smaller than all its suffixes or all its cyclic rotations. All cyclic rotations of a Lyndon word are different primitive words. Moreover, every self-minimal word can be expressed in a unique way as λ^k for some Lyndon word λ; see [23]. Below we show an additional property of Lyndon words that will be useful in section 7.

FACT 7. *Let $\lambda_1, \lambda_2 \in \mathcal{L}^{(n)}$.*

(i) *It is not possible that $\lambda_1 < \lambda_2 \leq \lambda_1^{n/|\lambda_1|}$.*

(ii) *If* $\lambda_1 < \lambda_2$, *then* $\lambda_1^{n/|\lambda_1|} < \lambda_2^{n/|\lambda_2|}$.

Proof. (i) The inequalities imply that λ_1 is a proper prefix of λ_2. Let $\lambda_2 = \lambda_1^k x$, where $k \geq 1$ is an integer and λ_1 is not a prefix of x. We have

$$\lambda_2 \leq \lambda_1^{n/|\lambda_1|} \implies x \leq \lambda_1^{n/|\lambda_1|-k}.$$

If $|x| < |\lambda_1|$, then we conclude that $x < \lambda_1$. Otherwise, $x = x'x''$, where $|x'| = |\lambda_1|$ and $x' \neq \lambda_1$. Hence, $x' < \lambda_1$, so again $x < \lambda_1$. In both cases we have $x < \lambda_1 < \lambda_2$, which contradicts the fact that a Lyndon word is smaller than all its suffixes.

(ii) Suppose to the contrary that $\lambda_1 < \lambda_2$ but $\lambda_1^{n/|\lambda_1|} \geq \lambda_2^{n/|\lambda_2|}$. Then

$$\lambda_1 < \lambda_2 \leq \lambda_2^{n/|\lambda_2|} \leq \lambda_1^{n/|\lambda_1|}.$$

This contradicts part (i). ∎

3. Combinatorics of ranking Lyndon words. Recall that, for a word $w \in \Sigma^n$, we defined $\mathit{LynRank}(w)$ as the number of Lyndon words in Σ^n not exceeding w. Our basic goal (stated as Problem 1) is to efficiently compute $\mathit{LynRank}(w)$ for a given word w. It suffices to compute $\mathit{LynRank}(w)$ for a self-minimal word w. If w is not self-minimal, then $\mathit{LynRank}(w) = \mathit{LynRank}(w')$, where w' is the greatest self-minimal word such that $w' \leq w$; such w' can be computed efficiently using Lemma 6.

We will show how to reduce computation of $\mathit{LynRank}(w)$ to the computation of the cardinality of the following set:

$$\mathsf{S}(v) = \{x \in \Sigma^{|v|} : \langle x \rangle \leq v\}$$

for some prefixes v of w.

Example 8. For $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, there are seven words of length 4 lexicographically smaller than or equal to $w = \mathtt{abba}$:

$$\mathtt{aaaa}, \mathtt{aaab}, \mathtt{aaba}, \mathtt{aabb}, \mathtt{abaa}, \mathtt{abab}, \mathtt{abba}.$$

This set contains words from the following four equivalence classes. Each class includes a self-minimal word that is underlined.

$$\{\underline{\mathtt{aaaa}}\} \cup \{\underline{\mathtt{aaab}}, \mathtt{aaba}, \mathtt{abaa}\} \cup \{\underline{\mathtt{aabb}}, \mathtt{abba}\} \cup \{\underline{\mathtt{abab}}\}.$$

Thus, $\mathsf{S}(w)$ consists of four full classes of cyclic equivalence:

$$\mathsf{S}(w) = \{\underline{\mathtt{aaaa}}\} \cup \{\underline{\mathtt{aaab}}, \mathtt{aaba}, \mathtt{abaa}, \mathtt{baaa}\} \cup \{\underline{\mathtt{aabb}}, \mathtt{abba}, \mathtt{bbaa}, \mathtt{baab}\} \cup \{\underline{\mathtt{abab}}, \mathtt{baba}\}.$$

Let us introduce the following auxiliary sets defined for $w \in \Sigma^n$ and divisors $d \mid n$:

$$\mathsf{S}_d(w) = \{x \in \Sigma^d : \langle x \rangle^{n/d} \leq w\},$$
$$\mathsf{S}'_d(w) = \{x \in \Sigma^d : x \text{ is primitive}, \langle x \rangle^{n/d} \leq w\}.$$

Example 9. For $w = \mathtt{abbaaa}$ and $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, we have

$$\mathsf{S}_2(w) = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}\}, \qquad \mathsf{S}'_2(w) = \{\mathtt{ab}, \mathtt{ba}\},$$
$$\mathsf{S}_3(w) = \{\mathtt{aaa}, \mathtt{aab}, \mathtt{aba}, \mathtt{baa}\}, \qquad \mathsf{S}'_3(w) = \{\mathtt{aab}, \mathtt{aba}, \mathtt{baa}\}.$$

As shown in the following two facts, *LynRank*(w) is closely related to $|\mathsf{S}'_n(w)|$, which can be expressed in terms of $|\mathsf{S}_d(w)|$ for $d \mid n$.

FACT 10. *For every word $w \in \Sigma^n$, we have* $\mathit{LynRank}(w) = \frac{1}{n}|\mathsf{S}'_n(w)|$.

Proof. Observe that $\mathsf{S}'_n(w)$ 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. ∎

FACT 11. *For every word $w \in \Sigma^n$, if $d \mid n$, then* $|\mathsf{S}'_d(w)| = \sum_{\ell|d} \mu(\frac{d}{\ell})|\mathsf{S}_\ell(w)|$.

Proof. We first show that

$$|\mathsf{S}_\ell(w)| = \sum_{d|\ell} |\mathsf{S}'_d(w)|. \tag{1}$$

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

$$\mathsf{S}_\ell(w) = \bigcup_{d|\ell} \left\{ y \in \Sigma^d : y \text{ is primitive}, \left\langle y^{\ell/d} \right\rangle^{n/\ell} \le w \right\},$$

and the sum is disjoint. Now $\left\langle y^{\ell/d} \right\rangle^{n/\ell} = \langle y \rangle^{n/d}$ implies (1). From this formula, we obtain the claimed equality by the Möbius inversion formula. ∎

The $\mathsf{S}_d(w)$ sets are closely related to the regular $\mathsf{S}(v)$ sets for prefixes v of w. It is most evident for a self-minimal word.

FACT 12. *If $w \in \Sigma^n$ is self-minimal and $d \mid n$, then* $\mathsf{S}_d(w) = \mathsf{S}(w_{(d)})$.

Proof. If $d = n$, the equality of the two sets is trivial. Assume that $d < n$. Let us prove the equality by showing both inclusions.

Assume that $x \in \mathsf{S}_d(w)$. This means that $\langle x \rangle^{n/d} \le w$, and therefore $\langle x \rangle \le w_{(d)}$ (as $|x| = d$). Hence, $x \in \mathsf{S}(w_{(d)})$.

Now assume that $x \in \mathsf{S}(w_{(d)})$. This means that $\langle x \rangle \le w_{(d)}$. We have $\langle x \rangle^{n/d} \le (w_{(d)})^{n/d} \le w$, where the second inequality is due to Fact 3. Hence, $x \in \mathsf{S}_d(w)$. ∎

The facts that we have just proved let us derive a formula for *LynRank*(w).

LEMMA 13. *If a word $w \in \Sigma^n$ is self-minimal, then*

$$\mathit{LynRank}(w) = \tfrac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d}) \left|\mathsf{S}(w_{(d)})\right|.$$

Proof. We use Facts 10 to 12 in a series of transformations to express *LynRank*(w) using $|\mathsf{S}'_n(w)|$, $|\mathsf{S}_d(w)|$ for $d \mid n$, and finally $|\mathsf{S}(w_{(d)})|$ for $d \mid n$:

$$\mathit{LynRank}(w) = \tfrac{1}{n} \left|\mathsf{S}'_n(w)\right| = \tfrac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d}) \left|\mathsf{S}_d(w)\right| = \tfrac{1}{n} \sum_{d|n} \mu(\tfrac{n}{d}) \left|\mathsf{S}(w_{(d)})\right|.$$

∎

Example 14. Let $w = \mathtt{ababbb}$. We have $w_{(1)} = \mathtt{a}$, $w_{(2)} = \mathtt{ab}$, $w_{(3)} = \mathtt{aba}$ and

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

$$\begin{aligned}\mathit{LynRank}(w) = \tfrac{1}{6} \cdot \big(\mu(1)\,\big|\mathsf{S}(w)\big| + \mu(2)\,\big|\mathsf{S}(w_{(3)})\big| + \mu(3)\,\big|\mathsf{S}(w_{(2)})\big| + \mu(6)\,\big|\mathsf{S}(w_{(1)})\big|\big)\\ = \tfrac{1}{6} \cdot (54 - 4 - 3 + 1) = 8.\end{aligned}$$

The set of Lyndon words of length 6 that are not greater than $w = \texttt{ababbb}$ is

$$\{\texttt{aaaaab}, \texttt{aaaabb}, \texttt{aaabab}, \texttt{aaabbb}, \texttt{aababb}, \texttt{aabbab}, \texttt{aabbbb}, \texttt{ababbb}\}.$$

Indeed, it contains eight elements.

The next three sections are devoted to a proof of the following lemma.

LEMMA 15. *For a self-minimal word $w \in \Sigma^n$, one can compute $|\mathsf{S}(w)|$*
(i) *in $\mathcal{O}(n^2)$ time in the unit-cost RAM and*
(ii) *in $\mathcal{O}(n^2 \log \sigma)$ time in the word RAM.*

As a consequence of this lemma, we obtain efficient ranking of Lyndon words.

FACT 16. *If $\alpha > 1$ is a real constant, 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,

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

∎

THEOREM 17. *For an arbitrary word w of length n, one can compute $\mathit{LynRank}(w)$ in $\mathcal{O}(n^2 \log \sigma)$ time in the word RAM or in $\mathcal{O}(n^2)$ time in the unit-cost RAM.*

Proof. We use the formula given by Lemma 13 and the algorithm of Lemma 15. If any of the words w, $w_{(d)}$ is not self-minimal, then instead we take the greatest word of the same length that is not greater than it and is self-minimal (using Lemma 6). The time complexity is $\mathcal{O}(\sum_{d|n} d^2 \log \sigma)$ in the word RAM or $\mathcal{O}(\sum_{d|n} d^2)$ in the unit-cost RAM which, by Fact 16, reduces to $\mathcal{O}(n^2 \log \sigma)$ in the word RAM or $\mathcal{O}(n^2)$ in the unit-cost RAM, respectively. ∎

We also obtain an efficient algorithm for "unranking" Lyndon words.

THEOREM 18. *The kth Lyndon word of length n can be found in $\mathcal{O}(n^3 \log^2 \sigma)$ time in the word RAM or $\mathcal{O}(n^3 \log \sigma)$ time in the unit-cost RAM.*

Proof. By definition of the *LynRank* function, we are looking for the smallest $w \in \Sigma^n$ such that $\mathit{LynRank}(w) \ge k$. We binary search Σ^n with respect to the lexicographic order, using the algorithm of Theorem 17 to check whether $\mathit{LynRank}(w) \ge k$. The size of the search space is σ^n, which gives an additional $n \log \sigma$-time factor. ∎

4. Automata-theoretic interpretation. From now on we assume that w is self-minimal. Our goal is to compute $|\mathsf{S}(w)|$.

Let $\mathrm{Pref}_-(w) = \{w_{(i)}s : i \in \{0, \dots, n-1\}, s \in \Sigma, s < w[i+1]\} \cup \{w\}$. 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 19. $\mathsf{S}(w) = \sqrt{L(w)} \cap \Sigma^n$.

Proof. Consider a word $x \in \Sigma^n$. If $x \in \mathsf{S}(w)$, then $\langle x \rangle \le w$. Take $y = \langle x \rangle$, which is a factor of x^2. 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 that $x \in \sqrt{L(w)}$ so that 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 $x' \leq w$, and hence $\langle x \rangle \leq x' \leq w$ and $x \in \mathsf{S}(w)$. ∎

We construct a deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ recognizing $L(w)$. It has $|Q| = n + 1$ states: one for each prefix of w. The initial state is $q_0 = w_{(0)}$, and the only accepting state (the only element of the set F) is $w_{(n)} = AC$. The transitions are defined as follows: we set $\delta(AC, c) = AC$ for any $c \in \Sigma$ and

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

Figure 1 contains an example of such an automaton.

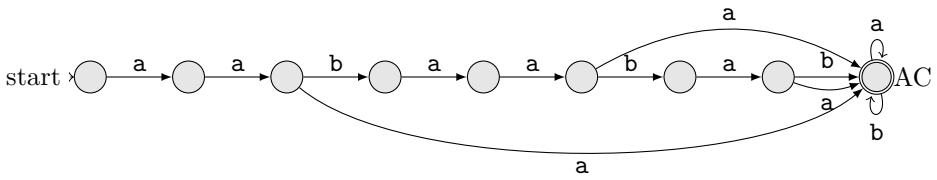


FIG. 1. *Automaton A that accepts $L(w)$ for a word $w =$* `aabaabab` *and alphabet $\Sigma =$* $\{$`a`, `b`$\}$. *Missing links lead to the initial state.*

Note that all accepting paths in the automaton have a simple structure. Each of them can be divided into fragments, each of which is a path that starts in $w_{(0)}$, visits a number of states corresponding to subsequent prefixes of w, and eventually either goes back to $w_{(0)}$ or terminates at AC. In the latter case the word spelled by the path fragment is an element of $\mathrm{Pref}_{-}(w)$. After the path reaches AC, it stays there. Hence, if a word x is accepted by the automaton, then it contains a factor from $\mathrm{Pref}_{-}(w)$, so $x \in L(w)$. Consequently, $L(A) \subseteq L(w)$. By a more thorough analysis we show below that $L(A) = L(w)$.

LEMMA 20. *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 suffix 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 three different states, so A is sparse.

The following corollary summarizes the construction of A.

COROLLARY 21. *Let $w \in \Sigma^n$ be a self-minimal word. One can construct a sparse automaton A with $\mathcal{O}(n)$ states recognizing $L(w)$.*

Let us use the natural extension of the transition function of an automaton into words:

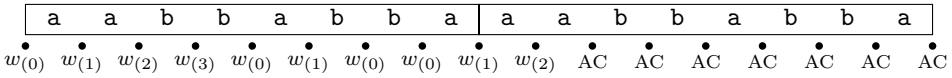
$$\delta(q, x) = \delta(\ldots\delta(\delta(q, x[1]), x[2])\ldots, x[k]) \quad \text{for } x \in \Sigma^k.$$

For states $q, q' \in Q$ let us define the set $L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}$ of the labels of walks from q to q'. The following lemma shows a crucial property of the words x^2 from the language $L(A)$ such that $x \notin L(A)$. It makes use of the special structure of the automaton A.

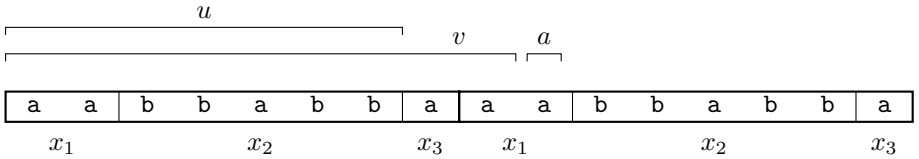
LEMMA 22. *Let $x \in \Sigma^n$. If $x^2 \in L(A)$ but $x \notin L(A)$, then there is a unique decomposition $x = x_1x_2x_3$ such that $x_1, x_3 \neq \varepsilon$, $x_3x_1 \in \mathrm{Pref}_-(w)$, and $x_1x_2 \in L_A(w_{(0)}, w_{(0)})$.*

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

Example 23. Let $w =$ `aabaabab`. Recall that the automaton A such that $L(A) = L(w)$ was shown in Figure 1. Consider a word $x =$ `aabbabba` of the same length as w. For this word, $x \notin L(A)$ and $x^2 \in L(A)$. Black circles below represent the states of the automaton A after processing the subsequent letters of x^2:



For this word, the decomposition of Lemma 22 is as follows:



In this case in the proof of the lemma we have $u =$ `aabbabb`, $v =$ `aabbabbaa`, and $k = 2$.

Denote $\pi_k(i, j) = |L_A(w_{(i)}, w_{(j)}) \cap \Sigma^k|$. We say that a number is *small* if it fits into a constant number of machine words or, in other words, is polynomial with respect

to $n+\sigma$. Using Lemma 22, we obtain a formula for $|\mathsf{S}(w)|$.

LEMMA 24. *For every self-minimal word $w \in \Sigma^n$, there exist coefficients $\alpha_{i,j}$ that are small numbers such that*

$$|\mathsf{S}(w)| = \pi_n(0,n) + \sum_{i,j=0}^{n} \alpha_{i,j}\pi_j(i,0).$$

Moreover, the coefficients $\alpha_{i,j}$ can all be computed in $\mathcal{O}(n^2)$ time.

Proof. We apply Fact 19 with Corollary 21 and actually compute $|\{x \in \Sigma^n : x^2 \in L(A)\}|$. If $x \in L(A)$, then obviously $x^2 \in L(A)$. For this part, we need to compute $|L(A) \cap \Sigma^n|$, which is exactly $\pi_n(0,n)$. Now it suffices to count $x \in \Sigma^n$ such that $x^2 \in L(A)$ but $x \notin L(A)$.

Let us recall the characterization of such words from Lemma 22. We consider all $\mathcal{O}(n^2)$ choices of $|x_1|$ and $|x_3|$ and count 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 $x_3x_1 = x_3x_1'a \in \mathrm{Pref}_-(w)$, so x_3x_1' is a prefix $w_{(k)}$ of w and $\delta(w_{(k)}, a) = AC$. Hence, k is uniquely determined by $|x_1|$ and $|x_3|$. In particular, $x_3 = w[1, |x_3|]$ and $x_1' = w[|x_3|+1, k]$ are uniquely determined.

Let us define ℓ as $w_{(\ell)} = \delta(w_{(0)}, x_1')$; see Figure 2. To efficiently determine ℓ for each choice of $|x_1|$ and $|x_3|$, we precompute $\delta(w_{(0)}, w[i,j])$ for all factors $w[i,j]$ of w. Since

$$\delta(w_{(0)}, w[i, j+1]) = \delta(\delta(w_{(0)}, w[i,j]), w[j+1]),$$

these values can be computed in $\mathcal{O}(n)$ time for each i, i.e., $\mathcal{O}(n^2)$ time in total.

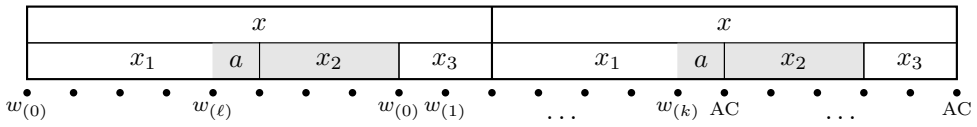


FIG. 2. *Illustration of Lemma 24. Both lines represent different factorizations of the same word x^2. Black circles represent states of the automaton. Only shaded letters are not necessarily uniquely determined by $|x_3|$ and $|x_1|$ for a fixed w.*

Once we know ℓ, we need to count

$$\left\{ax_2 \in \Sigma^{n-k} \ : \ a \in \Sigma,\, \delta(w_{(k)}, a) = AC, \text{ and } ax_2 \in L_A(w_{(\ell)}, w_{(0)})\right\}.$$

Note that $\delta(w_{(\ell)}, a) \in \{w_{(0)}, w_{(\ell+1)}\}$ since $\delta(w_{(\ell)}, a) = AC$ would imply that $x \in L(A)$. Thus, the number of words ax_2 is equal to

$$\sum_{q \in \{0, \ell+1\}} \gamma(k, \ell, q)\, \pi_{n-k-1}(q, 0), \tag{2}$$

where $\gamma(k,\ell,q) = |\{a \in \Sigma : \delta(w_{(\ell)}, a) = q \wedge \delta(w_{(k)}, a) = AC\}|$. Each coefficient $\gamma(k,\ell,q)$ can be computed in constant time since in our automaton A the transition function δ has an especially simple form. By rearranging the summands of (2), we obtain a formula for $|\mathsf{S}(w)|$ in the desired form. ∎

5. Ranking Lyndon words with $O(n^2)$ arithmetic operations. In this section by arithmetic operations we mean addition, subtraction, and multiplication. The following lemma shows how to efficiently count certain walks in the automaton A recognizing $L(w)$. Its proof is based on vector-matrix multiplication.

LEMMA 25. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a sparse deterministic automaton with n states. Given $q \in Q$ and $m \in \mathbb{Z}_{\geq 0}$, it takes $\mathcal{O}(mn)$ arithmetic operations on integers of magnitude σ^m to compute all values $|L_A(q, q') \cap \Sigma^k|$ and $|L_A(q', q) \cap \Sigma^k|$ for $0 \leq k \leq m$, $q' \in Q$.*

Proof. We construct an $n \times n$ matrix M with rows and columns indexed by states from Q. Set $M_{q,q'} = |\{a \in \Sigma : \delta(q, a) = q'\}|$. It is easy to see that $(M^k)_{q,q'} = |L_A(q, q') \cap \Sigma^k|$. Consequently, the entries of M^k belong to $\{0, \ldots, \sigma^k\}$.

Note that the matrix M is sparse, i.e., it contains $\mathcal{O}(n)$ nonzero entries. Thus, for a (vertical) vector $\mathbf{v}$ one can compute $M\mathbf{v}$ and $\mathbf{v}^T M$ using $\mathcal{O}(n)$ arithmetic operations. For $q \in Q$, let $\mathbf{e}_q$ be the unit vector with one at the position corresponding to q. Observe that $(M^k)_{q,q'}$ is equal to the q'th entry of $\mathbf{e}_q^T M^k$. For a fixed state $q \in Q$, we can compute these (horizontal) vectors for $0 \leq k \leq m$ using m vector-matrix multiplications. Symmetrically, $|L_A(q', q) \cap \Sigma^k| = (M^k)_{q',q}$ is the q'th entry of $M^k \mathbf{e}_q$, and we can also compute these (vertical) vectors for $0 \leq k \leq m$ using m matrix-vector multiplications. In total, we perform $\mathcal{O}(mn)$ arithmetic operations. ∎

The algorithm below combines the results obtained so far to provide the implementation for Lemma 15(i).

Proof of Lemma 15(i). Our algorithm is based on the formula of Lemma 24, whose proof already provides a procedure to compute the coefficients $\alpha_{i,j}$. On the other hand, Lemma 25 states that values $\pi_j(0, i)$ and $\pi_j(i, 0)$ for $0 \leq i, j \leq n$ can be determined using $\mathcal{O}(n^2)$ arithmetic operations given the automaton recognizing $L(w)$.

Algorithm *Computing $|\mathsf{S}(w)|$ in $\mathcal{O}(n^2)$ time in the unit-cost RAM model*
Construct automaton A for w { Corollary 21 }
Compute $\pi_j(0, i)$ and $\pi_j(i, 0)$ for all $0 \leq i, j \leq n$ { Lemma 25 }
Compute $\alpha_{i,j}$ coefficients { Lemma 24 }
$|\mathsf{S}(w)| := \pi_n(0, n) + \sum_{i,j=0}^{n} \alpha_{i,j} \pi_j(i, 0)$ { Lemma 24 }

Thus, our algorithm, given in the pseudocode above, performs $\mathcal{O}(n^2)$ arithmetic operations on integers of magnitude σ^n to compute $|\mathsf{S}(w)|$ for a self-minimal word w. ∎

In the unit-cost RAM model of arithmetic operations, we obtain $\mathcal{O}(n^2)$ time. It is easy to check that all arithmetic operations performed in the algorithm above are additions and subtractions of numbers not exceeding σ^n and multiplications of such numbers by small numbers. Hence, in the word-RAM model we obtain $\mathcal{O}(n^3)$ time. In the following section we give an algorithm working in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model.

6. Ranking Lyndon words in $\mathcal{O}(n^2 \log \sigma)$ time on word RAM. The improvement of the time complexity requires a modification of the formula of Lemma 24, after which we perform $\mathcal{O}(n^2)$ arithmetic operations only on small integers and only $\mathcal{O}(n)$ operations on large integers. We also use Newton's iteration for power series inversion [34]; see also [18, p. 140].

FACT 26. *Let $T(n)$ be the time necessary to compute the inverse of a power series $G(x)$ of degree n modulo x^n, that is, the time to compute a power series $F(x)$ of degree n such that $F(x)G(x) \equiv 1 \pmod{x^n}$. Then $T(n)$ satisfies*

$$T(2^k) \leq T(2^{k-1}) + cM(2^{k-1}),$$

where $c > 0$ is a constant and $M(n)$ is the time to multiply two polynomials of degree n with coefficients of magnitude not exceeding the nth coefficient of $F(x)$.

For an efficient implementation of Fact 26, we use an integer multiplication algorithm designed for the word-RAM model; see Fürer [17].

LEMMA 27. *Two polynomials of degree at most n with coefficients of magnitude σ^n can be multiplied in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model.*

Proof. Let $F(x)$ and $G(x)$ be the considered polynomials. We encode them as integers u and v as follows. Both u and v are divided into n chunks consisting of $n \log \sigma + \log n$ bits each. The ith least significant chunk of u (respectively, v) holds the ith coefficient of $F(x)$ (respectively, $G(x)$) prepended by zeroes. Then the corresponding chunks of uv hold the coefficients of $F(x)G(x)$. Both numbers u and v have $\mathcal{O}(n^2 \log \sigma)$ bits. Therefore, the product uv can be computed in $\mathcal{O}(n^2 \log \sigma)$ time [17]. ∎

With the auxiliary fact, Fact 16, we obtain the following tool.

LEMMA 28. *Let $F(x)$ and $G(x)$ be power series such that $F(x)G(x) \equiv 1$. Assume that the kth coefficient of $F(x)$ is of magnitude σ^k. If the coefficients of $G(x)$ can be computed in $\mathcal{O}(1)$ time, then $F(x) \bmod x^n$ can be computed in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model.*

Now we show how to use Lemma 28 to count specific paths in the automaton A for the word w. Denote

$$T_i = \pi_i(0,0) \quad \text{and} \quad a_i = |\{c \in \Sigma : \delta(w_{(i-1)}, c) = w_{(0)}\}| = |\{c \in \Sigma : c > w[i]\}|.$$

LEMMA 29. *All values $T_0, \ldots, T_n$ can be computed in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model.*

Proof. Assume that for $k < 0$, $T_k = 0$. Recall that a nonempty path from $w_{(0)}$ to itself in A passes through a number of consecutive states $w_{(1)}, w_{(2)}, \ldots, w_{(i)}$ before it first comes back to $w_{(0)}$. Hence, T_k satisfy the following recurrence:

$$T_k = \begin{cases} 0 & \text{for } k < 0, \\ 1 & \text{for } k = 0, \\ a_1 T_{k-1} + \cdots + a_n T_{k-n} & \text{otherwise.} \end{cases}$$

Let us set $a_0 = -1$. Let F and G be the generating functions of T_k and a_k:

$$F(x) = \sum_{k=0}^{\infty} T_k x^k, \quad G(x) = \sum_{k=0}^{n} a_k x^k.$$

Note that

$$\begin{aligned} F(x)G(x) &= \sum_{k=0}^{\infty} x^k \sum_{m=0}^{k} a_m T_{k-m} = -1 + \sum_{k=1}^{\infty} x^k \sum_{m=0}^{n} a_m T_{k-m} \\ &= -1 + \sum_{k=1}^{\infty} x^k (-T_k + \sum_{m=1}^{n} a_m T_{k-m}) = -1. \end{aligned}$$

This concludes that we can use Lemma 28 to compute n first coefficients of $F(x)$ in $\mathcal{O}(n^2 \log \sigma)$ time. ∎

We extend the results of the previous lemma to compute the first term of the formula for $|\mathsf{S}(w)|$.

LEMMA 30. *The value $\pi_n(0,n)$ can be computed in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model.*

Proof. Note that

$$\pi_n(0,n) = \sum_{i=0}^{n-1} T_i c_{n-i}, \tag{3}$$

where c_j is the number of paths of length j that start in $w_{(0)}$, end in AC, and do not pass through $w_{(0)}$ again. Denote $a'_i = |\{a \in \Sigma : \delta(w_{(i-1)}, a) = AC\}|$. Note that $a'_i = \sigma - 1 - a_i$ for $i < n$ and $a'_n = \sigma - a_n$. Moreover, for every $j \in \{1, \ldots, n\}$,

$$c_j = a'_1 \sigma^{j-1} + a'_2 \sigma^{j-2} + \cdots + a'_j,$$

as in the considered path we traverse some number of edges $k \in \{0, \ldots, j-1\}$ passing through $w_{(0)}, \ldots, w_{(k)}$; then we use an edge to the accepting state and stay in that state for the remaining $j - 1 - k$ steps.

Due to the recurrence $c_{j+1} = \sigma c_j + a'_{j+1}$, all values c_j can be computed in $\mathcal{O}(n^2)$ time. By Lemma 29, all values T_j can be computed in $\mathcal{O}(n^2 \log \sigma)$ time. Obviously $c_j, T_j \le \sigma^j$. This concludes that we can use the algorithm of Lemma 27 to multiply the polynomials

$$F(x) = \sum_{i=0}^{n-1} T_i x^i \quad \text{and} \quad G(x) = \sum_{i=0}^{n-1} c_{i+1} x^i.$$

The coefficient of $F(x)G(x)$ at x^{n-1} is exactly the desired sum (3). ∎

Finally, we are ready to prove Lemma 15(ii). To this end, we show that the remaining terms of the formula for $|\mathsf{S}(w)|$ can be computed efficiently in the word-RAM model.

Proof of Lemma 15(ii). We provide an efficient implementation of the formula from Lemma 24. For the $\pi_n(0,n)$ part we use Lemma 30. Now we show how to transform the coefficients $\alpha_{i,j}$ to obtain an equivalent set of small coefficients $\beta_{i,j}$ satisfying $\beta_{i,j} \neq 0$ if and only if $i = 0$ or $j = 0$. We use the following claim.

Claim 31. For $0 \le i < n$ and $j \ge 1$, we have

$$\pi_j(i,0) = \pi_{j-1}(i+1,0) + a_{i+1}\pi_{j-1}(0,0). \tag{4}$$

Moreover, $\pi_j(n,0) = 0$ for $j \ge 0$.

The formula (4) corresponds to traversing the first edge of the path from i to 0. We arrive at the following algorithm, which reduces computation of the required sum of a quadratic number of large numbers to the computation of a linear combination of only linearly many big numbers T_j.

Algorithm *Compute* $|\mathsf{S}(w)|$
  foreach $i, j \in \{0, \ldots, n\}$ **do**
    $\beta_{i,j} := \alpha_{i,j}$
  end
  for $j := n$ **downto** 1 **do**
    for $i := 1$ **to** $n-1$ **do**
      $\beta_{i+1,j-1}$ += $\beta_{i,j}$
      $\beta_{0,j-1}$ += $a_{i+1}\beta_{i,j}$
      $\beta_{i,j} := 0$
    end
  end
  return $\pi_n(0,n) + \sum_{j=0}^{n} \beta_{0,j} \cdot T_j$

Denote $A = \sum_{i,j=0}^{n} \beta_{i,j}\pi_j(i,0)$. By (4) we have

$$A = A - \beta_{i,j}\pi_j(i,0) + \beta_{i,j}\pi_{j-1}(i+1,0) + \beta_{i,j}a_{i+1}\pi_{j-1}(0,0).$$

Consequently, resetting $\beta_{i,j}$ to zero and increasing the coefficients $\beta_{i+1,j-1}$ and $\beta_{0,j-1}$ in the inner iteration does not alter the total sum A. Hence, after every iteration of the inner for-loop the coefficients satisfy the following invariant:

$$A = \sum_{i,j=0}^{n} \beta_{i,j}\pi_j(i,0) = \sum_{i,j=0}^{n} \alpha_{i,j}\pi_j(i,0).$$

Observe that once $\beta_{i,j}$ is reset to zero, it will not be changed anymore. Hence, at the end of the algorithm we have $\beta_{i,j} = 0$ if $j > 0$ and $1 \le i \le n-1$. Note that $\beta_{n,j} \cdot \pi_j(n,0) = \beta_{n,j} \cdot 0 = 0$ for each j and $\beta_{i,0} \cdot \pi_0(i,0) = \beta_{i,0} \cdot 0 = 0$ for $i \neq 0$. This concludes that at the end of the algorithm we have

$$\sum_{i,j=0}^{n} \alpha_{i,j}\pi_j(i,0) = \sum_{i,j=0}^{n} \beta_{i,j}\pi_j(i,0) = \sum_{j=0}^{n} \beta_{0,j} \cdot T_j.$$

Note that each $\alpha_{i,j}$ coefficient accounts in $\sum_j \beta_{0,j}$ as at most $(a_{i+1} + a_{i+2} + \cdots + a_n)\alpha_{i,j}$. Hence, the sum of the resulting nonzero coefficients $\beta_{i,j}$ does not exceed σn times the sum of the initial values $\alpha_{i,j}$. At the end, we will compute a linear combination of T_j with small coefficients. Consequently, Lemma 29 yields an $\mathcal{O}(n^2 \log \sigma)$-time algorithm on the word RAM. ∎

7. Decoding minimal de Bruijn sequence. In this section we focus on decoding lexicographically minimal de Bruijn sequence dB_n over Σ: we aim at an efficient algorithm that for every $w \in \Sigma^n$ computes *occ-pos*(w, dB_n), that is, the position of the sole occurrence of w in dB_n. Recall that by $\mathcal{L}^{(n)}$ we denote the set of Lyndon words over Σ whose length is a divisor of n. A theorem of Fredricksen and Maiorana (see [13, 16, 20]) states that dB_n is a concatenation of the Lyndon words from $\mathcal{L}^{(n)}$ in the lexicographic order. The proof of the theorem is constructive; i.e., for any word w of length n it shows the concatenation of a constant number of consecutive Lyndon words from the cyclic version of the sequence $\mathcal{L}^{(n)}$ that contain w. This, together with the following lemma which relates dB_n to S, lets us compute the exact position where w occurs in dB_n.

LEMMA 32. *Let $w \in \Sigma^n$ and $\mathcal{L}(w) = \{\lambda \in \mathcal{L}^{(n)} : \lambda^{n/|\lambda|} \leq w\}$. 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 $|\mathsf{S}(w)|$.*

Proof. First note that, by Fact 7(ii), the lexicographic order of elements $\lambda \in \mathcal{L}^{(n)}$ coincides with the lexicographic order of $\lambda^{n/|\lambda|}$. This shows that the concatenation of elements of $\mathcal{L}(w)$ indeed forms a prefix of dB_n.

It remains to show that $\sum_{\lambda\in\mathcal{L}(w)} |\lambda| = |\mathsf{S}(w)|$. For this we shall build a mapping $\phi : \Sigma^n \to \mathcal{L}^{(n)}$ such that $|\phi^{-1}(\lambda)| = |\lambda|$ and $\langle x\rangle \leq w$ for $x \in \Sigma^n$ 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 $\phi(x)$ indeed belongs to $\mathcal{L}^{(n)}$. Moreover, to each Lyndon word λ of length $d \mid n$ we have assigned $v^{n/d}$ for each cyclic rotation v of λ. Thus $|\phi^{-1}(\lambda)| = |\lambda|$. Also, $\langle x\rangle = \langle y\rangle^{n/d}$, so $\langle x\rangle \leq w$ if and only if $\phi(x)^{n/d} \leq w$, i.e., $\phi(x) \in \mathcal{L}(w)$. ∎

THEOREM 33. *Given a word $w \in \Sigma^n$, occ-pos(w, dB_n) can be found in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model or $\mathcal{O}(n^2)$ time in the unit-cost RAM model.*

Proof. Let $\lambda_1 < \lambda_2 < \cdots < \lambda_p$ be all Lyndon words in $\mathcal{L}^{(n)}$ (we have $\lambda_1\lambda_2\cdots\lambda_p = dB_n$). The proof of the theorem of Fredricksen and Maiorana (see [16, 20]) describes the occurrence of w in dB_n which can be stated succinctly as follows.

Claim 34 (Fredricksen and Maiorana [16], Knuth [20]). 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 $z = \max\Sigma$.

(i) If $w = z^i a^{n-i}$ for $i \geq 1$, then w occurs in dB_n at position $\sigma^n - i + 1$.
(ii) If $\alpha \neq z^{|\alpha|}$, then w is a factor of $\lambda_k\lambda_{k+1}$.
(iii) If $\alpha = z^{|\alpha|}$ and $d > 1$, then w is a factor of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.
(iv) 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 (i) it is easy to locate w in dB_n. Further on we consider only the cases (ii), (iii), and (iv).

Observe that λ_k can be retrieved as the primitive root of $\langle w\rangle$. Also note that, by Fact 7(i), $\lambda_{k'}$ is the primitive root of the largest self-minimal word $w' \in \Sigma^n$ such that $w' < \beta a^{|\alpha|}$. Thus, $\lambda_{k'}$ can be computed in $\mathcal{O}(n^2)$ time using Lemma 6.

Once we know $\lambda_{k'}$ and λ_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 [15], 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 [20]. 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 λ_k or $\lambda_{k'}$ in dB_n. Lemma 32 proves that such an occurrence of $\lambda \in \mathcal{L}^{(n)}$ *ends* at position $|\mathsf{S}(\lambda^{\frac{n}{|\lambda|}})|$, which can be computed in $\mathcal{O}(n^2 \log\sigma)$ time in the word-RAM model or $\mathcal{O}(n^2)$ time in the unit-cost RAM model by Lemma 15. Applied to λ_k or $\lambda_{k'}$, this concludes the proof. ∎

Example 35. Below we present the four cases of the claim in the proof of Theorem 33 on the sequence dB_6 over a binary alphabet (i.e., the lexicographically minimal binary de Bruijn sequence of rank 6), which has the following decomposition into Lyn-

don words $\lambda_1, \lambda_2, \ldots, \lambda_{14}$:

$$\begin{array}{cccccccccccccc}
\lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 & \lambda_8 & \lambda_9 & \lambda_{10} & \lambda_{11} & \lambda_{12} & \lambda_{13} & \lambda_{14} \\
\underline{0\ 00}0001 & 00\underline{0011\ 00}0101 & 0001\underline{11\ 001\ 0}01011 & 001101 & 001111 & 01 & 0101\underline{11\ 011\ 0}111\underline{11\ 1} \\
 & \text{(ii) } 001100 & \text{(iv) } 110010 & & & & & & & & \text{(iii) } 110110 & & \text{(i) } 111000 &
\end{array}$$

Case (i): *occ-pos*$(111000, dB_6) = 62$, and 111000 appears as a factor of $\lambda_{13}\lambda_{14}\lambda_1\lambda_2$.
Case (ii): *occ-pos*$(001100, dB_6) = 10$, and 001100 appears as a factor of $\lambda_3\lambda_4$.
Case (iii): *occ-pos*$(110110, dB_6) = 53$, and 110110 appears as a factor of $\lambda_{11}\lambda_{12}\lambda_{13}$.
Case (iv): *occ-pos*$(110010, dB_6) = 24$, and 110010 appears as a factor of $\lambda_5\lambda_6\lambda_7$.

To compute the kth symbol of dB_n, we have to locate the Lyndon word from $\mathcal{L}^{(n)}$ containing the kth position of dB_n. We apply binary search as in Theorem 18.

THEOREM 36. *Given integers n and k, the kth symbol of dB_n can be computed in $\mathcal{O}(n^3 \log^2 \sigma)$ time in the word-RAM model or $\mathcal{O}(n^3 \log \sigma)$ time in the unit-cost RAM model.*

Proof. We binary search for the smallest word $v \in \Sigma^n$ such that $|\mathsf{S}(v)| \geq k$, using Lemma 15 to test the condition. In each step of the binary search, we actually consider a self-minimal word, due to Lemma 6. Therefore the resulting word v is of the form λ^d for some $\lambda \in \mathcal{L}^{(n)}$. By Lemma 32, a prefix of dB_n of length $|\mathsf{S}(v)|$ contains all Lyndon words from $\mathcal{L}(v)$. Moreover, by Fact 7(i), this prefix ends with λ. This means that the kth position of dB_n lies within the canonical occurrence of λ. More precisely, it suffices to return the $(|\mathsf{S}(v)| - k + 1)$st *last* symbol of λ (which is also the $(|\mathsf{S}(v)| - k + 1)$st last symbol of v). As in Theorem 18, the binary search introduces a multiplicative $\mathcal{O}(n \log \sigma)$ factor to the complexity of the algorithm of Lemma 15. □

Recently, Au [1] introduced a variant of a de Bruijn sequence in which each (cyclic) factor of length n is primitive and each primitive word from Σ^n occurs as a (cyclic) factor. He also proved that the lexicographically minimal sequence satisfying this condition, denoted by dB'_n, is the concatenation in lexicographic order of Lyndon words of length n over Σ.

Example 37. For $n = 6$ and the binary alphabet we have the following decomposition of dB'_6:

$$000001\ 000011\ 000101\ 000111\ 001011\ 001101\ 001111\ 010111\ 011111.$$

The ranking algorithm for Lyndon words lets us derive a counterpart of Theorem 33 for dB'_n with a slightly simpler proof (admitting a similar structure, though).

PROPOSITION 38. *Given a primitive word $w \in \Sigma^n$, occ-pos(w, dB'_n) can be found in $\mathcal{O}(n^2 \log \sigma)$ time in the word-RAM model or $\mathcal{O}(n^2)$ time in the unit-cost RAM model.*

Proof. Let $\lambda_1 < \lambda_2 < \cdots < \lambda_p$ be all Lyndon words in $\mathcal{L}_n$ (we have $\lambda_1\lambda_2 \cdots \lambda_p = dB'_n$). The proof of a theorem of Au [1, Theorem 9] describes the occurrence of w in dB'_n, which can be stated succinctly as follows.

Claim 39 (Au [1]). Assume that $w = \alpha\beta$, where $\alpha \neq \varepsilon$ and $\beta\alpha = \lambda_k$ is a Lyndon word of length n. Denote $a = \min \Sigma$ and $z = \max \Sigma$.

(i) If $w = z^i a^{n-i}$ for $i \geq 1$, then w occurs in dB'_n at position $|dB'_n| - i + 1$.
(ii) If $\alpha \neq z^{|\alpha|}$, then w occurs in $\lambda_k\lambda_{k+1}$ at position $1 + |\beta|$.
(iii) If $\alpha = z^{|\alpha|}$, then w occurs in $\lambda_{k'}\lambda_{k'+1}$ at position $1 + |\beta|$, where $\lambda_{k'}$ is the largest Lyndon word $\lambda \in \mathcal{L}_n$ such that $\lambda < \beta$.

In case (i), it is easy to locate w in dB'_n with $|dB'_n| = \sum_{d|n} \mu(\frac{n}{d})\sigma^d$. Otherwise, we observe that $\lambda_k = \langle w \rangle$ and this word can be computed using Fact 4 along with the decomposition $w = \alpha\beta$. In case (ii), we observe that the position of λ_k in dB'_n is $1 + n(k-1)$, so w occurs in dB'_n at position $1 + n(k-1) + |\beta| = 1 + nk - |\alpha|$. Thus, it suffices to determine $k = \mathit{LynRank}(\lambda_k)$ using Theorem 17. The situation in case (iii) is similar: w occurs in dB'_n at position $1 + nk' - |\alpha|$. Since $\lambda_{k'}$ is the largest Lyndon word smaller than β, we have $k' = \mathit{LynRank}(\beta a^{|\alpha|})$, i.e., the computation is also reduced to Theorem 17. ∎

Example 40. Below we present the three cases of the claim in the proof of Proposition 38 on a sequence dB'_6 over a binary alphabet, which has the following decomposition into Lyndon words $\lambda_1, \lambda_2, \ldots, \lambda_9$:

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9
000001	000011	000101	000111	001011	001101	001111	010111	011111
	(ii) 001100		(iii) 110010					(i) 111000

Case (i): $\mathit{occ\text{-}pos}(111000, dB'_6) = 52$, and 111000 appears as a factor of $\lambda_9\lambda_1$.
Case (ii): $\mathit{occ\text{-}pos}(001100, dB'_6) = 9$, and 001100 appears as a factor of $\lambda_2\lambda_3$.
Case (iii): $\mathit{occ\text{-}pos}(110010, dB'_6) = 23$, and 110010 appears as a factor of $\lambda_4\lambda_5$.

The kth symbol of dB'_n is much easier to find than the kth symbol of dB_n, as shown in the following result.

PROPOSITION 41. *Given integers n and k, the kth symbol of dB'_n can be computed in $\mathcal{O}(n^3 \log^2 \sigma)$ time in the word-RAM model or $\mathcal{O}(n^3 \log \sigma)$ time in the unit-cost RAM model.*

Proof. The kth symbol of the sequence dB'_n is the ith symbol of the jth Lyndon word of length n, where

$$i = ((k-1) \bmod n) + 1 \quad \text{and} \quad j = \left\lfloor \tfrac{k-1}{n} \right\rfloor + 1.$$

This word can be determined using Theorem 18. ∎

8. Conclusions. The main result of this paper is an $\mathcal{O}(n^2 \log \sigma)$-time algorithm in the word-RAM model and an $\mathcal{O}(n^2)$-time algorithm in the unit-cost RAM model for ranking Lyndon words. We have also presented efficient algorithms for computing a Lyndon word of a given length and rank in the lexicographic order, decoding the lexicographically minimal de Bruijn sequence of a given rank and computing a particular symbol of this sequence. Our results can also be applied to ranking necklaces due to a known connection between Lyndon words and necklaces; see [22].

Acknowledgment. We would like to thank Joe Sawada for making us aware of the work of Kopparty, Kumar, and Saks [22].

REFERENCES

[1] Y. H. Au, *Generalized de Bruijn words for primitive words and powers*, Discrete Math., 338 (2015), pp. 2320–2331, https://doi.org/10.1016/j.disc.2015.05.025.

[2] J. Berstel and M. Pocchiola, *Average cost of Duval's algorithm for generating Lyndon words*, Theoret. Comput. Sci., 132 (1994), pp. 415–425, https://doi.org/10.1016/0304-3975(94)00013-1.

[3] S. Bonomo, S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino, *Suffixes, conjugates and Lyndon words*, in Proceedings of the 17th International Conference on Developments in Language Theory (DLT 2013), Lecture Notes in Comput. Sci. 7907, M. Béal and O. Carton,

eds., Springer, Heidelberg, 2013, pp. 131–142, https://doi.org/10.1007/978-3-642-38771-5_13.

[4] K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C. R. Miers, *Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF*(2), J. Algorithms, 37 (2000), pp. 267–282, https://doi.org/10.1006/jagm.2000.1108.

[5] K. T. Chen, R. H. Fox, and R. C. Lyndon, *Free differential calculus,* IV. *The quotient groups of the lower central series*, Ann. of Math. (2), 68 (1958), pp. 81–95, https://doi.org/10.2307/1970044.

[6] F. R. K. Chung, P. Diaconis, and R. L. Graham, *Universal cycles for combinatorial structures*, Discrete Math., 110 (1992), pp. 43–59, https://doi.org/10.1016/0012-365X(92)90699-G.

[7] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on Strings*, Cambridge University Press, New York, 2007.

[8] 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*, Theoret. Comput. Sci., 521 (2014), pp. 29–41, https://doi.org/10.1016/j.tcs.2013.11.018.

[9] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, New York, 1994.

[10] N. G. de Bruijn, *A combinatorial problem*, Proc. Konink. Nederl. Akad. Wetensch., 49 (1946), pp. 758–764, www.dwc.knaw.nl/DL/publications/PU00018235.pdf.

[11] J.-P. Duval, *Factorizing words over an ordered alphabet*, J. Algorithms, 4 (1983), pp. 363–381, https://doi.org/10.1016/0196-6774(83)90017-2.

[12] J.-P. Duval, *Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée*, Theoret. Comput. Sci., 60 (1988), pp. 255–283, https://doi.org/10.1016/0304-3975(88)90113-2.

[13] H. Fredricksen, *The lexicographically least de Bruijn cycle*, J. Combinatorial Theory, 9 (1970), pp. 1–5, https://doi.org/10.1016/S0021-9800(70)80050-3.

[14] H. Fredricksen, *Generation of the Ford sequence of length* 2^n*, n large*, J. Combinatorial Theory Ser. A, 12 (1972), pp. 153–154, https://doi.org/10.1016/0097-3165(72)90091-X.

[15] H. Fredricksen and I. J. Kessler, *An algorithm for generating necklaces of beads in two colors*, Discrete Math., 61 (1986), pp. 181–188, https://doi.org/10.1016/0012-365X(86)90089-0.

[16] H. Fredricksen and J. Maiorana, *Necklaces of beads in k colors and k-ary de Bruijn sequences*, Discrete Math., 23 (1978), pp. 207–210, https://doi.org/10.1016/0012-365x(78)90002-x.

[17] M. Fürer, *How fast can we multiply large integers on an actual computer?*, in Proceedings of the Latin American Symposium on Theoretical Informatics (LATIN 2014), Lecture Notes in Comput. Sci. 8392, A. Pardo and A. Viola, eds., Springer, Heidelberg, 2014, pp. 660–670, https://doi.org/10.1007/978-3-642-54423-1_57.

[18] K. O. Geddes, S. R. Czapor, and G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publishers, Boston, MA, 1992.

[19] T. Hagerup, *Sorting and searching on the word RAM*, in Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998), Lecture Notes in Comput. Sci. 1373, M. Morvan, C. Meinel, and D. Krob, eds., Springer, Berlin, Heidelberg, 1998, pp. 366–398, https://doi.org/10.1007/BFb0028575.

[20] D. E. Knuth, *The Art of Computer Programming*, Vol. 4, Fasc. 2, Addison–Wesley, Upper Saddle River, NJ, 2005.

[21] T. Kociumaka, J. Radoszewski, and W. Rytter, *Computing kth Lyndon word and decoding lexicographically minimal de Bruijn sequence*, in Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM 2014), Lecture Notes in Comput. Sci. 8486, A. S. Kulikov, S. O. Kuznetsov, and P. A. Pevzner, eds., Springer, Cham, 2014, pp. 202–211, https://doi.org/10.1007/978-3-319-07566-2_21.

[22] S. Kopparty, M. Kumar, and M. E. Saks, *Efficient indexing of necklaces and irreducible polynomials over finite fields*, in Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP 2014), Part I, Lecture Notes in Comput. Sci. 8572, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, eds., Springer, Heidelberg, 2014, pp. 726–737, https://doi.org/10.1007/978-3-662-43948-7_60.

[23] M. Lothaire, *Combinatorics on Words*, 2nd ed., Cambridge Math. Lib., Cambridge University Press, Cambridge, UK, 1997.

[24] R. C. Lyndon, *On Burnside's problem*, Trans. Amer. Math. Soc., 77 (1954), pp. 202–215, https://doi.org/10.1090/S0002-9947-1954-0064049-X.

[25] C. Martínez and X. Molinero, *An Efficient Generic Algorithm for the Generation of Unlabelled Cycles*, Birkhäuser, Basel, 2004, pp. 187–197, https://doi.org/10.1007/

978-3-0348-7915-6_19.

[26] C. J. Mitchell, T. Etzion, and K. G. Paterson, *A method for constructing decodable de Bruijn sequences*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1472–1478, https://doi.org/10.1109/18.532887.

[27] M. Mucha, *Lyndon words and short superstrings*, in Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013), S. Khanna, ed., SIAM, Philadelphia, 2013, pp. 958–972, https://doi.org/10.1137/1.9781611973105.69.

[28] J. Radoszewski, *Generation of Lexicographically Minimal de Bruijn Sequences with Prime Words*, Master's thesis, University of Warsaw, Warsaw, Poland, 2008, in Polish, http://www.mimuw.edu.pl/~jrad/dyplomowe/mgr.pdf.

[29] A. Ralston, *A new memoryless algorithm for de Bruijn sequences*, J. Algorithms, 2 (1981), pp. 50–62, https://doi.org/10.1016/0196-6774(81)90007-9.

[30] F. Ruskey, *Combinatorial Generation*, draft version of a book, 2003, http://www.1stworks.com/ref/RuskeyCombGen.pdf.

[31] F. Ruskey, C. D. Savage, and T. M. Y. Wang, *Generating necklaces*, J. Algorithms, 13 (1992), pp. 414–430, https://doi.org/10.1016/0196-6774(92)90047-G.

[32] F. Ruskey and J. Sawada, *An efficient algorithm for generating necklaces with fixed density*, SIAM J. Comput., 29 (1999), pp. 671–684, https://doi.org/10.1137/S0097539798344112.

[33] J. Sawada and A. Williams, *private communication*, 2015.

[34] M. Sieveking, *An algorithm for division of powerseries*, Computing, 10 (1972), pp. 153–156, https://doi.org/10.1007/BF02242389.

[35] J. Tuliani, *De Bruijn sequences with efficient decoding algorithms*, Discrete Math., 226 (2001), pp. 313–336, https://doi.org/10.1016/S0012-365X(00)00117-5.