Efficient Ranking of Lyndon Words and Decoding Lexicographically Minimal de Bruijn Sequence*

Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter

Institute of Informatics, University of Warsaw, Poland

[kociumaka,jrad,rytter]@mimuw.edu.pl

October 12, 2015

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 the 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 (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).

1 Introduction

We consider finite words over an ordered alphabet Σ of size $\sigma = |\Sigma|$. A *Lyndon word* 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 3 letters is equal to it. Lyndon words have a number of combinatorial properties (see, e.g., [17]) 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 [14]). They are also related to necklaces of n beads in k colors, that is, equivalence classes of k-ary n-tuples under rotation [8, 9]. Lyndon words have a number of applications in the field of text algorithms; see e.g. [1, 4, 5, 19].

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

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

*This is an extended version of our previous conference paper [15] with complexities reduced by a $(\log \sigma)/n$ factor in case of the word-RAM model.

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 [2, 14]. Fredricksen and Maiorana [9] 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 Ł and $Ł_n$ the set of all Lyndon words and all Lyndon words of length n, respectively, and define

$$\mathit{LynRank}(w) \ = \ |\{x \in Ł_{|w|} \ : \ x \leq w\}|.$$

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 8 Lyndon words of length 6 that are not greater than `ababbb`:

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

What was known previously is that all Lyndon words of length at most n can be generated in lexicographic order by algorithm of Fredricksen, Kessler and Maiorana (FKM) [8, 9] (another algorithm was developed by Duval in [7]). The analysis from [21] shows that the FKM algorithm generates the subsequent Lyndon words in constant amortized time. However, there was no direct algorithm to generate a Lyndon word of an arbitrary rank or for ranking Lyndon words.

Let $Ł^{(n)} = \bigcup_{d|n} Ł_d$. 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 $Ł^{(n)}$ in lexicographic order. For a word w of length n over Σ, by $\mathit{occ\text{-}pos}(w, dB_n)$ we denote the position of its occurrence in dB_n. The problem of decoding the minimal de Bruijn sequence can be defined 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 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 [18, 24]. Such algorithms prove useful in certain types of position sensing applications of de Bruijn sequences [24]. No decoding algorithm for lexicographically minimal de Bruijn sequence was known before. 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 [20]). However, it does it only *in order*.

Recently a variant of de Bruijn words was introduced in [13]. Let dB'_n be the concatenation in lexicographic order of Lyndon words of length n over Σ. Then dB'_n is a cyclic sequence containing all primitive words of length n. As a by-product we obtain a polynomial-time algorithm for random access of symbols in dB'_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 model of computations. Our algorithms work in the *word-RAM* model; see [12]. 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., the numbers which fit in a constant number of machine words) are performed in constant time. The basic arithmetic operations on (large) numbers of size at most σ^n take $\mathcal{O}(n)$ 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 k-th Lyndon word of length n (in the lexicographic order) for a given k. We show an $\mathcal{O}(n^2 \log \sigma)$-time solution for decoding minimal de Bruijn sequence dB_n (Problem 2). We also obtain $\mathcal{O}(n^3 \log^2 \sigma)$-time algorithms computing the k-th symbol of dB_n and dB'_n for a given k. All these algorithms work in the word-RAM model. In the unit-cost RAM the complexities reduce by a factor of $\log \sigma$.

Related work. A preliminary version of this paper appeared as [15]. At about the same time, similar results were published by Kopparty, Kumar and Saks [16]. The work in these two papers was done independently. The papers both have polynomial-time algorithms for indexing Lyndon words and necklaces; the authors in [16] put the results in a broader context and have some additional applications (indexing irreducible polynomials and explicit constructions). On the other hand, we exercised more care in designing the algorithm to obtain a better polynomial running time. In particular, [15] 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 model where arithmetic operations on all integers work in constant time. 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 [22].

Structure of the paper. Sections 2-5 (and 7) contain a full version of the paper [15]. Section 2 defines the notions of self-minimal words 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 find the numbers of paths of specific types 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 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 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 $w[1, i]$. 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 $rot(w, c)$ 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 $rot(w, c)$ 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* if $\langle w \rangle = w$. The following observation gives a simple property of self-minimal words.

Observation 4. *If $w \in \Sigma^n$ is self-minimal and $d \mid n$, then $(w_{(d)})^{n/d} \leq w$.*

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 $jd+1 \leq k \leq (j+1)d$. Then $w_{(d)} > w[jd+1..(j+1)d]$. Hence, $rot(w, jd) < w$, a contradiction. □

In the ranking algorithms that we design below we need an assumption that the input word is self-minimal. This makes the following auxiliary lemma a useful tool.

Fact 5. [see [6]] *For a given word $x \in \Sigma^n$, $\langle x \rangle$ can be computed in $\mathcal{O}(n)$ time.*

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 5 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. Also, let $z = \max \Sigma$.

Consider the word $w'' = w_{(k-1)}b'z^{n-k}$. Note that $w' \leq w'' < w$. If $w' < w''$, then the following claim applied for $x = w'$ and $x' = w''$ would show that $\langle w'' \rangle = w''$, and this would contradict the definition of w'. Hence, $w' = w''$.

Claim. Let $x = ycu$ and $x' = ydv$, where $c, d \in \Sigma$, $c < d$, $u \in \Sigma^*$, and $v = z^{|u|}$ where $z = \max \Sigma$. If x is self-minimal, then so is x'.

Proof (of the claim). Denote $n = |x| = |x'|$. First, note that $x[1] < z$. Indeed, if $x[1] = z$, then we would have $rot(x, |y|) = cuy < x$, which is not possible. Hence, if $x'[1] = z$, then $|y| = 0$, $d = z$ and $x' = z^n$, and the lemma trivially holds. From now on we assume that $x'[1] < z$.

Denote $m = |u|$. We have $x' < z \leq z^\ell ydz^{m-\ell}$ for any $\ell \in \{1, \ldots, m\}$. Thus it suffices to show that for any factorization $y = y_1y_2$ we have $ydv \leq y_2dvy_1$. If $y = y_2$ this is trivial, so assume $|y| > |y_2|$. For a proof by contradiction assume $ydv > y_2dvy_1$. Since $d > c$ and $\langle x \rangle = x$, we have

$$ydv > y_2dvy_1 > y_2cuy_1 \geq ycu = x.$$

This means that y_2dvy_1 and y_2cuy_1 both start with y, which is a contradiction since their longest common prefix y_2 is shorter than y. □

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 5. □

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. The shortest word u such that $w = u^k$ for some positive integer k is called the *primitive root* of w. The primitive root of a word of length n can be computed in $\mathcal{O}(n)$ time; see [3].

We say that $\lambda \in \Sigma^*$ is a *Lyndon word* if it is primitive and self-minimal. An equivalent definition is that a Lyndon word is smaller than all its suffixes. 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 λ being a Lyndon word [17]. Below we show an additional property of Lyndon words that will be useful in Section 7.

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

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

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

Proof. (a) 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:

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

If $|x| < |\lambda_1|$, then we obtain $x < \lambda_1$. 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.

(b) Assume 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 (a). □

3 Combinatorics of Ranking Lyndon Words

Our basic goal is to compute *LynRank*(w), that is, the number of Lyndon words in Σ^n not exceeding w ($n = |w|$). It suffices to compute *LynRank*(w) for a self-minimal word w. If w is not self-minimal, then *LynRank*(w) = *LynRank*(w') where w' is the greatest self-minimal word such that $w' \leq w$; w' can be computed efficiently using Lemma 6.

We will show how to reduce computation of *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 $u = \mathtt{abba}$ and $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $\mathsf{S}(u)$ is the set of all cyclic rotations of the words:

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

that is:

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

Let us introduce the following auxiliary sets for $\ell \mid n$:

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

Example 9. For $w = \mathtt{abba}$ and $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $\mathsf{S}_2(w) = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}\}$ and $\mathsf{S}'_2(w) = \{\mathtt{ab}, \mathtt{ba}\}$.

These sets are closely related to the basic sets $\mathsf{S}(v)$. It is most evident for a self-minimal word.

Observation 10. *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 $d < n$. Let us prove the equality by showing both inclusions.

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

Now assume that $x \in \mathsf{S}(w_{(d)})$. This means that $\langle x \rangle \leq w_{(d)}$. We have: $\langle x \rangle^{n/d} \leq (w_{(d)})^{n/d} \leq w$ where the second inequality is due to Observation 4. Hence, $x \in \mathsf{S}_d(w)$. □

The sets that we have just introduced provide a formula for *LynRank*.

Lemma 11. *If $w \in \Sigma^n$ is self-minimal, then*

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

Proof. The proof is divided into two claims that reduce computation of $LynRank(w)$ to the computation of the sizes of some number of sets $\mathsf{S}(v)$. Actually in both claims we do not need the assumption that w is self-minimal; we only add it at the very end.

Claim. $LynRank(w) = \frac{1}{n}|\mathsf{S}'_n(w)|$.

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. □

The following general claim enables us to compute $|\mathsf{S}'_n(w)|$ by computing the sizes of a number of sets S_d.

Claim. If $\ell \mid n$, then $|\mathsf{S}'_\ell(w)| = \sum_{d|\ell} \mu(\frac{\ell}{d})|\mathsf{S}_d(w)|$.

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, by Möbius inversion formula, we obtain the claim. □

Now the formula for $LynRank(w)$ combines the formulas from the respective claims and Observation 10. □

Example 12. Let $w = \mathtt{ababbb}$. We have $w_{(1)} = \mathtt{a}$, $w_{(2)} = \mathtt{ab}$, $w_{(3)} = \mathtt{aba}$ and

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

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

The set of Lyndon words that are not greater than w contains the following words:

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

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

Lemma 13. *For a self-minimal word $w \in \Sigma^n$ one can compute $|\mathsf{S}(w)|$:*

(a) in $\mathcal{O}(n^2)$ time in the unit-cost RAM,

(b) 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 14. *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 15. *For an arbitrary word w one can compute $LynRank(w)$ 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 11 and the algorithm of Lemma 13. 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 14, reduces to $\mathcal{O}(n^2 \log \sigma)$ in the word-RAM or $\mathcal{O}(n^2)$ in the unit-cost RAM, respectively. $\square$

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

Theorem 16. *The k-th 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 we look for the smallest $w \in \Sigma^n$ such that $LynRank(w) \geq k$. We binary search Σ^n with respect to the lexicographic order, using the algorithm of Theorem 15 to check whether $LynRank(w) \geq k$. The size of the search space is σ^n, which gives an additional $n \log \sigma$-time factor. $\square$

4 Automata-Theoretic Interpretation

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

Let $\mathrm{Pref}_-(w) = \{w_{(i)}s : i \in \{0, \ldots, n-1\}, s \in \Sigma, s < w[i+1]\} \cup \{w\}$. 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 $\sqrt{L} = \{x : x^2 \in L\}$.

Observation 17. $\mathsf{S}(w) = \sqrt{L(w)} \cap \Sigma^n$.

Proof. Consider a word $x \in \Sigma^n$. If $x \in \mathsf{S}(w)$, then $\langle x \rangle \leq w$. 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 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 $x' \leq w$, hence $\langle x \rangle \leq x' \leq w$ and $x \in \mathsf{S}(w)$. $\square$

We construct a deterministic finite automaton $A = (Q, q_0, F, \delta)$ 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

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

Fig. 1 contains an example of such an automaton.

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

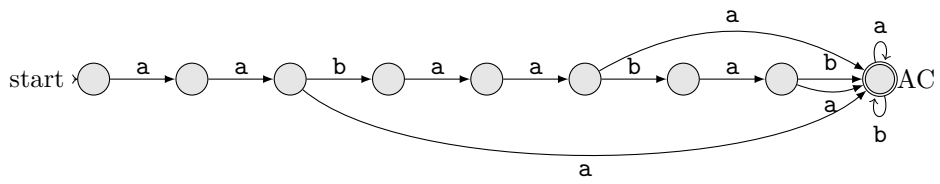


Figure 1: Automaton A that accepts $L(w)$ for a word $w = \texttt{aabaabab}$ and alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$. Missing links lead to the initial state.

prefixes of w and eventually goes either back to $w_{(0)}$ or to 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 18. *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

$$w_{(j)}w[i+1,n]w_{(i-j)} \geq \langle w \rangle = w = w_{(j)}w[j+1,n], \quad \text{hence } c \geq w[i+1] \geq w[j+1].$$

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 different states, so A is sparse.

The following corollary summarizes the construction of A.

Corollary 19. *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 automaton into words:

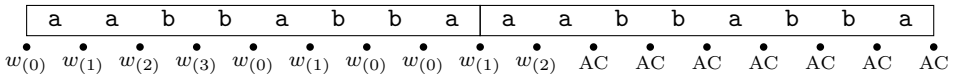
$$\delta(q, x) = \delta(\ldots\delta(\delta(q, x[1]), x[2])\ldots, x[k]) \quad \text{for } x \in \Sigma^k.$$

For states $q, q' \in Q$ let us define $L_A(q, q') = \{x \in \Sigma^* : \delta(q, x) = q'\}$. 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.

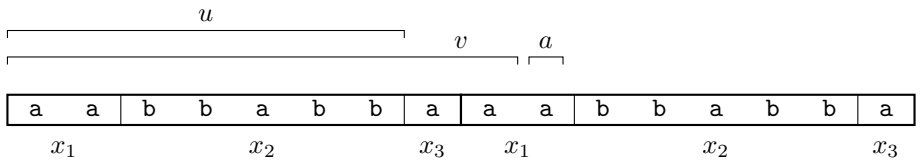
Lemma 20. *Assume $|x| = n$ and $x^2 \in L(A)$ but $x \notin L(A)$. Then there is a unique decomposition $x = x_1x_2x_3$ such that $x_1, x_3 \neq \varepsilon$, $x_3x_1 \in \mathrm{Pref}_{\_}(w)$ and $x_1x_2 \in L_A(w_{(0)}, w_{(0)})$.*

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 behaviour of the automaton. □

Example 21. Let $w = \mathtt{aabaabab}$. Recall that the automaton A such that $L(A) = L(w)$ was shown in Fig. 1. Consider the word $x = \mathtt{aabbabba}$ 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 20 is as follows:



In this case in the proof of the lemma we have: $u = \mathtt{aabbabb}$, $v = \mathtt{aabbabbaa}$, 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, in other words, it is polynomial with respect to $n + \sigma$. Using Lemma 20 we obtain a formula for $|\mathsf{S}(w)|$.

Lemma 22. *If $w \in \Sigma^n$ is self-minimal, then*

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

The coefficients $\alpha_{i,j}$ are small numbers and can all be computed in $\mathcal{O}(n^2)$ total time.

Proof. We apply Observation 17 with Corollary 19 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 20. 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 and x_1' are uniquely determined. Let us define ℓ as $w_{(\ell)} = \delta(w_{(0)}, x_1')$; see Fig. 2.

We need to count ax_2 such that $|x_2| = n - k - 1$, $ax_2 \in L_A(w_{(\ell)}, w_{(0)})$ and $\delta(w_{(k)}, a) = AC$. 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

$$\sum_{q \in \{0, \ell+1\}} \gamma(k, \ell, q)\, \pi_{n-k-1}(q, 0), \quad \text{where } \gamma(k, \ell, q) = |\{a \in \Sigma : \delta(w_{(\ell)}, a) = q \wedge \delta(w_{(k)}, a) = AC\}|. \tag{2}$$

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. □

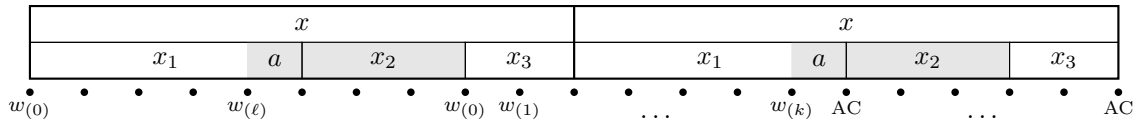


Figure 2: Illustration of Lemma 22. Both lines represent different factorizations of the same word x^2. Black circles represent states of the automaton. Only shaded letters are not necessarily uniquely determined by $|x_3|$ and $|x_1|$ for a fixed w.

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 simulate the automaton A recognizing $L(w)$. Its proof is based on matrix multiplication.

Lemma 23. *Let $A = (Q, q_0, F, \delta)$ be a sparse deterministic automaton with n states, and let $m \in \mathbb{Z}_{\geq 0}$. It takes $\mathcal{O}(mn)$ arithmetic operations on integers of magnitude σ^m to compute all values $|L_A(q, q') \cap \Sigma^k|$, $k \leq m$, for a fixed state q or 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 are in $\{0, \ldots, \sigma^k\}$.

Note that the matrix M is sparse, i.e. it contains $\mathcal{O}(n)$ non-zero entries. Consequently, 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 $\mathbf{e}_q^T M^k$, and simultaneously the q-th entry of $M^k \mathbf{e}_{q'}$. If q (or q') is fixed, we can compute these vectors for all $k \leq m$ using m vector-matrix multiplications. In total we perform $\mathcal{O}(mn)$ arithmetic operations. □

The algorithm below combines the results obtained so far to provide the first implementation of Lemma 13(a).

Proof (of Lemma 13(a)). Lemma 23 can be used to implement the formula of Lemma 22; see the algorithm below.

Algorithm *Computing $|\mathsf{S}(w)|$ in $\mathcal{O}(n^2)$ time in unit-cost RAM model*
Construct automaton A for w { Corollary 19 }
Compute $\pi_j(0, i)$ and $\pi_j(i, 0)$ for all $0 \leq i, j \leq n$ { Lemma 23 }
Compute $\alpha_{i,j}$ coefficients { Lemma 22 }
$|\mathsf{S}(w)| := \pi_n(0, n) + \sum_{i,j=0}^{n} \alpha_{i,j} \pi_j(i, 0)$ { Lemma 22 }

The algorithm performs $\mathcal{O}(n^2)$ arithmetic operations 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 22, 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 ([23]; see also [11], p. 140):

Fact 24. *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 n-th coefficient of $F(x)$.

For an efficient implementation of Fact 24 we use an integer multiplication algorithm that works in linear time in the word-RAM model; see Fürer [10].

Lemma 25. *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 i-th least significant chunk of u (respectively v) holds the i-th coefficient of $F(x)$ (respectively $G(x)$) appended from the front by zeroes. Then the corresponding chunks of uv hold the coefficients of $F(x)G(x)$. Each number u, v has length $\mathcal{O}(n^2 \log \sigma)$, therefore the product uv can be computed in $\mathcal{O}(n^2 \log \sigma)$ time [10]. □

With the auxiliary Fact 14 we obtain the following tool:

Lemma 26. *Let $F(x)$ and $G(x)$ be power series such that $F(x)G(x) \equiv 1$. Assume that the k-th 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 26 to count specific paths in the automaton A for the word w. Denote

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

Lemma 27. *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 non-empty 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:

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

Let us set $a_0 = -1$. Let F and G be 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 26 to compute n first coefficients of $F(x)$ in $\mathcal{O}(n^2 \log \sigma)$ time. □

We extend the results of the previous lemma in the following way.

Lemma 28. *$\pi_n(0,n)$ 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 $a'_i = \sigma - 1 - a_i$. Note that for every $j = 1, \ldots, n$:

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

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 c_j's can be computed in $\mathcal{O}(n^2)$ time. By Lemma 27, all T_j's can be computed in $\mathcal{O}(n^2 \log \sigma)$ time. Obviously $c_j, T_j \leq \sigma^j$. This concludes that we can use the algorithm of Lemma 25 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 the remaining part of Lemma 13 related with efficient computation of $|\mathsf{S}(w)|$ in the word-RAM model.

Proof (of Lemma 13(b)). We provide an efficient implementation of the formula from Lemma 22. For the $\pi_n(0,n)$ part we use Lemma 28. 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.

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

The formula 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 **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 be never changed later. This concludes that at the end of the algorithm we have:

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

Note that each $\alpha_{i,j}$ coefficient accounts in $\sum_j \beta_{0,j}$ as at most $(a_{i+1} + a_{i+2} + \ldots + a_n)\alpha_{i,j}$. Hence, the sum of the resulting non-zero coefficients $\beta_{i,j}$ does not exceed σn times the sum of the initial $\alpha_{i,j}$'s. At the end we are to compute a linear combination of T_j with small coefficients. Consequently Lemma 27 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 $Ł^{(n)}$ we denote the set of Lyndon words over Σ whose length is a divisor of n. Theorem of Fredricksen and Maiorana [9, 14] states that dB_n is a concatenation of the Lyndon words from $Ł^{(n)}$ 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 $Ł^{(n)}$ 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 29. *Let $w \in \Sigma^n$ and $Ł(w) = \{\lambda \in Ł^{(n)} : \lambda^{n/|\lambda|} \leq w\}$. Then the concatenation, in lexicographic order, of words $\lambda \in Ł(w)$ forms a prefix of dB_n and its length, $\sum_{\lambda \in Ł(w)} |\lambda|$, is equal to $|\mathsf{S}(w)|$.*

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

It remains to show that $\sum_{\lambda \in Ł(w)} |\lambda| = |\mathsf{S}(w)|$. For this we shall build a mapping $\phi : \Sigma^n \to Ł^{(n)}$ such that $|\phi^{-1}(\lambda)| = |\lambda|$ and $\langle x \rangle \leq w$ for $x \in \Sigma^n$ if and only if $\phi(x) \in Ł(w)$.

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 $Ł^{(n)}$. 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. $\phi(x) \in Ł(w)$. □

Theorem 30. *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 $\lambda_1 < \lambda_2 < \ldots < \lambda_p$ be all Lyndon words in $Ł^{(n)}$ (we have $\lambda_1\lambda_2 \ldots \lambda_p = dB_n$). The proof of theorem of Fredricksen and Maiorana [9, 14] describes the occurrence of w in dB_n which can be stated succinctly as follows.

Claim ([9, 14]). Assume that $w = (\alpha\beta)^d$, where $d \in \mathbb{Z}_+$ and $\beta\alpha = \lambda_k \in Ł^{(n)}$. Denote $a = \min \Sigma$ and $z = \max \Sigma$.

(a) If $w = z^i a^{n-i}$ for $i \geq 1$, then w occurs in dB_n at position $\sigma^n - i$.

(b) If $\alpha \neq z^{|\alpha|}$, 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 $\lambda \in \text{Ł}^{(n)}$ such that $\lambda < \beta$.

In case (a) it is easy to locate w in dB_n. Further on we consider only the cases (b), (c), (d).

Observe that λ_k can be retrieved as the primitive root of $\langle w \rangle$. Also note that, by Observation 7(a), $\lambda_{k'}$ 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 $\text{Ł}^{(n)}$ and possibly the predecessor in $\text{Ł}^{(n)}$ of one of them. For any $\lambda \in \text{Ł}^{(n)}$ the successor in $\text{Ł}^{(n)}$ can be generated by iterating a single step of the FKM algorithm at most $(n-1)/2$ times [8], i.e. in $\mathcal{O}(n^2)$ time. For the predecessor in $\text{Ł}^{(n)}$, a version of the FKM algorithm that visits the Lyndon words in reverse lexicographic order can be used [14]. 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 29 proves that such an occurrence of $\lambda \in \text{Ł}^{(n)}$ *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 13. Applied to λ_k or $\lambda_{k'}$ this concludes the proof. □

Example 31. Below we present the four cases of the claim in the proof of Theorem 30 on an example minimal binary de Bruijn sequence of rank 6, which has the following decomposition $\lambda_1, \lambda_2, \ldots, \lambda_{14}$ into Lyndon words:

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	λ_{14}
0	00	0001	00	0011	00	0101	0001	11	001	0	01011	001101	001111	01	0101	11	011	0	111	11	1
				(b) 001100				(d) 110010								(c) 110110				(a) 111000	

Case (a): *occ-pos*$(111000, dB_6) = 62$, and 111000 appears as a factor of $\lambda_{13}\lambda_{14}\lambda_1\lambda_2$

Case (b): *occ-pos*$(001100, dB_6) = 10$, and 001100 appears as a factor of $\lambda_3\lambda_4$

Case (c): *occ-pos*$(110110, dB_6) = 53$, and 110110 appears as a factor of $\lambda_{11}\lambda_{12}\lambda_{13}$

Case (d): *occ-pos*$(110010, dB_6) = 24$, and 110010 appears as a factor of $\lambda_5\lambda_6\lambda_7$.

To compute the k-th symbol of dB_n we have to locate the Lyndon word from $\text{Ł}^{(n)}$ containing the k-th position of dB_n. We apply binary search as in Theorem 16. The k-th symbol of dB'_n is much easier to find due to a simpler structure of the sequence, as shown in the proof of the following theorem.

Theorem 32. *Given integers n and k, the k-th symbol of dB_n and 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 13 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 $\lambda \in \text{Ł}^{(n)}$. By Lemma 29, a prefix of dB_n of length $|\mathsf{S}(v)|$ contains all Lyndon words from $\text{Ł}(v)$. Moreover, by Observation 7(a), this prefix ends with λ. This means that the k-th position of dB_n lies within the canonical occurrence of λ. More precisely, it suffices

to return the $(|\mathsf{S}(v)| - k + 1)$-th *last* symbol of λ (which is also the $(|\mathsf{S}(v)| - k + 1)$-th last symbol of v). As in Theorem 16, the binary search introduces a multiplicative $\mathcal{O}(n \log \sigma)$ factor to the complexity of the algorithm of Lemma 13.

The k-th symbol of the sequence dB'_n is the i-th symbol of the j-th 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 16. □

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 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 [16].

9 Acknowledgements

We would like to thank Joe Sawada for making us aware of the work of Kopparty et al. [16].

References

[1] Silvia Bonomo, Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffixes, conjugates and Lyndon words. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory*, volume 7907 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 2013.

[2] Fan Chung, Persi Diaconis, and Ron Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110:43–59, 1992.

[3] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[4] Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014.

[5] Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, 1994.

[6] Jean-Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983.

[7] Jean-Pierre Duval. Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. *Theoretical Computer Science*, 60:255–283, 1988.

[8] Harold Fredricksen and Irving J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Mathematics*, 61(2-3):181–188, 1986.

[9] Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Mathematics*, 23(3):207–210, 1978.

[10] Martin Fürer. How fast can we multiply large integers on an actual computer? In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium*, volume 8392 of *Lecture Notes in Computer Science*, pages 660–670. Springer, 2014.

[11] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra*. Springer Science & Business Media, 2007.

[12] Torben Hagerup. Sorting and searching on the word RAM. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 366–398. Springer, 1998.

[13] Yu Hin Au. Generalized de Bruijn words for primitive words and powers. *ArXiv e-prints*, June 2015.

[14] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2*. Addison-Wesley, 2005.

[15] Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Computing k-th Lyndon word and decoding lexicographically minimal de Bruijn sequence. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *Combinatorial Pattern Matching, CPM 2014*, volume 8486 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 2014.

[16] Swastik Kopparty, Mrinal Kumar, and Michael Saks. Efficient indexing of necklaces and irreducible polynomials over finite fields. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 726–737. Springer, 2014.

[17] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, MA., U.S.A., 1983.

[18] Chris J. Mitchell, Tuvi Etzion, and Kenneth G. Paterson. A method for constructing decodable de Bruijn sequences. *IEEE Transactions on Information Theory*, 42(5):1472–1478, 1996.

[19] Marcin Mucha. Lyndon words and short superstrings. In Sanjeev Khanna, editor, *SODA*, pages 958–972. SIAM, 2013.

[20] Jakub Radoszewski. Generation of lexicographically minimal de Bruijn sequences with prime words. Master's thesis, University of Warsaw, 2008 (in Polish).

[21] Frank Ruskey, Carla D. Savage, and Terry Min Yih Wang. Generating necklaces. *Journal of Algorithms*, 13(3):414–430, 1992.

[22] Joe Sawada and Aaron Williams, June 2015. Personal communication.

[23] M. Sieveking. An algorithm for division of powerseries. *Computing*, 10(1-2):153–156, 1972.

[24] Jonathan Tuliani. De Bruijn sequences with efficient decoding algorithms. *Discrete Mathematics*, 226(1-3):313–336, 2001.