Almost Linear Time Computation of Maximal Repetitions in Run Length Encoded Strings

Yuta Fujishige[1], **Yuto Nakashima**[2], **Shunsuke Inenaga**[3], **Hideo Bannai**[4], and **Masayuki Takeda**[5]

1 **Department of Informatics, Kyushu University, Japan**
`yuta.fujishige@inf.kyushu-u.ac.jp`
2 **Department of Informatics, Kyushu University, Japan**
`yuto.nakashima@inf.kyushu-u.ac.jp`
3 **Department of Informatics, Kyushu University, Japan**
`inenaga@inf.kyushu-u.ac.jp`
4 **Department of Informatics, Kyushu University, Japan**
`bannai@inf.kyushu-u.ac.jp`
5 **Department of Informatics, Kyushu University, Japan**
`takeda@inf.kyushu-u.ac.jp`

Abstract

We consider the problem of computing all maximal repetitions contained in a string that is given in run-length encoding. Given a run-length encoding of a string, we show that the maximum number of maximal repetitions contained in the string is at most $m+k-1$, where m is the size of the run-length encoding, and k is the number of run-length factors whose exponent is at least 2. We also show an algorithm for computing all maximal repetitions in $O(m\alpha(m))$ time and $O(m)$ space, where α denotes the inverse Ackermann function.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms

Keywords and phrases maximal repetitions, run length encoding

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.33

1 Introduction

Periodicity and repetitions in strings are one of the most important characteristic features in strings. They have been one of the first objects of study in the field of combinatorics on words [25] and have many theoretical, as well as practical applications, e.g., in bioinformatics [14].

Maximal repetitions are periodically maximal substrings of a string where the smallest period is at most half the length of the substring, i.e., there are at least two consecutive occurrences of the same substring. An $O(n \log n)$ time algorithm for computing all of the maximal repetitions contained in a string of length n, was shown by Main and Lorentz [24], which is optimal for general unordered alphabets, i.e., when only equality comparisons between the letters are allowed. Kolpakov and Kucherov [15] further showed that the number of maximal repetitions is actually $O(n)$, and gave a linear time algorithm for ordered constant size alphabets (and essentially for integer alphabets), to compute all of them. The algorithm was a modification of the algorithm by Main [23], which in turn relies on the Lempel-Ziv 77 (LZ77) factorization [27] of the string, which can be computed in linear time for ordered constant size or integer alphabets [8], but requires $\Omega(n \log \sigma)$ time for general ordered alphabets [16], where σ is the size of the alphabet. Recently, a new characterization of maximal repetitions using Lyndon words was proposed by Bannai et al. [2, 3], which lead to a very simple proof to what was known as the “runs” conjecture, i.e., that the number of maximal repetitions in a given string of length n is less than n [3]. The characterization also lead to a new linear time algorithm for computing maximal repetitions on ordered constant size and integer alphabets, which does not require the LZ77 factorization, but only on a linear number of *longest common extension queries*. Furthermore, based on this algorithm, the running time for computing all maximal repetitions for general ordered alphabets were subsequently improved to $O(n \log^{2/3} n)$ by Kosolobov [17], $O(n \log\log n)$ by Gawrychowski et al. [12], and $O(n\alpha(n))$ by Crochemore et al. [9], where α denotes the inverse Ackermann function.

In this paper, we consider the problem of computing all maximal repetitions contained in a string when given the *run-length encoding* (RLE) of the string, which is a well known compressed representation where each maximal substring of the same character is encoded as a pair consisting of the letter and the length of the substring. For example, the run-length encoding of the string `aaaabbbaaacc` is $(\mathtt{a},4)(\mathtt{b},3)(\mathtt{a},3)(\mathtt{c},2)$. The main contributions of the paper are:

1. an upper bound $m + k - 1$ on the number of maximal repetitions contained in a string, where m is the size of its run-length encoding and k is the number of run-length factors whose exponent is at least 2, and
2. an $O(m\alpha(m))$ time and $O(m)$ space algorithm to compute all maximal repetitions in a string.

Our algorithm is at least as efficient as the non-RLE algorithms for general ordered alphabets. Furthermore, when the input string is compressible via RLE, our algorithm can be faster and more space efficient compared to the non-RLE algorithms. Although our algorithm mimics those for non-RLE strings and is conceptually simple, its correctness is based on new non-trivial observations on the occurrence of specific Lyndon words in run-length encoded strings.

Efficient algorithm for string problems when the input is given in RLE has been considered in various contexts, for example, edit distance [6], various Longest Common Subsequence problems [20, 18], palindrome retrieval [7], computing Lempel Ziv factorization [26], etc. We shall repeat below a claim made in [18] concerning the significance of RLE-based solutions:

> “A common criticism against RLE based solutions is a claim that, although they are theoretically interesting, since most strings “in the real world” are not compressible by RLE, their applicability is limited and they are only useful in extreme artificial cases. We believe that this is not entirely true. There can be cases where RLE is a natural encoding of the data, for example, in music, a melody can be expressed as a string of pitches and their duration. Furthermore, in the data mining community, there exist popular preprocessing schemes for analyzing various types of time series data, which convert the time series to strings over a fairly small alphabet as an approximation of the original data, after which various analyses are conducted (e.g. SAX (Symbolic Aggregate approXimation) [19], *clipped* bit representation [1], etc.). These conversions are likely to produce strings which are compressible by RLE (and in fact, shown to be effective in [1]), indicating that RLE based solutions may have a wider range of application than commonly perceived.”

2 Preliminaries

2.1 Strings

Let Σ denote the alphabet, i.e. the set of letters (or characters). An element of Σ^* is called a string. For any strings $x, y \in \Sigma^*$, xy represents their concatenation. If $w = xyz$ for any strings $w, x, y, z \in \Sigma^*$, x, y, z are respectively called a prefix, substring, suffix of w. A prefix x and a suffix z of w are respectively called a proper prefix, and proper suffix of w, if $x \neq w$ and $z \neq w$. A string x is called a border of w, if it is a proper suffix as well as a prefix of w. The length of a string w is denoted as $|w|$. The empty string is a string of length 0 and will be denoted by ε. For any $1 \leq i \leq j \leq |w|$, $w[i]$ denotes the ith letter of w, and $w[i..j] = w[i] \cdots w[j]$. For convenience, let $w[i..j] = \varepsilon$ when $i > j$. For any integer $k \geq 0$ and string $x \in \Sigma^*$, $x^0 = \varepsilon$, and $x^k = x^{k-1}x$.

We assume a general ordered alphabet, where a total order $\prec$ is defined on Σ, and the order between two letters in the alphabet can be computed in constant time. A total order $\prec$ on the alphabet induces a total order on the set of strings called the *lexicographic order*, which we also denote by $\prec$, i.e., for any $x, y \in \Sigma^*$, $x \prec y \iff x$ is a proper prefix of y, or, there exists $1 \leq i \leq \min\{|x|, |y|\}$ s.t. $x[1..i-1] = y[1..i-1]$ and $x[i] \prec y[i]$.

All previous linear time algorithms either assume a constant size ordered alphabet or an integer alphabet, i.e., $\Sigma = \{1, \ldots, n^c\}$ for some constant c. We will later see that this assumption does not help in our case.

2.2 Maximal Repetitions

For any string $w \in \Sigma^*$, an integer $1 \leq p < |w|$ is called a *period* of w if $w[i] = w[i+p]$ for all $1 \leq i \leq |w| - p$. A string whose smallest period is at most half its length is called a *repetition*. We are interested in occurrences of repetitions as a substring of a given string which are periodically maximal. Specifically, a triplet $r = (i, j, p)$ is called a *maximal repetition* of w, if and only if all the following hold:

1. p is the smallest period of $w[i..j]$ and $|w[i..j]| \geq 2p$ (repetition),
2. $i = 1$ or $w[i-1] \neq w[i-1+p]$ (left maximal), and
3. $j = |w|$ or $w[j+1] \neq w[j+1-p]$ (right maximal).

For any string w, we denote the set of maximal repetitions as $\mathit{MReps}(w)$. Although maximal repetitions are commonly referred to as "runs" in the literature, we use the term "maximal repetitions" so as not to confuse it with "run" in "run-length encoding".

For example, the string $w =$ `abaababaabaab` contains seven maximal repetitions, i.e., $\mathit{MReps}(w) = \{(3,4,1), (8,9,1), (11,12,1), (4,8,2), (1,6,3), (6,13,3), (1,11,5)\}$.

2.3 Run Length Encoding

Let $\mathcal{N}$ denote the set of positive integers. For any string $w \in \Sigma^*$, let $a_i \in \Sigma$ and $e_i \in \mathcal{N}$, for $1 \leq i \leq m$, be such that $w = a_1^{e_1} \cdots a_m^{e_m}$ and $a_i \neq a_{i+1}$ for all $1 \leq i < m$. The *run-length encoding* $\mathit{RLE}(w)$ of string w is a string over the alphabet $\Sigma \times \mathcal{N}$, and is defined as $\mathit{RLE}(w) = (a_1, e_1) \cdots (a_m, e_m)$. For any $1 \leq i \leq m$, each letter $\mathit{RLE}(w)[i] = (a_i, e_i)$ and its corresponding substring $a_i^{e_i}$ in w is called a run-length factor, and e_i is called its exponent.

The set of starting (resp. ending) positions of run-length factors of w is denoted by S_w (resp. E_w), i.e., $S_w = \{1 + \sum_{k=1}^{i-1} e_k : 1 \leq i \leq m\}$ and $E_w = \{\sum_{k=1}^{i} e_k : 1 \leq i \leq m\}$. We will also write $S_w[i] = 1 + \sum_{k=1}^{i-1} e_k$ and $E_w[i] = \sum_{k=1}^{i} e_k$ for any $1 \leq i \leq m$.

2.4 Lyndon Words

A string w is a *Lyndon word* [22] with respect to lexicographic order $\prec$, if and only if $w \prec w[i..|w|]$ for any $1 < i \leq |w|$, i.e., w is lexicographically smaller than any of its proper suffixes with respect to $\prec$. It is easy to see that a Lyndon word w cannot have a non-empty border, since a border would be a proper suffix of w that is lexicographically smaller than w, since it is also a prefix of w. An equivalent definition for a Lyndon word, is a word which is lexicographically smaller than any of its proper cyclic rotations.

For example, if $\mathtt{a} \prec \mathtt{b}$, then, the string `abaabb`, `baa`, `abab` are not Lyndon words with respect to $\prec$, while `aabab` is. The following is also well known.

▶ **Lemma 1** (Proposition 1.3 [10]). *For any Lyndon words u and v, uv is a Lyndon word iff $u \prec v$.*

2.5 Longest Common Extension

For any string w of length n, the *longest common extension query* is, given two positions $1 \leq i, j \leq n$, to answer

$$LCE_w(i,j) = \max\{k \mid w[i..i+k-1] = w[j..j+k-1], i+k-1, j+k-1 \leq n\}.$$

We also define the longest common extension in the reverse direction, i.e.,

$$LCE^R_w(i,j) = \max\{k \mid w[i-k+1..i] = w[j-k+1..j], i-k+1, j-k+1 \geq 1\}.$$

Note that if there is a way to compute $LCE_w(i,j)$ given w, there is also a way to compute $LCE^R_w(i,j)$ by considering the reversed string $w^R = w[n] \cdots w[1]$, since $LCE^R_w(i,j) = LCE_{w^R}(n-i+1, n-j+1)$.

3 The Maximum Number of Maximal Repetitions by RLE

The goal of this section is to prove the following Theorem.

▶ **Theorem 2.** *For any string w, let m be the size of its run-length encoding, and k the number of run-length factors of w whose exponent is at least 2. Then, $|MReps(w)| \leq m+k-1$.*

The proof basically follows the idea of [3] for normal strings, but it is extended to deal with RLE strings.

For any maximal repetition $r = (i, j, p)$ of string w and any lexicographic order $\prec$, there exists a substring of length p in $w[i..j]$ that is a Lyndon word with respect to $\prec$. This is because the set $\{w[i'..i'+p-1] \mid i+1 \leq i' \leq i+p\}$ contains all p cyclic rotations of $w[i+1..i+p]$ which are all distinct, since p is the smallest period of w, and a lexicographically smallest rotation will always exist. Any length p subinterval $[\ell, \ell+p-1]$ of a maximal repetition $r = (i, j, p)$ such that $w[\ell..\ell+p-1]$ is a Lyndon word with respect to $\prec$, is called an *L-root* of r with respect to $\prec$.

Theorem 2 is trivial when $|\Sigma| = 1$, so we can assume $|\Sigma| \geq 2$, and thus, we are able to consider two orderings denoted by $\prec_0$ and $\prec_1$, where $\prec_0 = \prec$ and for any $a, b \in \Sigma$, $a \prec_0 b \iff b \prec_1 a$. We also use $\prec_0$ and $\prec_1$ to denote the lexicographic orders on Σ^* induced by the respective total orders. As in [3], we choose, for each maximal repetition $r = (i, j, p)$, a specific lexicographic order denoted by $\prec_r \in \{\prec_0, \prec_1\}$ so that $w[j+1] \prec_r w[j+1-p]$. We note that either order can be chosen when $j = n$. The set B_r is defined as the beginning positions of L-roots of r with respect to this order, but excludes a position if it coincides with the beginning position of the maximal repetition, i.e., for any maximal repetition $r = (i, j, p)$,

$$B_r = \{\ell \mid [\ell..\ell + p - 1] \text{ is an L-root of } r \text{ w.r.t. } \prec_r \text{ , and } \ell \neq i\}.$$

Note that $|B_r| \geq 1$ since a maximal repetition always contains an L-root that does not start at its beginning. One of the crucial results of [3] was the following lemma, which implies that the number of maximal repetitions in a string w of length n is at most $n - 1$ since $\cup_{r \in MReps(w)} B_r \subseteq [2..n]$ and thus $|MReps(w)| \leq \sum_{r \in MReps(w)} |B_r| \leq n - 1$.

▶ **Lemma 3** (Lemma 8 of [3]). *For any distinct maximal repetitions r, r' of w, $B_r \cap B_{r'} = \emptyset$.*

The following lemma is an important new observation for L-roots of maximal repetitions with respect to their run-length encoding.

▶ **Lemma 4.** *For any maximal repetition $r = (i, j, p)$ of string w with $p \geq 2$, it holds that $B_r \subset S_w$, i.e., a position in B_r must be the beginning of an RLE-factor.*

Proof. Suppose to the contrary, that there is some $\ell \in B_r$ that is not at the beginning of an RLE-factor, i.e., $\ell \notin S_w$, and let $[\ell..\ell + p - 1]$ be the corresponding L-root of r. By the assumption, $w[\ell - 1] = w[\ell]$. Furthermore, by the definition of B_r, we have that $i < \ell$ and by the periodicity of r, $w[\ell - 1] = w[\ell + p - 1]$. However, this implies that $w[\ell..\ell + p - 1]$ has a border, contradicting that it is a Lyndon word. The lemma holds, since $1 \in S_w$ but $1 \notin B_r$. ◀

Of course, a run-length factor can be a maximal repetition of period 1, and can be stated as follows.

▶ **Lemma 5.** *For any string w, let $RLE(w) = (a_1, e_1) \cdots (a_m, e_m)$. For any $1 \leq i \leq m$, $(S_w[i], E_w[i], 1)$ is a maximal repetition of period 1 if and only if $e_i \geq 2$.*

We are now ready to prove Theorem 2.

Proof of Theorem 2. Recall that k is the number of run-length factors of w whose exponent is at least 2. Due to Lemma 5, the number of maximal repetitions with period 1 is equal to k. Note that for any maximal repetition r of period 1, any position $i \in B_r$ satisfies $i \notin S_w$. Let $MReps_{p \geq 2}(w)$ be the set of maximal repetitions such that the period is at least 2. From $|B_r| \geq 1$ and Lemmas 3 and 4,

$$|MReps_{p \geq 2}(w)| \leq \sum_{r \in MReps_{p \geq 2}(w)} |B_r| \leq m - 1 < m = |S_w|$$

holds. Thus, the total number of maximal repetitions is at most $m + k - 1$. ◀

If we consider the 2 cases w.r.t. m, we can get better bounds for each of 2 cases. Corollary 6 is the tight bound for smaller m. Corollary 7 is a improved bound for larger m.

▶ **Corollary 6.** *For any string w, let m be the size of its run-length encoding. If $m \leq 3$, $|MReps(w)| \leq m$.*

If $m = 3$, it is easy to see that $|MReps_{p \geq 2}(w)| = 0$. Obviously, $|MReps(w)| = k$ also holds, where k is the number of run-length factors of w whose exponent is at least 2.

▶ **Corollary 7.** *For any string w, let m be the size of its run-length encoding, and k the number of run-length factors of w whose exponent is at least 2. If $m \geq 4$, $|MReps(w)| \leq m + k - 3$.*

Proof. Since an L-root of $r \in \mathit{MReps}_{p\geq 2}(w)$ must contain at least two different characters, the beginning position $S_w[m]$ of the last run-length factor (a_m, e_m) cannot be in B_r for any $r \in \mathit{MReps}_{p\geq 2}(w)$. If $S_w[m-1] \in B_r$ for some $r \in \mathit{MReps}_{p\geq 2}(w)$, this implies that $[S_w[m-1], E_w[m]]$ is an L-root of r. Thus, $[S_w[m-2], E_w[m-1]]$ must also be an L-root with respect to the lexicographically reversed order, and $S_w[m-2] \notin B_r$. Since r ends at position $|w|$, we can choose either $S_w[m-1]$ or $S_w[m-2]$ as an element of B_r. This implies that either $S_w[m-1]$ or $S_w[m-2]$ is not in B_r for any $r \in \mathit{MReps}_{p\geq 2}(w)$. Thus $|\mathit{MReps}_{p\geq 2}(w)| \leq m-3$ also holds. Since 1 and $S_w[m]$ are not contained in B_r, and since only one of $S_w[m-1]$ or $S_w[m-2]$ is contained in some B_r, we have that the maximum number of maximal repetitions in a string is at most $m+k-3$. ◀

4 Computing All Maximal Repetitions on RLE strings

In this section, we propose an algorithm to compute all maximal repetitions on RLE strings. Our algorithm follows the new algorithm for normal strings proposed in [3], but is modified to handle RLE strings. We first review the algorithm for non-RLE strings.

4.1 Overview of Algorithm for Non-RLE Strings

The crucial observation made in [3] (which was also required for the proof of Lemma 3 in the previous section) is the following:

▶ **Lemma 8** (Lemma 7 of [3]). *For any maximal repetition $r = (i, j, p)$ of string w, let $[\ell, \ell + p - 1]$ be an L-root of r with respect to order $\prec_r$. Then, $w[\ell..\ell + p - 1]$ is the longest Lyndon word that is a prefix of $w[\ell..|w|]$.*

Based on this observation, the algorithm consists of two steps. Step 1: Compute all the longest Lyndon words with respect to $\prec_0$ and $\prec_1$ that start at each position of the string (the occurrences are candidates for L-roots). Step 2: For each such candidate $\lambda = w[i_\lambda..j_\lambda]$, compute $\ell_h = \mathit{LCE}_w(i_\lambda, j_\lambda + 1)$ and $\ell_g = \mathit{LCE}^R_w(i_\lambda - 1, j_\lambda)$ to see how long the period $p_\lambda = |w[i_\lambda..j_\lambda]| = j_\lambda - i_\lambda + 1$ continues to the left and to the right. We see that $[i_\lambda, j_\lambda]$ is indeed an L-root of the maximal repetition $r = (i_\lambda - \ell_g, j_\lambda + \ell_h, p_\lambda)$ if and only if $\ell_g + \ell_h \geq p_\lambda$.

Noticing that a Lyndon word can be created from any string by appending a unique smallest letter to the front of the string, we can use the *Lyndon tree* of a Lyndon word for Step 1. Given a Lyndon word w of length $n > 1$, (u, v) is the *standard factorization* [5, 21] of w, if $w = uv$ and v is the longest proper suffix of w that is a Lyndon word, or equivalently, the lexicographically smallest proper suffix of w. It is well known that for the standard factorization (u, v) of any Lyndon word w, the factors u and v are also Lyndon words (e.g.[4]). The *Lyndon tree* of w is the full binary tree defined by recursive standard factorization of w; w is the root of the Lyndon tree of w, its left child is the root of the Lyndon tree of u, and its right child is the root of the Lyndon tree of v. The longest Lyndon word that starts at each position can be obtained from the Lyndon tree, due to the following lemma.

▶ **Lemma 9** (Lemma 22 of [3]). *Let w be a Lyndon word with respect to $\prec$. $w[i..j]$ corresponds to a right node (or possibly the root) of the Lyndon tree with respect to $\prec$ if and only if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that starts from i.*

The Lyndon tree of a normal string can be computed in $O(n\alpha(n))$ time over general ordered alphabet because of the following lemmas.

▶ **Lemma 10** (Observation 4 of [9]). *The Lyndon tree of a string of length n can be constructed by using $O(n)$ non-crossing LCE queries.*

▶ **Lemma 11** (Theorem 12 of [9]). *In a string of length n, a sequence of q non-crossing LCE queries can be answered in time $O(q + n\alpha(n))$, where α denotes the inverse Ackermann function.*

Here, a set of LCE queries is *non-crossing* if there are no two queries (i, j) and (i', j'), such that $i < i' < j < j'$ or $i' < i < j' < j$. After computing the Lyndon tree, $O(n)$ non-crossing LCE queries are computed again for each right node in Step 2 as described above. Thus the total time complexity for computing all maximal repetitions in non-RLE string is $O(n\alpha(n))$ time over general ordered alphabet.

We note that the LCE queries and thus all maximal repetitions can be computed in total $O(n)$ time for integer alphabets (using e.g. [11]).

4.2 Extending Lyndon structures for RLE

We now consider computing maximal repetitions on RLE strings. By Theorem 2, the number of maximal repetitions in an RLE string is $O(m)$, and from Lemmas 4 and 8, we can limit the candidate L-roots of maximal repetitions with period at least 2, to the longest Lyndon words that start at beginning positions of a run-length factor. We propose the *RLE-Lyndon tree* of a string which can be represented in $O(m)$ space and contains this information. In the RLE-Lyndon tree, we treat each run-length factor like a character. The idea of the extension comes from the following lemma.

▶ **Lemma 12.** *For any $1 \le i < j \le |w|$, if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$, then $j \in E_w$, i.e., j is an end of a RLE-factor.*

Proof. Suppose to the contrary, that there is some $j \notin E_w$ such that $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$. Let $RLE(w)[k]$ be the run-length factor such that $S_w[k] \le j < E_w[k]$. Since $w[i..j]$ is a Lyndon word of length at least 2 and $w[j] = a_k = w[j+1]$, $w[i..j] \prec w[j] = w[j+1]$ holds. By Lemma 1, $w[i..j+1]$ is also a Lyndon word. This contradicts that $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that is a prefix of $w[i..|w|]$. ◀

From Lemmas 4 and 12, we have that for any maximal repetition r, each L-root of r that has a starting position in B_r, starts at the beginning position of some run-length factor and ends at the ending position of some run-length factor. We note that RLE-Lyndon substring and RLE-Lyndon factorization which will be defined in this section were introduced in [13] in a different context.

▶ **Definition 13** (RLE-Lyndon substring). A string x is an *RLE-Lyndon substring* of w if x is a Lyndon word that is a concatenation of consecutive run-length factors of w, or x is a run-length factor.

▶ **Definition 14** (RLE-standard factorization). A pair of strings (u, v) is an *RLE-standard factorization* of w if $w = uv$ and v is the longest proper suffix of w that is an RLE-Lyndon substring.

▶ **Definition 15** (RLE-Lyndon tree). The *RLE-Lyndon tree* of a Lyndon word w, denoted $LyndonTre^2(w)$, is an ordered full binary tree defined recursively as follows:

- if $|RLE(w)| = 1$, then $LyndonTre^2(w)$ consists of a single node labeled by (a_1, e_1);
- if $|RLE(w)| \ge 2$, then the root of $LyndonTre^2(w)$, labeled by $RLE(w)$, has left child $LyndonTre^2(u)$ and right child $LyndonTre^2(v)$, where (u, v) is the RLE-standard factorization of w.

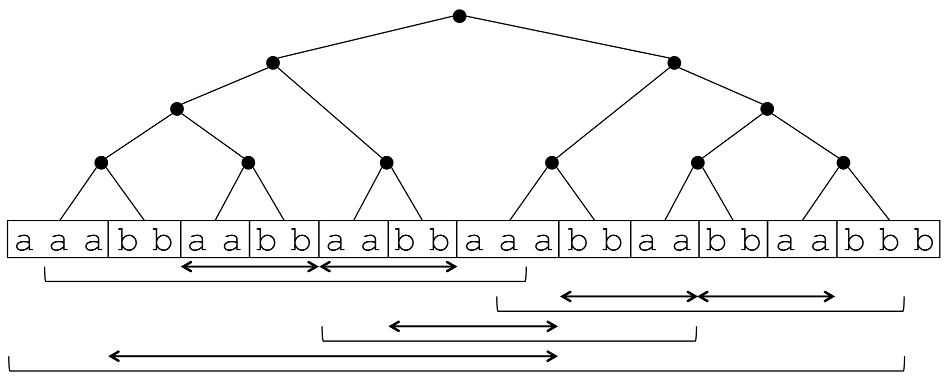

Figure 1 The RLE-Lyndon tree for the Lyndon word $\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^3$ with respect to order $\mathtt{a} \prec \mathtt{b}$. The double-headed arrows shows the L-roots that start at a position in B_r,for all 4 maximal repetitions with period at least 2.

Figure 1 shows the RLE-Lyndon tree of a string $\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^3\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^3$. Though the above structures are simply extended to RLE, it is interesting to note that these structures satisfy similar properties of the original structures. The most important property of the RLE-Lyndon tree in this paper is stated in Lemma 16, which is an analogous to Lemma 9. The lemma can be shown by similar arguments as in [3].

▶ **Lemma 16.** *Let w be a Lyndon word with respect to $\prec$. For any $i \in S_w$, $w[i..j]$ corresponds to a right node (or possibly the root) of $LyndonTre^2(w)$ with respect to $\prec$ if and only if $w[i..j]$ is the longest Lyndon word with respect to $\prec$ that starts from i.*

From the above lemma, we can detect all maximal repetitions in $MReps_{p\geq 2}(w)$ if we have $LyndonTre^2(w)$ (maximal repetitions with period 1 correspond to run-length factor or leaves of $LyndonTre^2(w)$). In the example of Figure 1, for each maximal repetition r, the L-roots that start at a position in B_r are drawn by double-headed arrows. For example, the 2 L-roots $[S_w[3]..E_w[4]]$ and $[S_w[5]..E_w[6]]$ (corresponding to a Lyndon word $\mathtt{aabb}$) with respect to the same order $\prec$ as the Lyndon tree is represented by an internal node which is a right child. Also, it can be observed that each L-root begins at the starting position of a run-length factor and ends at the ending position of a run-length factor of w.

In Section 4.3, we show an algorithm to compute $LyndonTre^2(w)$. For convenience, we present the notion of *RLE-Lyndon factorizations* and show some properties of RLE-Lyndon factorizations.

▶ **Definition 17** (RLE-Lyndon factorization). A sequence $w_1, \ldots, w_s$ is the RLE-Lyndon factorization of w if each w_i is an RLE-Lyndon substring, $w_1 \succeq \ldots \succeq w_s$, and $w = w_1 \cdots w_s$.

The difference between the original Lyndon factorization [5] and the RLE-Lyndon factorization arises for Lyndon factors which are a single letter in the original Lyndon factorization. For a string $w = \mathtt{bbbabbaabbaa}$, the original Lyndon factorization of w is $\mathtt{b} \succeq \mathtt{b} \succeq \mathtt{b} \succeq \mathtt{abb} \succeq \mathtt{aabb} \succeq \mathtt{a} \succeq \mathtt{a}$, the RLE-Lyndon factorization of w is $\mathtt{b}^3 \succeq \mathtt{abb} \succeq \mathtt{aabb} \succeq \mathtt{a}^2$. Thus similar argument about the longest Lyndon word on Lyndon factorizations holds, as below.

▶ **Lemma 18.** *Let $w_1, \ldots, w_s$ be the RLE-Lyndon factorization of w. Then, w_1 is either $RLE(w)[1]$ or the longest Lyndon word that is a prefix of w.*

This implies that w_i is either $RLE(w_i \cdots w_s)[1]$ or the longest Lyndon word that is a prefix of $w_i \cdots w_s$.

4.3 Algorithms

Finally, we show how to compute $\mathit{LyndonTre}^2(w)$ in $O(m\alpha(m))$ time and $O(m)$ space. After we compute $\mathit{LyndonTre}^2(w)$, we can compute all maximal repetitions by using non-crossing LCE queries. Note that the $O(n)$ time and space solution for non-RLE strings over the integer alphabet cannot be applied to $\mathit{RLE}(w)$ to achieve an $O(m)$ time and space solution, since the alphabet for $\mathit{RLE}(w)$ cannot be assumed to be an integer alphabet in terms of its length m (m could be much smaller than n, while an exponent of a run-length factor could be as large as n). However, the solution to non-crossing LCE queries for non-RLE strings over a general ordered alphabet can be easily extended to LCE queries on an RLE string, since the algorithm is based only on character comparisons.

▶ **Corollary 19.** *For any RLE string $\mathit{RLE}(w)$ of size m, a sequence of q non-crossing LCE queries on $\mathit{RLE}(w)$ can be answered in time $O(q + m\alpha(m))$.*

We use the above corollary in order to decide the lexicographic order between RLE substrings in the construction of $\mathit{LyndonTre}^2(w)$, and to compute maximal repetitions.

▶ **Lemma 20.** *$\mathit{LyndonTre}^2(w)$ can be computed in $O(m\alpha(m))$ time and $O(m)$ space.*

Proof. Firstly, we show our algorithm. The algorithm constructs $\mathit{LyndonTre}^2(w)$ in bottom-up and from right to left. The main idea is that the right factor of RLE-standard factorization is the longest proper suffix which is an RLE-Lyndon substring. We will find such a suffix by concatenating two RLE-Lyndon substrings based on Lemma 1. Since each leaf corresponds to a single run-length factor (i.e., RLE-Lyndon substring), we know that the tree has m leaves. A stack is maintained so that at the beginning of k-th step, the stack contains the sequence of subtrees of $\mathit{LyndonTre}^2(w)$ such that the corresponding sequence of RLE-Lyndon substrings is the RLE-Lyndon factorization of the suffix $w[S_w[m-k+2]..|w|]$. In the k-th step, the algorithm pushes the leaf corresponding to $\mathit{RLE}(w)[m-k+1]$ on the stack. Let (f_b, f_e) (resp. (s_b, s_e)) be pair of positions in $\mathit{RLE}(w)$ such that the top (resp. second) subtree in the stack corresponds to the RLE-Lyndon substring $w[S_w[f_b]..E_w[f_e]]$ (resp. $w[S_w[s_b]..E_w[s_e]]$). Note that $E_w[f_e] + 1 = S_w[s_b]$ always holds. After pushing the new leaf, the algorithm does the following;

- If $w[S_w[f_b]..E_w[f_e]] \prec w[S_w[s_b]..E_w[s_e]]$, pop the two elements and push the subtree which is the concatenation of the two popped subtrees, and repeat the process.
- Otherwise, go to the next step.

We now prove that the above invariant condition of the stack holds before $k+1$-th step. We denote the RLE-Lyndon factorization of the suffix $w[S_w[m-k+2]..|w|]$ by $W_1, \ldots, W_j$. Because of the above operations, a factorization of the suffix $w[S_w[m-k+1]..|w|]$ can be represented by $W', W_i, \ldots, W_j$ for some $1 \le i \le j$ where $W' = \mathit{RLE}(w)[m-k+1]W_1 \cdots W_{i-1}$ (for convenience, $W_0 = \epsilon$). By the assumption, $W_i, \ldots, W_j$ is the RLE-Lyndon factorization of the suffix $W_i \cdots W_j$. By the algorithm and Lemma 1, W' is an RLE-Lyndon substring and $W' \succeq W_i$ holds. Thus $W', W_i, \ldots, W_j$ is the RLE-Lyndon factorization of the suffix $w[S_w[m-k+1]..|w|]$ since $W' \succeq W_i \succeq \ldots \succeq W_j$ holds. Since w is a Lyndon word, when all leaves are pushed on the stack and the number of elements in the stack is one, the algorithm stops and the RLE-Lyndon tree is completely constructed.

We can determine the lexicographic order by using LCE queries. More precisely, for each lexicographic comparison described above, we compute $\mathit{LCE}_{\mathit{RLE}(w)}(f_b, s_b) = k$. Then, $w[S_w[f_b]..E_w[f_e]] \prec w[S_w[s_b]..E_w[s_e]]$ if and only if $s_b + k - 1 < s_e$ and, either

1. $a_{f_b+k} \prec a_{s_b+k}$ or, $a_{f_b+k} = a_{s_b+k}$ and
2. $e_{f_b+k} < e_{s_b+k}$ and $a_{f_b+k+1} \prec a_{s_b+k}$ or
3. $e_{f_b+k} > e_{s_b+k}$ and $a_{s_b+k+1} \prec a_{f_b+k}$.

Thus, in the algorithm, we call $O(m)$ non-crossing LCE queries such that each query positions is the beginning position of some run-length factor and we can compute $\mathit{LyndonTre}^2(w)$ in $O(m\alpha(m))$ time.

To compute all maximal repetitions, we need to compute another $O(m)$ sets of LCE queries on w (or w^R) for each candidate L-root. The query positions are starting positions of run-length factors in w (or w^R). It is easy to see that this can also be achieved in $O(m\alpha(m))$ time by Corollary 19 since if $k = \mathit{LCE}_{\mathit{RLE}(w)}(i,j)$, then $\mathit{LCE}_w(S_w[i], S_w[j]) = E_w[i+k-1] - S_w[i] + 1 + e$, where $e = \min\{e_{i+k}, e_{j+k}\}$ if $a_{i+k} = a_{j+k}$ and 0 otherwise. It is also clear that the algorithm requires $O(m)$ space. ◀

Therefore, the following theorem holds.

▶ **Theorem 21.** *Given a run-length encoding of a string w, all maximal repetitions in w can be computed in $O(m\alpha(m))$ time and $O(m)$ space.*

▶ **Corollary 22.** *For any string w, let $\mathit{RLE}(w) = (a_1, e_1) \cdots (a_m, e_m)$. If for all $1 \le i \le m$, $a_i \in \{1, \ldots, m^{c_1}\}$, and $e_i = O(m^{c_2})$ for some constants c_1 and c_2, then all maximal repetitions in w can be computed in $O(m)$ time and $O(m)$ space.*

Proof. Under the assumption, any set of $O(m)$ LCE queries on $\mathit{RLE}(w)$ can be answered in $O(m)$ total time using the methods for integer alphabets (i.e., $\Sigma = \{1, \ldots, m^c\}$ for some constant c), since $a_i^{e_i} = O(m^{c_1+c_2})$. ◀

References

1 Anthony Bagnall, Chotirat "Ann" Ratanamahatana, Eamonn Keogh, Stefano Lonardi, and Gareth Janacek. A bit level representation for time series data mining with shape based similarity. *Data Mining and Knowledge Discovery*, 13(1):11–40, 2006.

2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. A new characterization of maximal repetitions by lyndon trees. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 562–571. SIAM, 2015. doi:10.1137/1.9781611973730.38.

3 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

4 Frédérique Bassino, Julien Clément, and Cyril Nicaud. The standard factorization of Lyndon words: an average point of view. *Discrete Mathematics*, 290(1):1–25, 2005.

5 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. iv. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.

6 Kuan-Yu Chen and Kun-Mao Chao. A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, 65(2):354–370, 2013. doi:10.1007/s00453-011-9592-4.

7 Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao. Efficient retrieval of approximate palindromes in a run-length encoded string. *Theor. Comput. Sci.*, 432:28–37, 2012. doi:10.1016/j.tcs.2012.01.023.

8 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/j.ipl.2007.10.006.

9 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Near-optimal computation of runs over general alphabet via non-crossing LCE queries. In *Proc. SPIRE 2016*, pages 22–34, 2016.

10 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4(4):363–381, 1983.

11 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. `doi:10.1007/11780441_5`.

12 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. In *Proc. CPM 2016*, pages 5:1–5:13, 2016.

13 Sukhpal Singh Ghuman, Emanuele Giaquinta, and Jorma Tarhio. Alternative algorithms for lyndon factorization. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2014, Prague, Czech Republic, September 1-3, 2014*, pages 169–178. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2014. URL: `http://www.stringology.org/event/2014/p16.html`.

14 Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: Efficient and flexible detection of tandem repeats in DNA. *Nucleic acids research*, 31(13):3672–3678, July 2003.

15 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. `doi:10.1109/SFFCS.1999.814634`.

16 Dmitry Kosolobov. Lempel-ziv factorization may be harder than computing all runs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 582–593. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. `doi:10.4230/LIPIcs.STACS.2015.582`.

17 Dmitry Kosolobov. Computing runs on a general alphabet. *Inf. Process. Lett.*, 116(3):241–244, 2016.

18 Keita Kuboi, Yuta Fujishige, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster STR-IC-LCS computation via RLE. In *Combinatorial Pattern Matching, 28th Annual Symposium, CPM 2017, Warsaw, Poland, July 4-6, 2017, Proceedings*, 2017. in press. URL: `http://arxiv.org/abs/1703.04954`.

19 Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

20 Jia Jie Liu, Yue-Li Wang, and Yu-shan Chiu. Constrained longest common subsequences with run-length-encoded strings. *Comput. J.*, 58(5):1074–1084, 2015. `doi:10.1093/comjnl/bxu012`.

21 M. Lothaire. *Combinatorics on Words*. Addison-Wesley, 1983.

22 Roger C. Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77(2):202–202, February 1954.

23 Michael G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989. `doi:10.1016/0166-218X(89)90051-6`.

24 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

25 Axel Thue. Über unendliche zeichenreihen. *Christiana Videnskabs Selskabs Skrifter, I. Math. naturv. Klasse, 7*, 1906.

26 Jun-ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Faster compact on-line lempel-ziv factorization. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 675–686. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. `doi:10.4230/LIPIcs.STACS.2014.675`.

27 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–349, 1977.