Left Lyndon Tree Construction

Golnaz Badkobeh[1] and Maxime Crochemore[2,3]

[1] Department of Computing
Goldsmiths University of London
United Kingdom
g.badkobeh@gold.ac.uk

[2] Department of Informatics
King's College London
United Kingdom
Maxime.Crochemore@kcl.ac.uk

[3] Université Gustave Eiffel
Marne-la-Vallée, France

Abstract. We extend the left-to-right Lyndon factorisation of a word to the left Lyndon tree construction of a Lyndon word. It yields an algorithm to sort the prefixes of a Lyndon word according to the infinite ordering defined by Dolce et al. (2019). A straightforward variant computes the left Lyndon forest of a word. All algorithms run in linear time on a general alphabet (letter-comparison model).

1 Lyndon words

In this article we consider algorithmic questions related to Lyndon words. Introduced in the field of combinatorics by Lyndon (see [11]) and used in algebra, these words have shown their usefulness for designing efficient algorithms on words. The notion of Lyndon tree associated with the decomposition of a Lyndon word, for example, has been used by Bannai et al. [1] to solve a conjecture of Kolpakov and Kucherov [9] on the maximal number of runs (maximal periodicities) in words, following a result in [2].

The key result in [1] is that every run in a word y contains as a factor, a Lyndon root (according to the alphabet order or its inverse), that corresponds to a node of the associated Lyndon tree. Since the Lyndon tree has a linear number of nodes according to the length of y, browsing all its nodes leads to a linear-time algorithm in order to report all the runs occurring in y. However, the time complexity of this technique also depends on the time it takes to build the tree and to extend a potential root to an actual run.

Here we consider the left Lyndon tree of a Lyndon word y. This tree has a single node if y is reduced to a single letter, otherwise its structure corresponds recursively to the left standard factorisation (see Viennot [13]) of y as uv where u is the longest proper Lyndon prefix of y.

The dual notion of the right Lyndon tree of a Lyndon word y (based on the factorisation $y = uv$ where v is the longest proper Lyndon suffix of y) is strongly related to the sorted list of suffixes of y. Indeed, Hohlweg and Reutenauer [8] showed that the Lyndon tree is the Cartesian tree built from the ranks of suffixes in their sorted list (sometimes called the inverse suffix array of the word). The list corresponds to the standard permutation of suffixes of the word and is the main component of the suffix array (see [4]), one of the major data structures for text indexing.

Golnaz Badkobeh, Maxime Crochemore: Left Lyndon Tree Construction, pp. 84–95.
Proceedings of PSC 2020, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06749-9 © Czech Technical University in Prague, Czech Republic

Inspired by a result of Ufnarovskij [12], Dolce et al. [6] showed that the left Lyndon tree is also a Cartesian tree built from ranks of prefixes sorted according to an order they call the infinite order.

The main result of this article is to show that sorting prefixes of a Lyndon word according to the infinite order can be attained in linear time in the letter-comparison model (comparing letters is assumed to be carried out in constant time). This produces the prefix standard permutation of the word. The algorithm is based on the Lyndon factorisation of words by Duval [7] and it extends naturally to build the Lyndon forest of a word.

Definitions

Let A be an alphabet with an ordering $<$ and A^+ be the set of all non-empty words over A. The length of a word w is denoted by $|w|$. Let ϵ denotes the empty word, i.e., word of length 0. We say that uv is a non-trivial factorisation of a word w if $uv = w$ and u, v are non-empty words.

A word is said to be strongly smaller than a word v, $u \ll v$, if there are words r, s and t, and letters a and b with $u = ras$, $v = rbt$ and $a < b$. A word u is smaller than a word v, $u < v$, if either $u \ll v$ or u is a proper prefix of v. In addition to this usual lexicographical ordering the infinite order $\prec$ (see [5,6]) is defined by: $u \prec v$ if $u^\infty < v^\infty$ or both $u^\infty = v^\infty$ and $|u| > |v|$. Note that $u^\infty = v^\infty$ implies that u and v are powers of the same word, consequence of Fine and Wilf's Periodicity lemma (see [10, Proposition 1.3.5]). For example, if $u = \mathtt{abba}$, $v = \mathtt{abb}$, then $u^\infty = \mathtt{abbaabbaabba}\,..$ and $v^\infty = \mathtt{abbabbabb}\,..$, therefore $u^\infty < v^\infty$ and consequently $u \prec v$. If $u = \mathtt{ababab}$, $v = \mathtt{abab}$, then $u^\infty = v^\infty = \mathtt{abababab}\,..$ and $u \prec v$.

The next proposition defines Lyndon words that are not reduced to a single letter. Condition in item (i) is the original definition and condition in item (iii) is by Ufnarovskij [12].

Proposition 1. *The following conditions are equivalent and define a Lyndon word w, $|w| > 1$: for any non-trivial factorisation uv of w, (i) $w < vu$, (ii) $w < v$, (iii) $u^\infty < w^\infty$.*

2 Lyndon suffix table

This section recalls known algorithms. The algorithms presented in this article strongly use the notion of Lyndon suffix table of a word, which is denoted by $LynS_y$ or simply by $LynS$ for the generic word y. Table $LynS$ of a word y is defined, for each position j on y, by

$$LynS[j] = \max\{|w| \mid w \text{ Lyndon suffix of } y[0\,..\,j]\}.$$

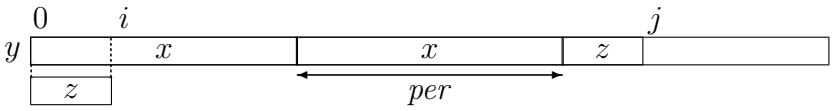
Example 2. Let $y_0 = \mathtt{ababbababbabac}$ on the alphabet of constant letters $\{\mathtt{a}, \mathtt{b}, \ldots\}$ ordered as usual $\mathtt{a} < \mathtt{b} < \cdots$. The corresponding $LynS_{y_0}$ table is as follows:

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$y[j]$	a	b	a	b	b	a	b	a	b	b	a	b	a	c
$LynS_{y_0}[j]$	1	2	1	2	5	1	2	1	2	5	1	2	1	14

The $LynS$ table is the dual notion of the Lyndon table l in [1] or Lyn in [3] used to detect maximal periodicities (also called runs) in words: $Lyn[j]$ is the maximal length of the Lyndon prefixes of $y[j\,..\,|y|-1]$.

The computation of *LynS* is a mere extension of the algorithm for testing if a word is the prefix of a Lyndon word. It includes the key point of the factorisation algorithm in [7] and is recalled first as Algorithm LyndonWordPrefix that works online on its input word y.

```
LyndonWordPrefix(y non-empty word of length n)
 1  (per, i) ← (1, 0)
 2  for j ← 1 to n − 1 do
 3       if y[j] < y[i] then          ▷ y[i] = y[j − per]
 4            return FALSE
 5       elseif y[j] > y[i] then
 6            (per, i) ← (j + 1, 0)
 7       else i ← i + 1 mod per
 8  return TRUE
```



The key feature of the method stands in lines 5-6 of the algorithm and is illustrated on the above picture. If $y[j] > y[i] = y[j - per]$, not only the periodicity *per* of $y[0\,..\,j-1]$ breaks but $y[0\,..\,j]$ is a Lyndon word with period $j+1$. This feature is a consequence of the following known properties.

Proposition 3. *(i) Let z be a word and a a letter for which za is a prefix of the Lyndon word x. Let b be a letter with $a < b$. Then zb is a Lyndon word.*
(ii) Let u and v be two Lyndon words with $u < v$, then uv is Lyndon word.

Algorithm LyndonSuffix computes the Lyndon suffix table of a Lyndon word. This algorithm results from a minor modification of Algorithm LyndonWordPrefix and can be easily enhanced to compute also the smallest period of all non-empty prefixes of the input.

```
LyndonSuffix(y Lyndon word of length n)
 1  LynS[0] ← 1
 2  (per, i) ← (1, 0)
 3  for j ← 1 to n − 1 do
 4       if y[j] ≠ y[i] then          ▷ y[j] > y[i] = y[j − per]
 5            LynS[j] ← j + 1
 6            (per, i) ← (j + 1, 0)
 7       else LynS[j] ← LynS[i]
 8            i ← i + 1 mod per
 9  return LynS
```

Proposition 4. *Algorithm* LyndonSuffix *computes the Lyndon suffix table of a Lyndon word of length n in time $O(n)$ in the letter-comparison model.*

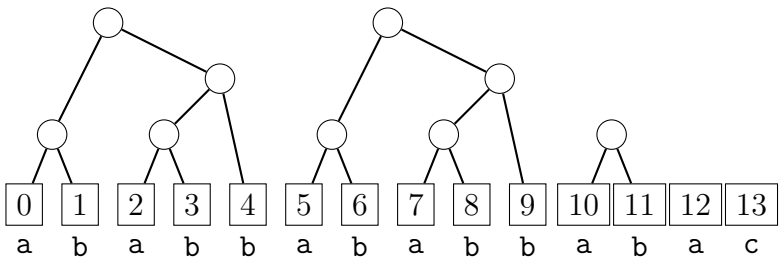
3 Left Lyndon tree construction

The left Lyndon tree $\mathcal{L}(y)$ of a Lyndon word y represents recursively the left standard factorisation of Lyndon words. Leaves of the tree are positions on the word and internal nodes correspond to concatenations of Lyndon factors of the word, and as such can be viewed as interpositions. Namely, $\mathcal{L}(y) = (0)$ if $|y| = 1$ else it is $(p, \mathcal{L}(u), \mathcal{L}(v))$ where the root $p \in \{|y| \,..\, 2|y| - 2\}$ is an integer and uv is the left standard factorisation of y, that is, u is the longest proper Lyndon prefix of y (v is then a Lyndon word).

Subtrees of $\mathcal{L}(y)$ are handled from positions on y in the following manner. The subtree associated with position j is $\mathcal{L}(y[i \,..\, j])$ with root root$[j]$, where $y[i \,..\, j]$ is the longest Lyndon suffix of $y[0 \,..\, j]$, i.e. $j - i + 1 = LynS[j]$. Position j on y is the rightmost leaf of the subtree and $LynS[j]$ is its width.

It is known that y, $|y| > 1$, is of the form $x^k zb$ where x is a Lyndon word of length $per = period(x^k z)$, $k > 0$, z is a proper prefix of x and b is a letter greater than letter a following z in x (za is a prefix of x) [7].

The construction of $\mathcal{L}(y)$ is achieved with the help of the table $LynS$ of y. It is done by processing y from left to right building first $\mathcal{L}(x)$ and reproducing that tree or part of it up to z. The picture displays the subtrees built for the word $(\texttt{ababb})^2\texttt{aba}$.

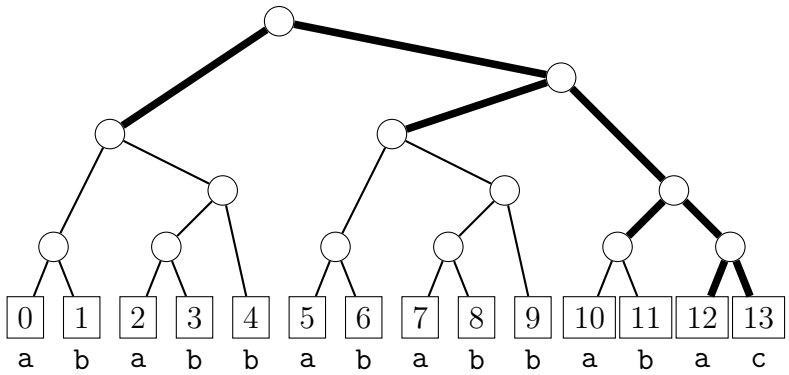


The main step of the procedure, in addition to computing $LynS$ by Algorithm LyndonSuffix, is to aggregate partial Lyndon trees when processing the last letter b of y; to create the final tree as a bundle of all subtrees. In fact, this step is also carried out when dealing with $x^k z$ at each position j for which $LynS[j] > 1$. In order to aggregate the subtrees, the second property of Proposition 3 is applied iteratively, processing the subtrees from right to left. Instruction of this step appear at lines 10-15 in Algorithm LeftLyndonTree, in which left$[q]$ and right$[q]$ are respectively the left and right children of the internal node q of the tree.

The process of bundling can be viewed as a translation into the tree structure of the proof of the key feature of Algorithm LyndonWordPrefix. Even so the latter deals with this process in constant time, which is not the case here, the iteration of bundling instructions does not affect the asymptotic running time of the present algorithm.

```
LeftLyndonTree(y Lyndon word of length n)
 1  (LynS[0], root[0]) ← (1, 0)
 2  (per, i) ← (1, 0)
 3  for j ← 1 to n − 1 do
 4        root[j] ← j
 5        if y[j] ≠ y[i] then          ▷ y[j] > y[i] = y[j − per]
 6              LynS[j] ← j + 1
 7              (per, i) ← (j + 1, 0)
 8        else LynS[j] ← LynS[i]
 9              i ← i + 1 mod per
10        (ℓ, k) ← (1, j − 1)
11        while ℓ < LynS[j] do
12              q ← new node ≥ n
13              (left[q], right[q]) ← (root[k], root[j])
14              root[j] ← q
15              (ℓ, k) ← (ℓ + LynS[k], k − LynS[k])
16  return root[n − 1]
```

The picture below shows thick links and nodes created by the final round of instructions at lines 10-15 in Algorithm LeftLyndonTree.



Proposition 5. *Algorithm* LeftLyndonTree *builds the left Lyndon tree of a Lyndon word of length n in time $O(n)$ in the letter-comparison model.*

Proof. All instructions inside the **for** loop are executed in constant time except the **while** loop. In addition, since each execution of instructions in the **while** loop takes constant time and leads to the creation of an internal node of the final tree, twinned with the fact that there are exactly $n - 1$ internal nodes, the total running time is $O(n)$.

4 Sorting prefixes

We show that Algorithm LeftLyndonTree can be adapted to sort the prefixes of a Lyndon word according to the infinite ordering $\prec$. This is a consequence of Theorem 7 below.

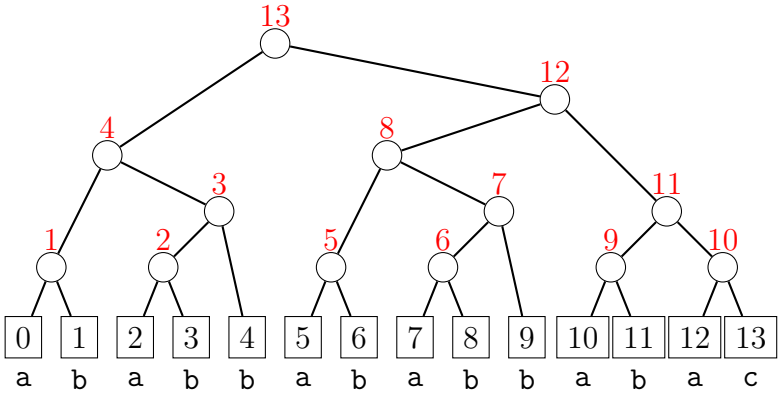
An internal node p of the left Lyndon tree of a Lyndon word y is the root of a Lyndon subtree associated with a Lyndon factor w, $|w| > 1$ of y. This factor is

obtained by concatenating two consecutive occurrences of Lyndon factors u and v. If the concerned occurrence of w ends at position j on y, node p is identified with the prefix of y ending at position j. The correspondence between internal nodes of the tree and proper non-empty prefixes of y is clearly one-to-one because internal nodes are identified with interpositions, pairs $(i, i+1)$ of positions on y.

Labelling internal nodes with the $\prec$-ranks of their associated prefixes transforms the tree into a heap, i.e. ranks are increasing from leaves to the root. The relation between the infinite order and left Lyndon trees is established by the next result [6].

Theorem 6 (Dolce, Restivo, Reutenauer, 2019). *The tree of internal nodes of the left Lyndon tree of a Lyndon word y in which nodes are labelled by the ranks of proper prefixes of y sorted according to the infinite order is the Cartesian tree of the ranks.*

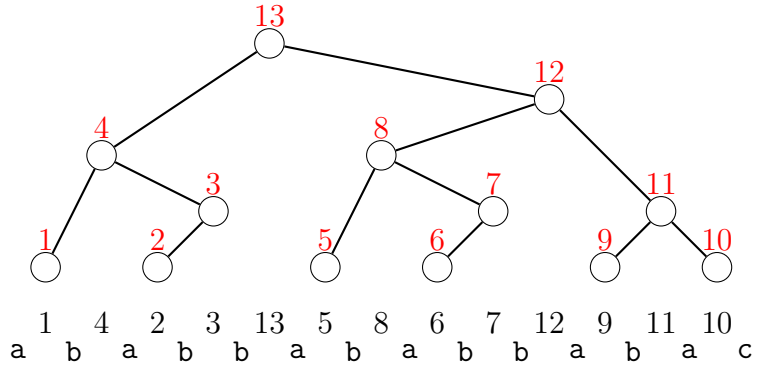
The following picture shows the left Lyndon tree of $y_0 =$ `ababbababbabac` and the $\prec$-rank labels of its internal nodes.



Denoting a prefix of y_0 by the position of its last letter (length minus 1), the table below shows $\prec$-ranks of proper non-empty prefixes of the word and their sorted list, inverse of Rank. The sorted list is `a` $\prec$ `aba` $\prec$ `abab` $\prec$ `ab` $\prec$ `ababba` $\prec$ `ababbaba` $\prec$ `ababbabab` $\prec$ `ababbab` $\prec$ `ababbababba` $\prec$ `ababbababbaba` $\prec$ `ababbababbab` $\prec$ `ababbababb` $\prec$ `ababb`.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$y[j]$	a	b	a	b	b	a	b	a	b	b	a	b	a	c
rank$[j]$	1	4	2	3	13	5	8	6	7	12	9	11	10	
prefix list	0	2	3	1	5	7	8	6	10	12	11	9	4	

The tree below is the Cartesian tree of prefix $\prec$-ranks.



Note that Algorithm LeftLyndonTree processes the resulting tree in left-to-right post-order. The next result links this order to prefix $\prec$-ranks.

Theorem 7. *Algorithm* LeftLyndonTree *on a Lyndon word y of length $n > 1$, creates and processes internal nodes of the tree in the order of their corresponding prefix ranks according to the ordering $\prec$.*

Proof. A Lyndon word that is not reduced to a single letter, y is of the form $x^k zb$ where x is a Lyndon word of length $period(x^k z)$, $k > 0$, z is a proper prefix of x and b is a letter greater than letter a following prefix z in x [7].

Algorithm LeftLyndonTree processes nodes of the Lyndon trees $\mathcal{L}(y)$ as follows. Initially, it builds $\mathcal{L}(x)$ and Lyndon trees of the next occurrences of x in a left-to-right manner. It continues with the tree related to z. Eventually during the last bundling (run of instructions at lines 10-15) the algorithm builds $\mathcal{L}(zb)$ and follows with the nodes corresponding to the concatenations $x \cdot zb$, $x \cdot xzb$, $\ldots$, $x \cdot x^{k-1}zb$ in that order.

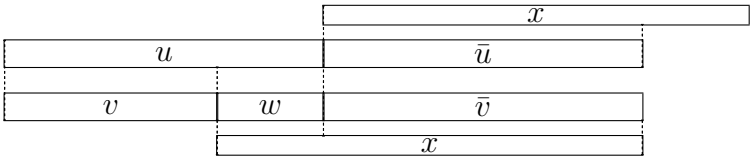
The statement is proved by induction on the length of the period $|x|$ of $x^k z$. If is $|x| = 1$, x is reduced to a single letter and y is of the form $a^k b$ for two letters a and b with $a < b$. Nodes associated with prefixes a^k, a^{k-1}, $\ldots$, a are processed in this order, which matches the $\prec$-order of prefixes, $a^k \prec a^{k-1} \prec \cdots \prec a$, as expected.

We then assume $|x| > 1$ and consider disjoint groups of non-empty proper prefixes of y. For $e = 0, 1, \ldots, k$, let

$$P_e = \{x^e u \text{ prefix of } y \mid e|x| < |x^e u| < \min\{(e+1)|x|, |y|\}\}.$$

The main part of the proof relies on three claims that we prove first.

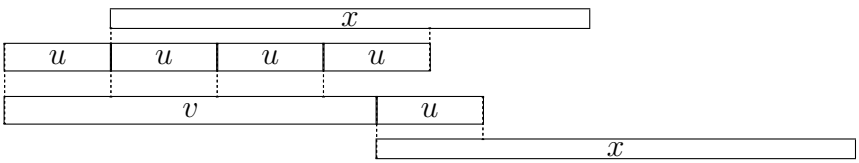
Claim 1: prefixes $x^e u \in P_e$, $0 < e \leq k$, are in the same relative $\prec$-order as prefixes $u \in P_0$. Let $u, v \in P_0$ with $u \prec v$ and let us show $x^e u \prec x^e v$ considering two cases.



Case $u^\infty = v^\infty$ and $|u| > |v|$. By the Periodicity lemma u, v and $v^{-1}u$ are powers of the same word. Let $w = v^{-1}u$, $\bar{v} = w^{-1}x$ and $\bar{u}$ the prefix of x of length $|\bar{v}|$ (see picture). Since x is a Lyndon word, $\bar{u} < x < \bar{v}$, which implies $ux < vx$ because w is a prefix of x. Therefore we have $(x^e u)^\infty < (x^e v)^\infty$, that is, $x^e u \prec x^e v$.

Case $u^\infty < v^\infty$. Assume u is shorter than v and let h be the largest exponent for which u^h is a prefix of v. It is a proper prefix because $u^\infty < v^\infty$ and then $w = (u^h)^{-1}v$ is not empty.

If $|u| \leq |w|$, we have $u \ll w$, which implies $ux \ll vx$ and $(x^e u)^\infty < (x^e v)^\infty$, that is, $x^e u \prec x^e v$. (This case can happen if for example, $u = \texttt{ab}$, $v = \texttt{abababbbb}$, then $w = \texttt{bbb}$ which means $|u| < |w|$)

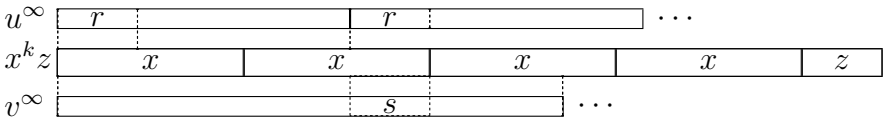


If $|u| > |w|$, v is a proper prefix of u^{h+1} but u^{h+1} shorter than vu cannot be a prefix of it due to the Periodicity lemma applied on periods $|u|$ and $|v|$ of u^{h+1}. Then

$u \ll wu$ and since u is a prefix of x it implies $ux \ll vx$ and $(x^e u)^\infty < (x^e v)^\infty$, that is, $x^e u \prec x^e v$ as before.

The situation in which u is longer than v is fairly symmetric and treated similarly. Therefore again $u \prec v$ implies $x^e u \prec x^e v$, which proves the claim.

Claim 2: prefixes in P_e are $\prec$-smaller than prefixes in P_f when $0 \le e < f \le k$. Let $u \in P_e$ and $v \in P_f$. We have to compare u and v according to $\prec$, that is, to compare u^∞ and v^∞.



When $e > 0$, u is longer than x. Let r be the prefix of u for which $|ur| = |x^{e+1}|$ (see picture in which $u \in P_1$ and $v \in P_2$) and s the suffix of x of the same length. Comparing u^∞ and v^∞ amounts to compare r and s, because u is a prefix of v. Since r is a prefix and s a suffix of the Lyndon word x, we have $r < s$ and even more, $r \ll s$, then $u^\infty < v^\infty$ and $u \prec v$.

When $e = 0$, u is shorter than x. Let h be the largest integer for which u^h is a prefix of x. It is a proper prefix because x is a Lyndon word and $w = (u^h)^{-1}x$ is not empty. As in the proof of previous claim, u^{h+1} cannot be prefix of xu that is a prefix of v. The same conclusion follows, that it, $u^{h+1} \ll vu$ and eventually $u \prec v$.

Claim 3: prefixes in P_e, $0 \le e \le k$, are $\prec$-smaller than prefixes x^f, $0 < f \le k$. To prove the claim, in view of the statement of Claim 2 and the fact $x^k \prec x^{k-1} \prec x$ by definition, it is enough to show that $P_k \prec x^k$. Note that if P_k is empty the proof can be shown with P_{k-1} instead, and if in addition $k = 1$ then we are left with an element in the proof of Claim 2.

Let $x^k u \in P_k$, $s = u^{-1}x$ and r the prefix of x of length $|s|$. As prefix and suffix of x, r and s satisfy $r < s$. Since $x^k ur < x^k us = x^{k+1}$ and r is a prefix of x, it results in $(x^k u)^\infty < x^\infty$ and eventually $x^k u \prec x^k$. This proves the claim.

To summarise, claims show

$$P_0 \prec P_1 \prec \cdots \prec P_k \prec x^k \prec x^{k-1} \prec \cdots \prec x.$$

Let us go back to induction. By induction hypothesis, the result holds for internal nodes of $\mathcal{L}(x)$ corresponding to prefixes in P_0.

Consider the next occurrences of x. Since the Lyndon suffix table for each of them is copied from that of prefix x due to the instruction at line 8 in Algorithm LeftLyndonTree, the Lyndon trees of all occurrences of x have the same structure. Therefore, both from the induction hypothesis and from Claim 1, the order in which internal nodes of the eth occurrence of x are processed and created matches the $\prec$-order of prefixes in P_e, for $0 < e \le k$.

The algorithm processes occurrences of x from left to right, which corresponds to the result of Claim 2. The treatment of zb is done at the beginning of the bundling run, which also corresponds to the fact that prefixes in P_k are $\prec$-larger than all prefixes that have been considered before.

Finally, the last part of the bundling creates nodes associated with x^k, x^{k-1}, ..., x in that order, which matches the order $x^k \prec x^{k-1} \prec \cdots \prec x$.

This ends the proof of the theorem.

A consequence of the theorem is that Algorithm LeftLyndonTree can be downgraded to compute directly the $\prec$-sorted list of non-empty proper prefixes of a Lyndon word. Dolce et al. [6] call this ordered list the prefix standard permutation. The following algorithm computes the prefix standard permutation.

```
PrefixStandardPermutation(y Lyndon word of length n)
 1  S ← ()
 2  (LynS[0], per, i) ← (1, 1, 0)
 3  for j ← 1 to n − 1 do
 4       if y[j] ≠ y[i] then          ▷ y[j] > y[i] = y[j − per]
 5            LynS[j] ← j + 1
 6            (per, i) ← (j + 1, 0)
 7       else LynS[j] ← LynS[i]
 8            i ← i + 1 mod per
 9       (k, m) ← (j − 1, 1)
10       while m < LynS[j] do
11            S ← S · (j − m)
12            m ← m + LynS[k]
13            k ← k − LynS[k]
14  return S
```

Corollary 8. *Sorting the proper non-empty prefixes of a Lyndon word of length n according to the infinite order $\prec$ can be carried out in time $O(n)$ in the letter-comparison model.*

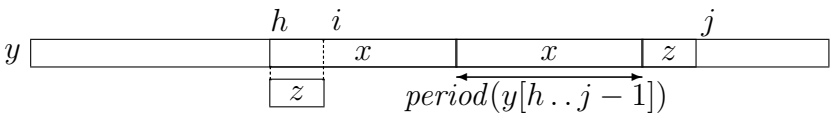
Proof. It essentially suffices to substitute the handling of the sequence to the processing of internal nodes of the Lyndon tree of the word in Algorithm LeftLyndonTree, which is equivalent to do a left-to-right post-order traversal of the tree. The change is realised by Algorithm PrefixStandardPermutation.

5 Lyndon forest

When the non-empty word y is not a Lyndon word, the above process can be carried out on each factor of its Lyndon factorisation, a decreasing list of Lyndon factors of the word. Lyndon factorisation of y is a list $x_1, x_2, \ldots, x_k$ for which both $x_1 x_2 \cdots x_k = y$ and $x_1 \geq x_2 \geq \cdots \geq x_k$. This factorisation is unique (see [10, Chen-Fox-Lyndon theorem]).

The factorisation and its algorithm by Duval [7] is the guiding thread of previous algorithms. The Lyndon forest of word y is the list of Lyndon trees $\mathcal{L}(x_1)$, $\mathcal{L}(x_2)$, …, $\mathcal{L}(x_k)$. Its computation uses again the Lyndon suffix table of the word, computed by Algorithm LongestLyndonSuffix whose input is not necessarily a Lyndon word.

It is a revision of Algorithm LyndonSuffix. The change stands in instructions on lines 4-7. They reset the computation to the suffix $y[h\,..\,n-1]$ of the input after the factorisation of the prefix $y[0\,..\,h-1]$ is definitely achieved.



```
LongestLyndonSuffix(y non-empty word of length n)
 1  LynS[0] ← 1
 2  (per, h, i, j) ← (1, 0, 0, 1)
 3  while j < n do
 4      if y[j] < y[i] then
 5          h ← j − (i − h)
 6          LynS[h] ← 1
 7          (per, i, j) ← (1, h, h + 1)
 8      elseif y[j] > y[i] then
 9          LynS[j] ← j − h + 1
10          j ← j + 1
11          (per, i) ← (j − h, h)
12      else LynS[j] ← LynS[i]
13          (i, j) ← (h + (i − h + 1 mod per), j + 1)
14  return LynS
```

Proposition 9. *Algorithm* LongestLyndonSuffix *computes the Lyndon suffix table of a word of length $n > 0$ in time $O(n)$ in the letter-comparison model.*

Proof. Let us consider values of expression $h+j$ and show they form a strictly increasing sequence after each iteration in the **while** loop. This claim holds if the condition at line 4 is false, because j is incremented by at least one unit (on line 10 or on line 13) and h remains unchanged. This claim also holds if the condition at line 4 is true, because h is incremented by at least $period(y[h\,..\,j-1])$ while j is decremented by less than the same value.

Since $h + j$ goes from 1 to at most $2n - 1$ the running time is $O(n)$.

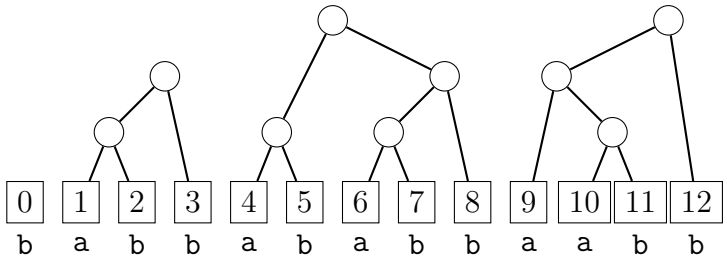
Note the Lyndon factorisation of a word y can be retrieved from its $LynS$ table by sequentially tracing back the starting position of the previous factor, starting from $|y|$. The list of starting positions of factors, in reverse order, is $i_k = |y| - LynS[|y|-1]$, $i_{k-1} = i_k - LynS[i_{k-1} - 1], \ldots, 0$.

Example 10. The Lyndon suffix table of $y_1 =$ `babbababbaabb` is as follows.

j	0	1	2	3	4	5	6	7	8	9	10	11	12
$y[j]$	b	a	b	b	a	b	a	b	b	a	a	b	b
$LynS[j]$	1	1	2	3	1	2	1	2	5	1	1	3	4

Starting positions of factors of its Lyndon factorisation are $9 = 13 - LynS[12]$, $4 = 9 - LynS[8]$, $1 = 4 - LynS[3]$, $0 = 1 - LynS[0]$. The factorisation is `b·abb·ababb·aabb`.

The following depicts the Lyndon forest corresponding to y_1.



Algorithm LeftLyndonForest is merely adapted from the previous algorithm in order to manage Lyndon tree constructions of factors of the Lyndon factorisation while computing the latter. The next proposition is a direct consequence of Proposition 9.

Proposition 11. *Algorithm* LeftLyndonForest *computes the Lyndon forest of a word of length $n > 0$ in time $O(n)$ in the letter-comparison model.*

```
LeftLyndonForest(y non-empty word of length n)
 1  (LynS[0], root[0]) ← (1, 0)
 2  (per, h, i, j) ← (1, 0, 0, 1)
 3  while j < n do
 4      root[j] ← j
 5      if y[j] < y[i] then
 6          h ← j − (i − h)
 7          LynS[h] ← 1
 8          (per, i, j) ← (1, h, h + 1)
 9      elseif y[j] > y[i] then
10          LynS[j] ← j − h + 1
11          j ← j + 1
12          (per, i) ← (j − h, h)
13      else LynS[j] ← LynS[i]
14          (i, j) ← (h + (i − h + 1 mod per), j + 1)
15      ▷ Bundle
16      (p, m, k) ← (root[j], 1, j − 1)
17      while m < LynS[j] do
18          q ← new node ≥ n
19          (left[q], right[q]) ← (root[k], p)
20          (p, m) ← (q, m + LynS[k])
21          k ← k − LynS[k]
22  return root[n − 1]
```

6 Conclusions

In this paper, algorithm LyndonSuffix computes, for a given Lyndon word, its Lyndon suffix table. The Lyndon suffix table is an essential part of algorithm LeftLyndonTree which constructs the left Lyndon tree of a Lyndon word in linear time. We further investigated the prefix standard permutation, initially introduced by Dolce et al. [6], and its relation to the left Lyndon tree. This study resulted in a linear-time algorithm for computing prefix standard permutation in the letter-comparison model. In addition, we exhibited a strong connection between the prefix ranks and the left Lyndon tree. This connection dictates that the order in which the internal nodes of the left Lyndon tree are created and processed coincides with that of the prefix ranks according to infinite ordering.

We finally endeavoured to design a linear-time algorithm LyndonForest which computes the Lyndon forest of a given word. This process entailed modifications

of algorithm LyndonSuffix to create algorithm LongestLyndonSuffix, which enables us to construct the Lyndon suffix table of also non-Lyndon words.

Many interesting questions remain, for example, is there a connection between runs and the internal nodes of the Lyndon forest? Is there a relation between the left and the right Lyndon trees?

References

1. H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta: *The "runs" theorem.* SIAM J. Comput., 46(5) 2017, pp. 1501–1514.
2. M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen: *The maximal number of cubic runs in a word.* J. Comput. Syst. Sci., 78(6) 2012, pp. 1828–1836.
3. M. Crochemore, T. Lecroq, and W. Rytter: *One Twenty Five Problems in Text Algorithms*, Cambridge University Press, 2020, In press.
4. M. Crochemore and L. M. S. Russo: *Cartesian and Lyndon trees.* Theoretical Computer Science, 806 February 2020, pp. 1–9.
5. F. Dolce, A. Restivo, and C. Reutenauer: *On generalized lyndon words.* Theor. Comput. Sci., 777 2019, pp. 232–242.
6. F. Dolce, A. Restivo, and C. Reutenauer: *Some variations on Lyndon words.* CoRR, abs/1904.00954 2019.
7. J. Duval: *Factorizing words over an ordered alphabet.* J. Algorithms, 4(4) 1983, pp. 363–381.
8. C. Hohlweg and C. Reutenauer: *Lyndon words, permutations and trees.* Theor. Comput. Sci., 307(1) 2003, pp. 173–178.
9. R. M. Kolpakov and G. 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, IEEE Computer Society, 1999, pp. 596–604.
10. M. Lothaire: *Combinatorics on Words*, Addison-Wesley, 1983, Reprinted in 1997.
11. R. C. Lyndon: *On Burnside problem i.* Trans. Amer. Math. Soc., 77 1954, pp. 202–215.
12. V. A. Ufnarovskij: *Combinatorial and asymptotic methods in algebra*, in Algebra VI: Combinatorial and Asymptotic Methods of Algebra. Non-Associative Structures, A. Kostrikin and I. Shafarevich, eds., vol. 57 of Encyclopaedia of Mathematical Sciences, Springer, Berlin, 2011, pp. 1–196.
13. G. Viennot: *Algèbres de Lie libres et monoïdes libres*, vol. 691 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1978.