Left Lyndon tree Construction

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

1 Goldsmiths University of London, UK
2 King's College London, UK
3 Université Gustave Eiffel, France

Prague Stringology Conference 2020

Definitions

A word (string) is a sequence of symbols, e.g. `abbaaba`

Definition

Lexicographic order:

Definitions

A word (string) is a sequence of symbols, e.g. `abbaaba`

Definition

Lexicographic order:

- $u \ll v$ if $u = ras$, $v = rbt$
  for words r, s and t and letters a and b with and $a < b$.

Definitions

A word (string) is a sequence of symbols, e.g. `abbaaba`

Definition

Lexicographic order:

- $u \ll v$ if $u = ras$, $v = rbt$
  for words r, s and t and letters a and b with and $a < b$.
- $u < v$, word u is smaller than word v,
  if either $u \ll v$ or u is a proper prefix of v.

Definitions

A word (string) is a sequence of symbols, e.g. `abbaaba`

Definition

Lexicographic order:

- $u \ll v$ if $u = ras$, $v = rbt$
  for words r, s and t and letters a and b with and $a < b$.
- $u < v$, word u is smaller than word v,
  if either $u \ll v$ or u is a proper prefix of v.

Definition

Infinite order:

- $u \prec v$ if $u^\infty < v^\infty$ or both $u^\infty = v^\infty$ and $|u| > |v|$.
  [*Dolce*, *Restivo*, *Reutenauer*, 2019].

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$
3. $u^\infty < w^\infty$

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$
3. $u^\infty < w^\infty$

Binary Lyndon words:

- a, b, ab, aab, abb, . . .

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$
3. $u^\infty < w^\infty$

Binary Lyndon words:

- a, b, ab, aab, abb, . . .

Not Lyndon words:

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$
3. $u^\infty < w^\infty$

Binary Lyndon words:

- `a`, `b`, `ab`, `aab`, `abb`, ...

Not Lyndon words:

- w = `abbab`: for u = `abb` and v = `ab` we get $w > vu$ = `ababb`.

Lyndon Words

Theorem (Lyndon 1954, Ufnarovskij 2011)

A Lyndon word w is either a singleton or defined by any of the following equivalent conditions, in which uv is any non-trivial factorisation of w:

1. $w < vu$
2. $w < v$
3. $u^\infty < w^\infty$

Binary Lyndon words:

- a, b, ab, aab, abb, . . .

Not Lyndon words:

- $w =$ abbab: for $u =$ abb and $v =$ ab we get $w > vu =$ ababb.
- $w =$ abab: for $u =$ ab we get $(\mathtt{ab})^\infty = (\mathtt{abab})^\infty$.

Motivation

- Words are structured by their Lyndon factors.
- Lyndon factorisation: a word factorises uniquely into a decreasing sequence of Lyndon words.
- Can be viewed as a preprocessing step to word algorithms.

Motivation

- Words are structured by their Lyndon factors.
- Lyndon factorisation: a word factorises uniquely into a decreasing sequence of Lyndon words.
- Can be viewed as a preprocessing step to word algorithms.
- Help discovering maximal periodicities in words.
  e.g. [*Banai et al.*, 2017], [*C. et al.*, 2012].

Motivation

- Words are structured by their Lyndon factors.
- Lyndon factorisation: a word factorises uniquely into a decreasing sequence of Lyndon words.
- Can be viewed as a preprocessing step to word algorithms.
- Help discovering maximal periodicities in words.
  e.g. [*Banai et al.*, 2017], [*C. et al.*, 2012].
- Help sorting word suffixes to create a Suffix Array.
  e.g. [*Mantaci et al.*, 2013], [*Louza et al.*, 2019].

Motivation

- Words are structured by their Lyndon factors.
- Lyndon factorisation: a word factorises uniquely into a decreasing sequence of Lyndon words.
- Can be viewed as a preprocessing step to word algorithms.
- Help discovering maximal periodicities in words.
  e.g. [*Banai et al.*, 2017], [*C. et al.*, 2012].
- Help sorting word suffixes to create a Suffix Array.
  e.g. [*Mantaci et al.*, 2013], [*Louza et al.*, 2019].
- etc.

Lyndon Suffix Table

Table *LynS* of a word y is defined, for each position j on y, by

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

Lyndon Suffix Table

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

Let $y = \mathtt{ababbababbabac}$ on the alphabet of constant letters $\{\mathtt{a}, \mathtt{b}, \ldots\}$ ordered as usual $\mathtt{a} < \mathtt{b} < \cdots$.

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
$LynS_y[j]$														

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1													

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1	2												

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1	2	1											

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1	2	1	2										

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1	2	1	2	5									

Lyndon Suffix Table

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

Let $y = \texttt{ababbababbabac}$ on the alphabet of constant letters $\{\texttt{a}, \texttt{b}, \dots\}$ ordered as usual $\texttt{a} < \texttt{b} < \cdots$.

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
$LynS_y[j]$	1	2	1	2	5	1	2	1	2	5	1	2	1	14

Properties of Lyndon Words

- Let z be a word and a a letter for which za is a prefix of a Lyndon word. Let b be a letter with $a < b$. Then zb is a Lyndon word.

Properties of Lyndon Words

- Let z be a word and a a letter for which za is a prefix of a Lyndon word. Let b be a letter with $a < b$. Then zb is a Lyndon word.
- Let u and v, $u < v$, be two Lyndon words, then uv is Lyndon word.

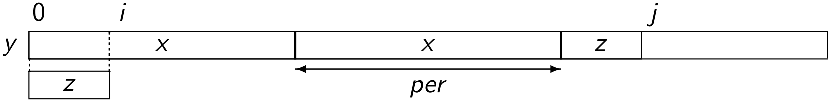
Properties of Lyndon Words

- Let z be a word and a a letter for which za is a prefix of a Lyndon word. Let b be a letter with $a < b$. Then zb is a Lyndon word.
- Let u and v, $u < v$, be two Lyndon words, then uv is Lyndon word.
- A Lyndon word y is borderfree, i.e. $\mathrm{period}(y) = |y|$.

Properties of Lyndon Words

- Let z be a word and a a letter for which za is a prefix of a Lyndon word. Let b be a letter with $a < b$. Then zb is a Lyndon word.
- Let u and v, $u < v$, be two Lyndon words, then uv is Lyndon word.
- A Lyndon word y is borderfree, i.e. $\mathrm{period}(y) = |y|$.

Invariant of *LynS* computation: $w = x^e z$
where x is a Lyndon word and z a proper prefix of x.



If $y[j] > y[i]$ then $y[0\,.\,.\,j]$ is a Lyndon word **with period** $j+1$.
[*Duval*, 1983].

Lyndon Suffix Table - Algorithm

LyndonSuffix(y Lyndon word of length n)

$LynS[0] \leftarrow 1$
$(per, i) \leftarrow (1, 0)$
for $j \leftarrow 1$ **to** $n - 1$ **do**
    if $y[j] \neq y[i]$ **then** $\quad \triangleright\ y[j] > y[i] = y[j - per]$
        $LynS[j] \leftarrow j + 1$
        $(per, i) \leftarrow (j + 1, 0)$
    else $LynS[j] \leftarrow LynS[i]$
        $i \leftarrow i + 1 \bmod per$
return $LynS$

Lyndon Suffix Table - Algorithm

LyndonSuffix(y Lyndon word of length n)

```
LynS[0] ← 1
(per, i) ← (1, 0)
for j ← 1 to n − 1 do
        if y[j] ≠ y[i] then                ▷ y[j] > y[i] = y[j − per]
                LynS[j] ← j + 1
                (per, i) ← (j + 1, 0)
        else    LynS[j] ← LynS[i]
                i ← i + 1 mod per
return LynS
```

Proposition

Algorithm LyndonSuffix *computes the Lyndon suffix table of a Lyndon word of length n in time* $O(n)$.

Left Lyndon tree of a Lyndon word

Let y be a Lyndon word.

- Let u be the longest proper Lyndon prefix of y and $y = uv$.
  Then v is a Lyndon word.
  uv is the **left Lyndon factorisation** of y.
  aaaababbaabaab = aaaababbaab · aab

Left Lyndon tree of a Lyndon word

Let y be a Lyndon word.

- Let u be the longest proper Lyndon prefix of y and $y = uv$.
  Then v is a Lyndon word.
  uv is the **left Lyndon factorisation** of y.
  $\texttt{aaaababbaabaab} = \texttt{aaaababbaab} \cdot \texttt{aab}$
- Let v be the longest proper Lyndon suffix of y and $y = uv$. Then u is a Lyndon word. uv is the **right Lyndon factorisation** of y.
  $\texttt{aaaababbaabaab} = \texttt{a} \cdot \texttt{aaababbaabaab}$

Left Lyndon tree of a Lyndon word

Let y be a Lyndon word.

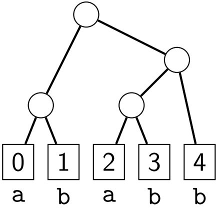
- Let u be the longest proper Lyndon prefix of y and $y = uv$. Then v is a Lyndon word. uv is the **left Lyndon factorisation** of y.
  $\mathtt{aaaababbaabaab} = \mathtt{aaaababbaab} \cdot \mathtt{aab}$
- Let v be the longest proper Lyndon suffix of y and $y = uv$. Then u is a Lyndon word. uv is the **right Lyndon factorisation** of y.
  $\mathtt{aaaababbaabaab} = \mathtt{a} \cdot \mathtt{aaababbaabaab}$

Left Lyndon tree of y:

- Obtained by recursive application of left Lyndon factorisation:
  $\mathtt{ababb} = \mathtt{ab} \cdot \mathtt{abb} = (\mathtt{a} \cdot \mathtt{b}) \cdot (\mathtt{ab} \cdot \mathtt{b}) = (\mathtt{a} \cdot \mathtt{b}) \cdot ((\mathtt{a} \cdot \mathtt{b}) \cdot \mathtt{b})$

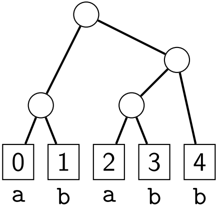
Left, right Lyndon trees

Left Lyndon tree of ababb:

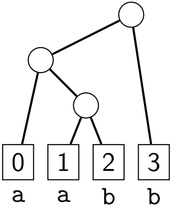


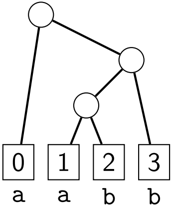
Left, right Lyndon trees

Left Lyndon tree of ababb:



Left and right Lyndon trees of aabb:





Left Lyndon Tree - Algorithm

```
LeftLyndonTree(y Lyndon word of length n)
(LynS[0], root[0]) ← (1, 0)
(per, i) ← (1, 0)
for j ← 1 to n − 1 do
    root[j] ← j
    if y[j] ≠ y[i] then                ▷ y[j] > y[i] = y[j − per]
        LynS[j] ← j + 1
        (per, i) ← (j + 1, 0)
    else LynS[j] ← LynS[i]
        i ← i + 1 mod per
    (ℓ, k) ← (1, j − 1)
    while ℓ < LynS[j] do
        q ← new node ≥ n
        (left[q], right[q]) ← (root[k], root[j])
        root[j] ← q
        (ℓ, k) ← (ℓ + LynS[k], k − LynS[k])
return root[n − 1]
```

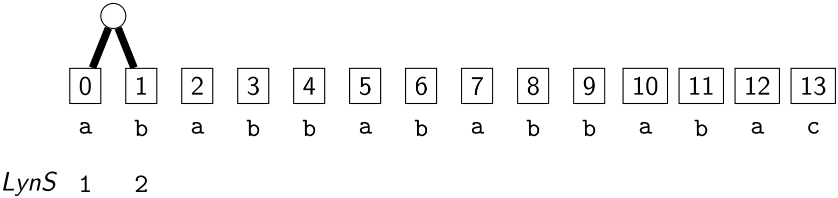
Left Lyndon Tree - Algorithm

```
LeftLyndonTree(y Lyndon word of length n)
(LynS[0], root[0]) ← (1, 0)
(per, i) ← (1, 0)
for j ← 1 to n − 1 do
        root[j] ← j
        if y[j] ≠ y[i] then                ▷ y[j] > y[i] = y[j − per]
                LynS[j] ← j + 1
                (per, i) ← (j + 1, 0)
        else    LynS[j] ← LynS[i]
                i ← i + 1 mod per
        (ℓ, k) ← (1, j − 1)
        while ℓ < LynS[j] do
                q ← new node ≥ n
                (left[q], right[q]) ← (root[k], root[j])
                root[j] ← q
                (ℓ, k) ← (ℓ + LynS[k], k − LynS[k])
return root[n − 1]
```

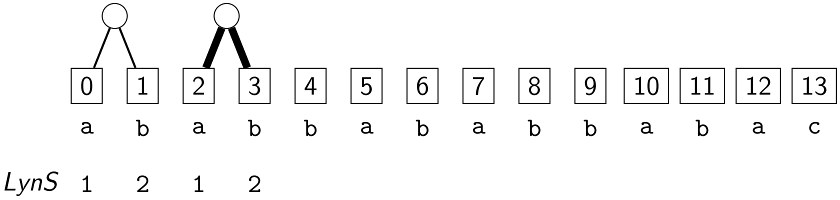
Theorem

Algorithm LeftLyndonTree *builds the left Lyndon tree of a Lyndon word of length n in time $O(n)$.*

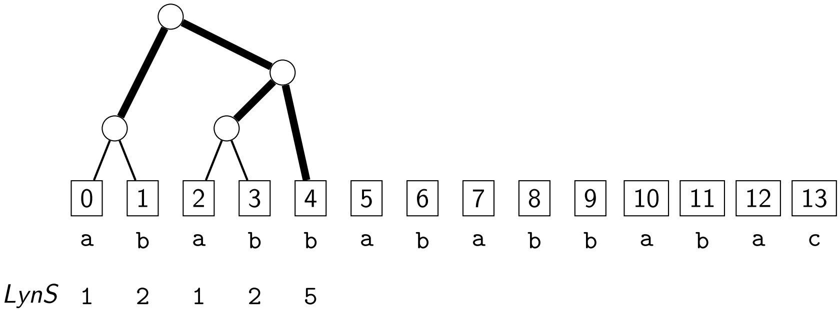
Building the tree



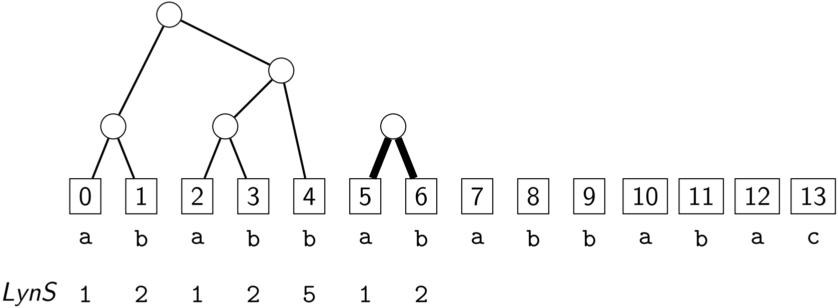
Building the tree

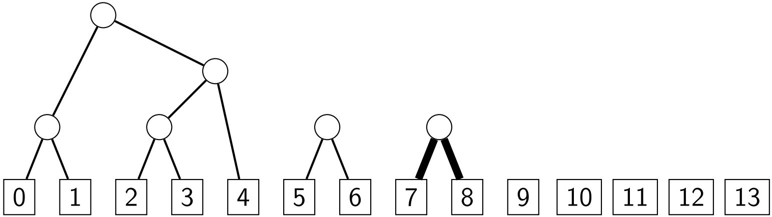


Building the tree



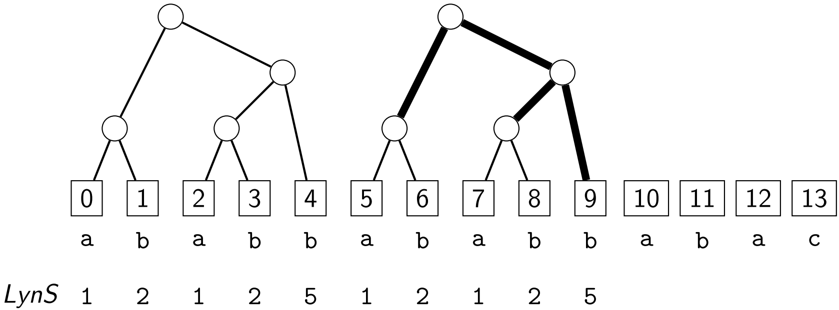
Building the tree



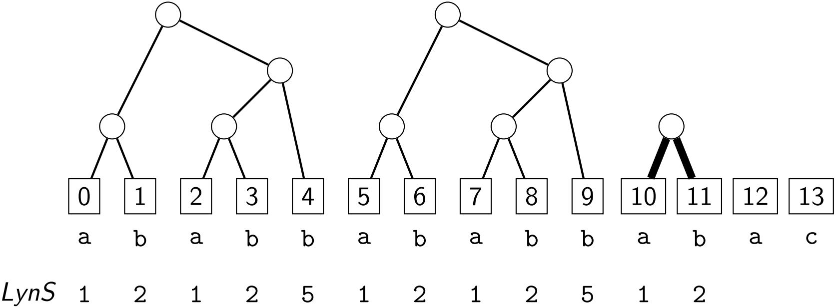


	a	b	a	b	b	a	b	a	b	b	a	b	a	c
LynS	1	2	1	2	5	1	2	1	2					

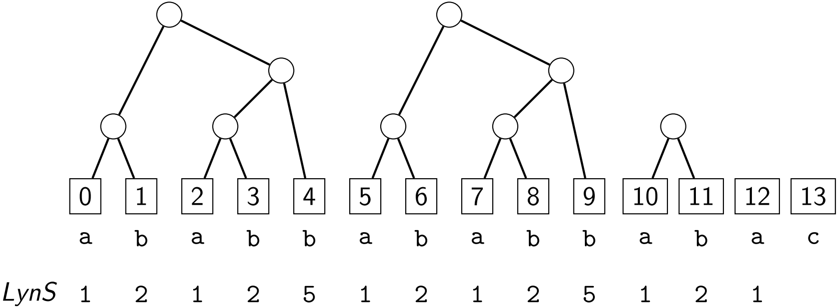
Building the tree



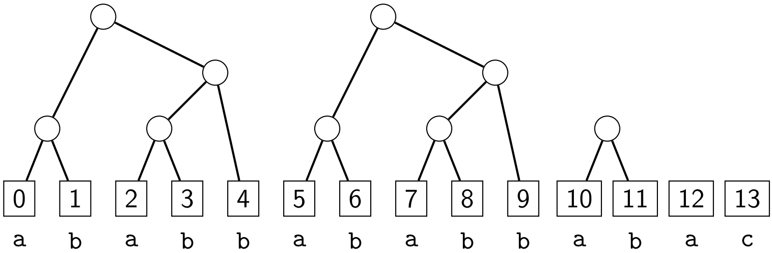
Building the tree



Building the tree

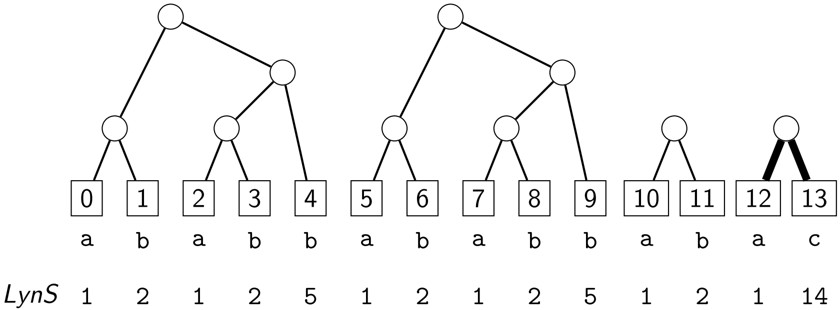


Building the tree

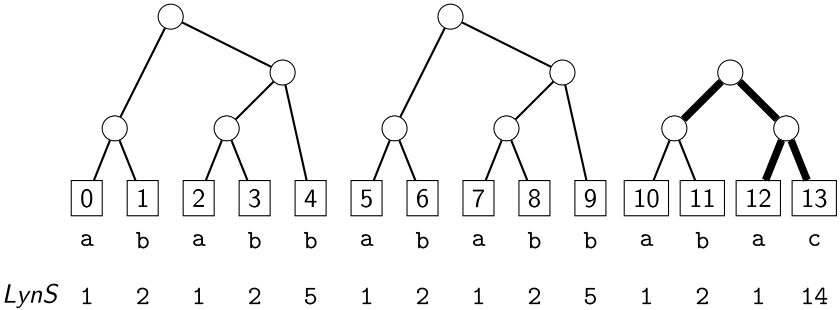


LynS	1	2	1	2	5	1	2	1	2	5	1	2	1	14

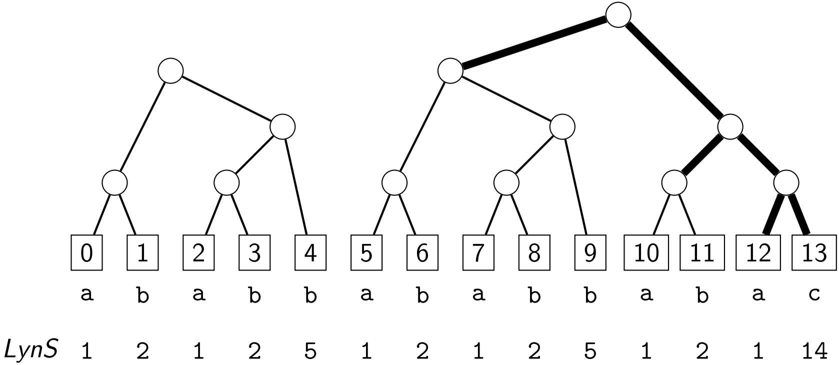
Building the tree



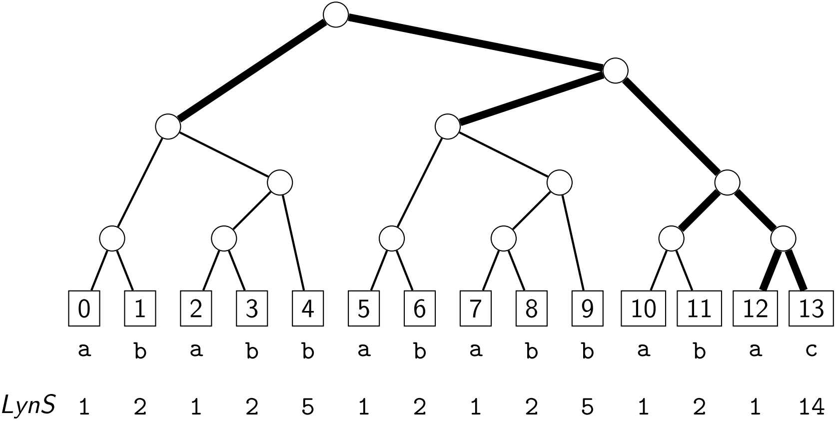
Building the tree



Building the tree



Building the tree



Theorem

Algorithm LEFTLYNDONTREE *builds the left Lyndon tree of a Lyndon word with a left-to-right postorder tree traversal.*

Prefix standard permutation

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	`a. a`
2		`a b. a b. a b. a b. a b. a b`
3		`a b a. a b a`
4		`a b a b. a b a b. a b a b`
5		`a b a b b. a b a b b`
6		`a b a b b a. a b a b b`
7		`a b a b b a b. a b a b b a b`
8		`a b a b b a b a. a b a b b a b a`
9		`a b a b b a b a b. a b a b b a b a b`
10		`a b a b b a b a b b. a b a b b a b a b b`
11		`a b a b b a b a b b a. a b a b b a b a b b`
12		`a b a b b a b a b b a b. a b a b b a b a b`
13		`a b a b b a b a b b a b a. a b a b b a b a`

Prefix standard permutation

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2		a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4		a b a b. a b a b. a b a b
5		a b a b b. a b a b b
6		a b a b b a. a b a b b
7		a b a b b a b. a b a b b a b
8		a b a b b a b a. a b a b b a b a
9		a b a b b a b a b. a b a b b a b a b
10		a b a b b a b a b b. a b a b b a b a b b
11		a b a b b a b a b b a. a b a b b a b a b b a
12		a b a b b a b a b b a b. a b a b b a b a b b a
13		a b a b b a b a b b a b a. a b a b b a b a b b

Prefix standard permutation

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2	4	a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4	3	a b a b. a b a b. a b a b
5		a b a b b. a b a b b
6		a b a b b a. a b a b b
7		a b a b b a b. a b a b b a b
8		a b a b b a b a. a b a b b a b a
9		a b a b b a b a b. a b a b b a b a b
10		a b a b b a b a b b. a b a b b a b a b b
11		a b a b b a b a b b a. a b a b b a b a b b
12		a b a b b a b a b b a b. a b a b b a b a b
13		a b a b b a b a b b a b a. a b a b b a b a

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2	4	a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4	3	a b a b. a b a b. a b a b
5		a b a b b. a b a b b
6	5	a b a b b a. a b a b b
7		a b a b b a b. a b a b b a b
8		a b a b b a b a. a b a b b a b a
9		a b a b b a b a b. a b a b b a b a b
10		a b a b b a b a b b. a b a b b a b a b b
11		a b a b b a b a b b a. a b a b b a b a b b
12		a b a b b a b a b b a b. a b a b b a b a b
13		a b a b b a b a b b a b a. a b a b b a b a

Prefix standard permutation

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2	4	a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4	3	a b a b. a b a b. a b a b
5		a b a b b. a b a b b
6	5	a b a b b a. a b a b b
7	8	a b a b b a b. a b a b b a b
8	6	a b a b b a b a. a b a b b a b a
9	7	a b a b b a b a b. a b a b b a b a b
10		a b a b b a b a b b. a b a b b a b a b b
11		a b a b b a b a b b a. a b a b b a b a b b
12		a b a b b a b a b b a b. a b a b b a b a b
13		a b a b b a b a b b a b a. a b a b b a b a

Prefix standard permutation

Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2	4	a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4	3	a b a b. a b a b. a b a b
5		a b a b b. a b a b b
6	5	a b a b b a. a b a b b
7	8	a b a b b a b. a b a b b a b
8	6	a b a b b a b a. a b a b b a b a
9	7	a b a b b a b a b. a b a b b a b a b
10		a b a b b a b a b b. a b a b b a b a b b
11	9	a b a b b a b a b b a. a b a b b a b a b b
12	11	a b a b b a b a b b a b. a b a b b a b a b
13	10	a b a b b a b a b b a b a. a b a b b a b a

Prefix standard permutation

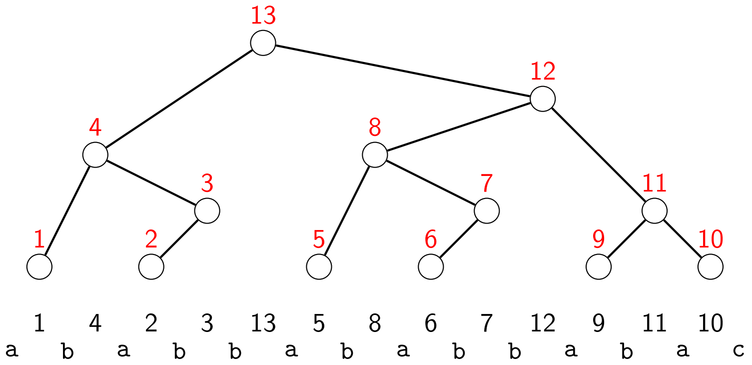
Ranks according to the infinite ordering $\prec$.
Permutation $\mathrm{psp} = \mathrm{rank}^{-1}$.

j	$\mathrm{rank}[j]$	
1	1	a a
2	4	a b. a b. a b. a b. a b. a b
3	2	a b a. a b a
4	3	a b a b. a b a b. a b a b
5	13	a b a b b. a b a b b
6	5	a b a b b a. a b a b b
7	8	a b a b b a b. a b a b b a b
8	6	a b a b b a b a. a b a b b a b a
9	7	a b a b b a b a b. a b a b b a b a b
10	12	a b a b b a b a b b. a b a b b a b a b b
11	9	a b a b b a b a b b a. a b a b b a b a b b
12	11	a b a b b a b a b b a b. a b a b b a b a b
13	10	a b a b b a b a b b a b a. a b a b b a b a

Cartesian tree of prefix ranks

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
$\text{rank}[j]$	1	4	2	3	13	5	8	6	7	12	9	11	10	

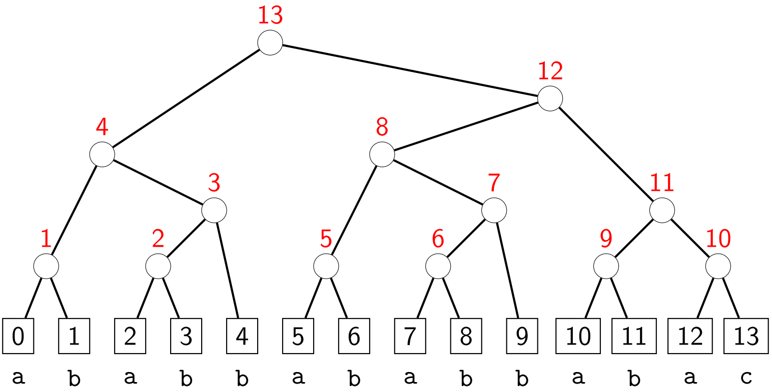
$\text{psp}(y) = (0, 2, 3, 1, 5, 7, 8, 6, 10, 12, 11, 9, 4)$



Prefix ranks and Left Lyndon tree

Theorem (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.



Prefix sorting

Theorem

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

Proof: In Algorithm LeftLyndonTree, list prefixes associated to internal nodes instead of building the tree.

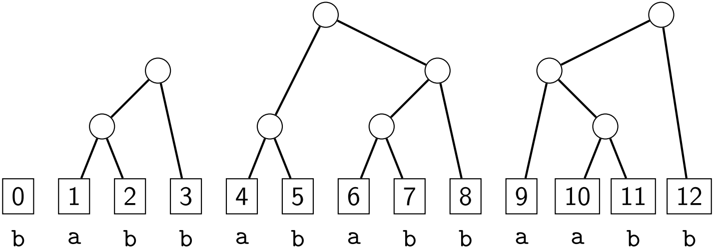
Lyndon Forest: input is any non-empty word

Algorithms extend to factors of the Lyndon factorisation of a non-empty word (algorithm by [*Duval*, 1983]).

Example

The Lyndon suffix table of y = 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



Works on hand

Reverse engineering

- On a binary alphabet, function psp is one-to-one.
- Given a permutation p of $\{0, 1, \ldots, n-2\}$, the word y of length n for which $\mathrm{psp}(y) = p$ can be found in linear time.

Works on hand

Reverse engineering

- On a binary alphabet, function psp is one-to-one.
- Given a permutation p of $\{0, 1, \ldots, n-2\}$, the word y of length n for which $\mathrm{psp}(y) = p$ can be found in linear time.

Relation between left and right Lyndon trees

Works on hand

Reverse engineering

- On a binary alphabet, function psp is one-to-one.
- Given a permutation p of $\{0, 1, \ldots, n-2\}$, the word y of length n for which $\mathrm{psp}(y) = p$ can be found in linear time.

Relation between left and right Lyndon trees

Right Lyndon tree

- Recursive application of the right Lyndon factorisation of a Lyndon word (see [*Holweg*, *Reutenauer*, 2003]).
- Can be computed in linear time when suffixes are sorted.
- Conjecture: $\Omega(n \log n)$ lower bound on a general alphabet.
- How much can the right Lyndon tree of y help sort its suffixes?