Sorting suffixes of a text via its Lyndon Factorization

Sabrina Mantaci, Antonio Restivo,
Giovanna Rosone and Marinella Sciortino

Dipartimento di Matematica e Informatica
University of Palermo
Palermo, ITALY

Incontro di Combinatoria delle Parole
Progetto PRIN 2010/2011
"Automi e Linguaggi Formali: aspetti matematici e applicativi"
Palermo, 10-11 ottobre 2013

The sorting of the suffixes

Our goal

The goal is to introduce a new strategy for sorting the suffixes of a word w.

- The process of sorting the suffixes of a word plays a fundamental role in *Text Algorithms* with several applications in many areas of Computer Science and Bioinformatics.
- For instance, it is a fundamental step, in implicit or explicit way, for the construction of
  - the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
  - the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

The sorting of the suffixes

Our goal

The goal is to introduce a new strategy for sorting the suffixes of a word w.

- The process of sorting the suffixes of a word plays a fundamental role in *Text Algorithms* with several applications in many areas of Computer Science and Bioinformatics.
- For instance, it is a fundamental step, in implicit or explicit way, for the construction of
  - the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
  - the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

The sorting of the suffixes

Our goal

The goal is to introduce a new strategy for sorting the suffixes of a word w.

- The process of sorting the suffixes of a word plays a fundamental role in *Text Algorithms* with several applications in many areas of Computer Science and Bioinformatics.
- For instance, it is a fundamental step, in implicit or explicit way, for the construction of
  - the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
  - the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

The sorting of the suffixes

Our goal

The goal is to introduce a new strategy for sorting the suffixes of a word w.

- The process of sorting the suffixes of a word plays a fundamental role in *Text Algorithms* with several applications in many areas of Computer Science and Bioinformatics.
- For instance, it is a fundamental step, in implicit or explicit way, for the construction of
  - the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
  - the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

The sorting of the suffixes

Our goal

The goal is to introduce a new strategy for sorting the suffixes of a word w.

- The process of sorting the suffixes of a word plays a fundamental role in *Text Algorithms* with several applications in many areas of Computer Science and Bioinformatics.
- For instance, it is a fundamental step, in implicit or explicit way, for the construction of
  - the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
  - the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

Sorting suffixes by Lyndon factorization

Our idea

Our strategy uses the *Lyndon factorization* and is based on a combinatorial property that allows to sort the suffixes of w ("global suffixes") by using the sorting of the suffixes inside blocks of consecutive Lyndon factors of the decomposition ("local suffixes").

Lyndon Words

- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A word $w \in \Sigma^+$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.

Definition

A *Lyndon word* is a (primitive) word that is smaller in lexicographic order than all of its conjugates.

Example

- $u = mathematics$ is not a Lyndon word;
- $v = athematicsm$ is a Lyndon word.

There exist linear algorithms for the computation of the Lyndon word of a given word [*Duval*, 1983].

Lyndon Words

- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A word $w \in \Sigma^+$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.

Definition

A *Lyndon word* is a (primitive) word that is smaller in lexicographic order than all of its conjugates.

Example

- $u = mathematics$ is not a Lyndon word;
- $v = athematicsm$ is a Lyndon word.

There exist linear algorithms for the computation of the Lyndon word of a given word [*Duval*, 1983].

Lyndon Words

- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A word $w \in \Sigma^+$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.

Definition

A *Lyndon word* is a (primitive) word that is smaller in lexicographic order than all of its conjugates.

Example

- $u = mathematics$ is not a Lyndon word;
- $v = athematicsm$ is a Lyndon word.

There exist linear algorithms for the computation of the Lyndon word of a given word [*Duval*, 1983].

Lyndon Words

- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A word $w \in \Sigma^+$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.

Definition

A *Lyndon word* is a (primitive) word that is smaller in lexicographic order than all of its conjugates.

Example

- $u = mathematics$ is not a Lyndon word;
- $v = athematicsm$ is a Lyndon word.

There exist linear algorithms for the computation of the Lyndon word of a given word [*Duval*, 1983].

Lyndon Factorization

Theorem (*Chen, Fox and Lyndon*: 1958)

Every word $w \in \Sigma^+$ has a unique factorization $w = L_1 \cdots L_k$ such that

$$L_1 \geq \cdots \geq L_k$$

is a non-increasing sequence of Lyndon words.

Let $w = abaaaabaaaaabaaaabaaaaaab$. The Lyndon factorization of w is

$$ab|aaaab|aaaaabaaaab|aaaaaab$$

Note that each L_i is strictly less than any of its proper conjugates/suffixes.

The Lyndon factorization of a given word can be computed

- in linear time [*Duval*, 1983];
- in parallel way [*Apostolico and Crochemore*, 1989] and [*Daykin, Iliopoulos and Smyth*, 1994];
- in external memory [*Roh, Crochemore, Iliopoulos and Park*, 2008].

Lyndon Factorization

Theorem (*Chen, Fox and Lyndon*: 1958)

Every word $w \in \Sigma^+$ has a unique factorization $w = L_1 \cdots L_k$ such that

$$L_1 \geq \cdots \geq L_k$$

is a non-increasing sequence of Lyndon words.

Let $w = abaaaabaaaaabaaaabaaaaaab$. The Lyndon factorization of w is

$$ab|aaaab|aaaaabaaaab|aaaaaab$$

Note that each L_i is strictly less than any of its proper conjugates/suffixes.

The Lyndon factorization of a given word can be computed

- in linear time [*Duval*, 1983];
- in parallel way [*Apostolico and Crochemore*, 1989] and [*Daykin, Iliopoulos and Smyth*, 1994];
- in external memory [*Roh, Crochemore, Iliopoulos and Par*, 2008].

Lyndon Factorization

Theorem (*Chen, Fox and Lyndon*: 1958)

Every word $w \in \Sigma^+$ has a unique factorization $w = L_1 \cdots L_k$ such that

$$L_1 \geq \cdots \geq L_k$$

is a non-increasing sequence of Lyndon words.

Let $w = abaaaabaaaaabaaaabaaaaaab$. The Lyndon factorization of w is

$$ab|aaaab|aaaaabaaaab|aaaaaab$$

Note that each L_i is strictly less than any of its proper conjugates/suffixes.

The Lyndon factorization of a given word can be computed

- in linear time [*Duval*, 1983];
- in parallel way [*Apostolico and Crochemore*, 1989] and [*Daykin, Iliopoulos and Smyth*, 1994];
- in external memory [*Roh, Crochemore, Iliopoulos and Par*, 2008]

Lyndon Factorization

Theorem (*Chen, Fox and Lyndon*: 1958)

Every word $w \in \Sigma^+$ *has a unique factorization* $w = L_1 \cdots L_k$ *such that*

$$L_1 \geq \cdots \geq L_k$$

is a non-increasing sequence of Lyndon words.

Let $w = abaaaabaaaaabaaaabaaaaaab$. The Lyndon factorization of w is

$$ab|aaaab|aaaaabaaaab|aaaaaab$$

Note that each L_i is strictly less than any of its proper conjugates/suffixes.

The Lyndon factorization of a given word can be computed

- in linear time [*Duval*, 1983];
- in parallel way [*Apostolico and Crochemore*, 1989] and [*Daykin, Iliopoulos and Smyth*, 1994];
- in external memory [*Roh, Crochemore, Iliopoulos and Par*, 2008].

Local and Global suffixes

For each factor u of w, we denote by $first(u)$ and $last(u)$ the position of the first and the last symbol, respectively, of the factor u in w.

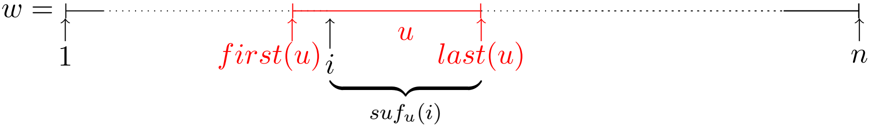
We denote by

- $suf_u(i) = w[i, last(u)]$ and we call it *local suffix* at the position i with respect to u.
- $suf(i) = w[i, n]$ and we call it *global suffix* of w at the position i.

Local and Global suffixes

For each factor u of w, we denote by $first(u)$ and $last(u)$ the position of the first and the last symbol, respectively, of the factor u in w.
We denote by

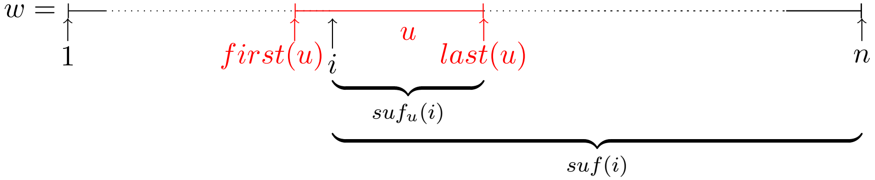
- $suf_u(i) = w[i, last(u)]$ and we call it *local suffix* at the position i with respect to u.
- $suf(i) = w[i, n]$ and we call it *global suffix* of w at the position i.



Local and Global suffixes

For each factor u of w, we denote by $first(u)$ and $last(u)$ the position of the first and the last symbol, respectively, of the factor u in w.
We denote by

- $suf_u(i) = w[i, last(u)]$ and we call it *local suffix* at the position i with respect to u.
- $suf(i) = w[i, n]$ and we call it *global suffix* of w at the position i.



Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Compatible sorting

Definition

Let w be a word and let u be a factor of w. We say that the sorting of the *local* suffixes with respect to u is *compatible* with the sorting of the *global* suffixes of w if for all i, j with $first(u) \leq i < j \leq last(u)$,

$$suf_u(i) < suf_u(j) \iff suf(i) < suf(j).$$

In general, taken an arbitrary factor of a word w, the sorting of its suffixes is *not compatible* with the sorting of the suffixes of w, as the following example shows.

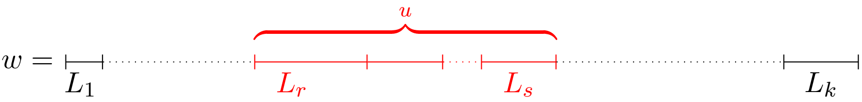
Example

Consider the word $w = abababb$ and its factor $u = ababa$.
Then $suf_u(1) = ababa > a = suf_u(5)$
whereas $suf(1) = abababb < abb = suf(5)$.
Such sorting is not compatible.

Our result

Theorem

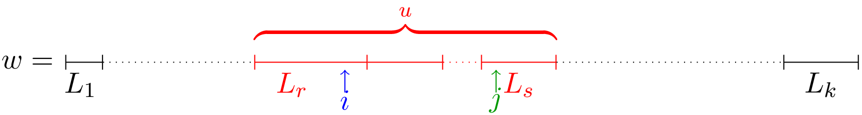
Let $w \in \Sigma^*$ and let $w = L_1 L_2 \cdots L_k$ be its Lyndon factorization. For each factor $u = L_r L_{r+1} \cdots L_s$, the sorting of the local suffixes with respect to u is compatible with the sorting of the global suffixes of w.



Our result

Theorem

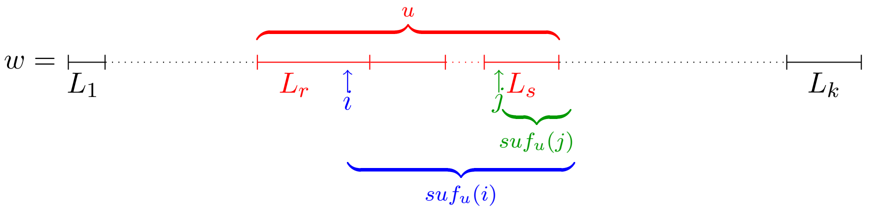
Let $w \in \Sigma^*$ and let $w = L_1 L_2 \cdots L_k$ be its Lyndon factorization. For each factor $u = L_r L_{r+1} \cdots L_s$, the sorting of the local suffixes with respect to u is compatible with the sorting of the global suffixes of w.



Our result

Theorem

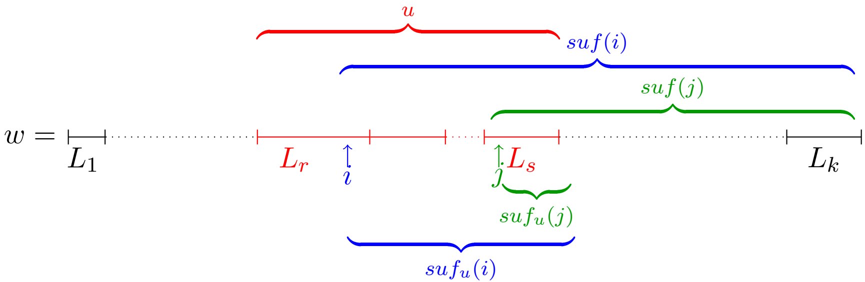
Let $w \in \Sigma^*$ and let $w = L_1 L_2 \cdots L_k$ be its Lyndon factorization. For each factor $u = L_r L_{r+1} \cdots L_s$, the sorting of the local suffixes with respect to u is compatible with the sorting of the global suffixes of w.



Our result

Theorem

Let $w \in \Sigma^*$ and let $w = L_1 L_2 \cdots L_k$ be its Lyndon factorization. For each factor $u = L_r L_{r+1} \cdots L_s$, the sorting of the local *suffixes with respect to u is* compatible *with the sorting of the* global *suffixes of w.*

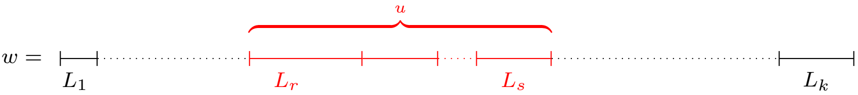


Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

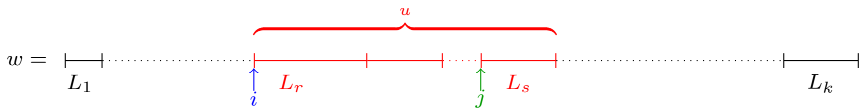
We don't need to compare any symbol.

Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

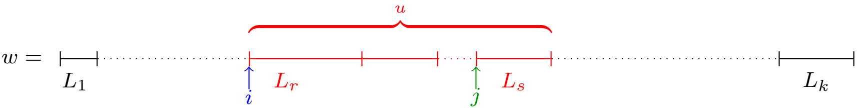
We don't need to compare any symbol.

Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

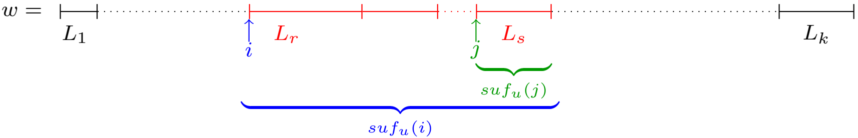
We don't need to compare any symbol.

Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

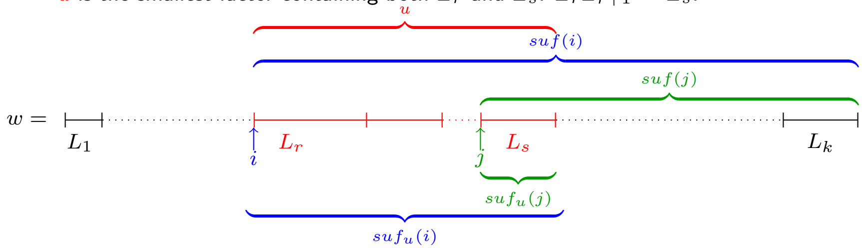
We don't need to compare any symbol.

Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

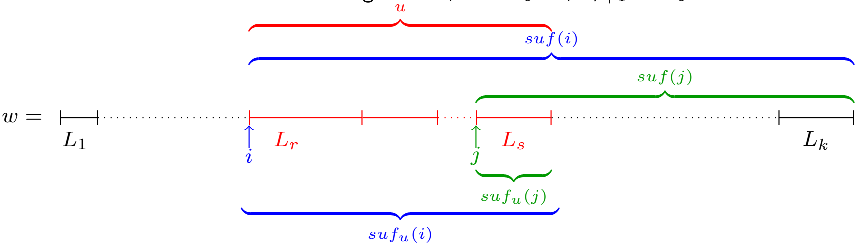
We don't need to compare any symbol.

Easy case

The theorem is trivially true when the two suffixes start with two different Lyndon factors.

Suppose that

- i is the position of the first symbol of L_r
- j is the position of the first symbol of L_s
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.



Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that

- $L_r L_{r+1} \cdots L_s > L_s$
- $L_r L_{r+1} \cdots L_k > L_s L_{s+1} \cdots L_k$

We don't need to compare any symbol.

Other cases

The theorem is true when the two suffixes of w start inside the same factor u of consecutive Lyndon words.

Suppose that

- i is a position inside L_r;
- j is a position inside L_s;
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.

$$suf(i) = \underbrace{L_r[i, last(L_r)] \; L_{r+1} \cdots L_s}_{suf_u(i)} \cdots L_k$$

$$suf(j) = \underbrace{L_s[j, last(L_s)]}_{suf_u(j)} \; L_{s+1} \cdots L_k$$

How many symbol comparisons we need to establish the order relation between $suf(i)$ and $suf(j)$?

Other cases

The theorem is true when the two suffixes of w start inside the same factor u of consecutive Lyndon words.

Suppose that

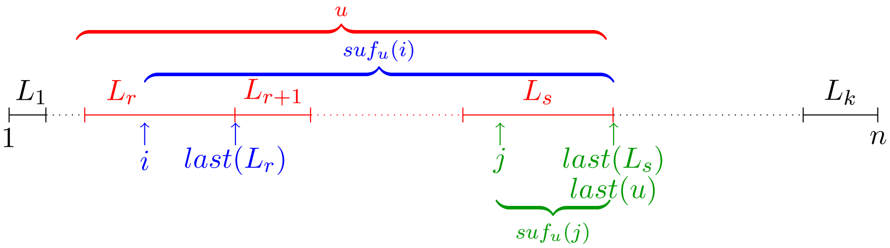
- i is a position inside L_r;
- j is a position inside L_s;
- u is the smallest factor containing both L_r and L_s: $L_r L_{r+1} \cdots L_s$.

$$suf(i) = \underbrace{L_r[i, last(L_r)]\; L_{r+1} \cdots L_s}_{suf_u(i)} \cdots L_k$$

$$suf(j) = \underbrace{L_s[j, last(L_s)]}_{suf_u(j)}\; L_{s+1} \cdots L_k$$

How many symbol comparisons we need to establish the order relation between $suf(i)$ and $suf(j)$?

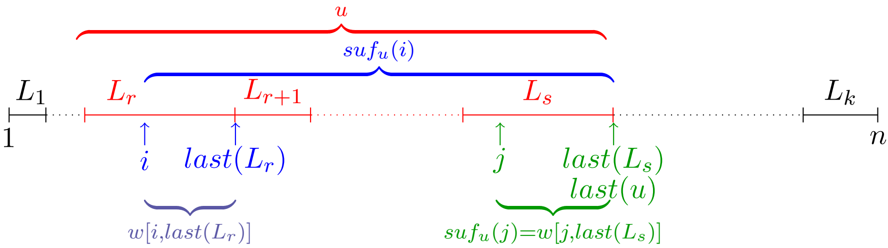
How many symbol comparisons?



Possible cases:

- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- There is not a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$:
  - $w[i, last(L_r)] = w[j, last(L_s)]$;
  - $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$;
  - $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$.

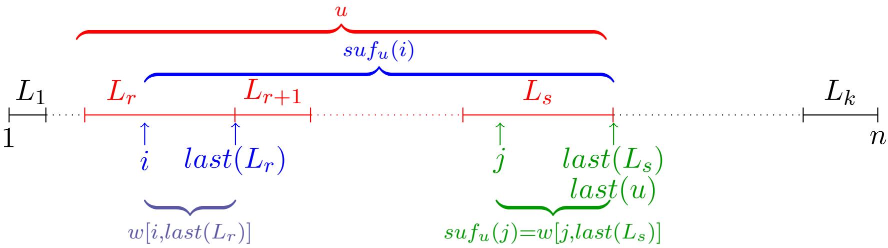
How many symbol comparisons?



Possible cases:

- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- There is not a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$:
  - $w[i, last(L_r)] = w[j, last(L_s)]$;
  - $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$;
  - $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$.

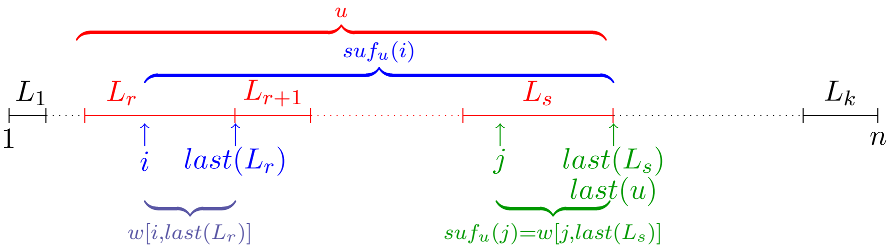
How many symbol comparisons?



Possible cases:

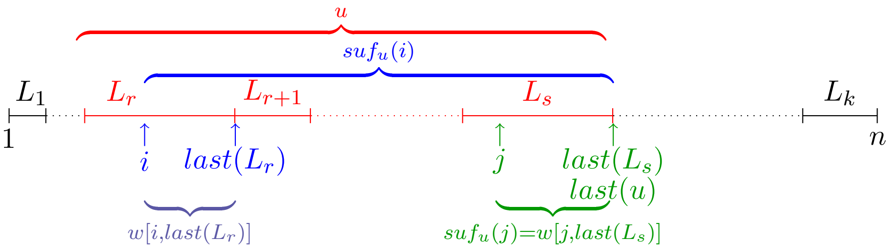
- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- There is not a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$:
  - $w[i, last(L_r)] = w[j, last(L_s)]$;
  - $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$;
  - $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$.

First case



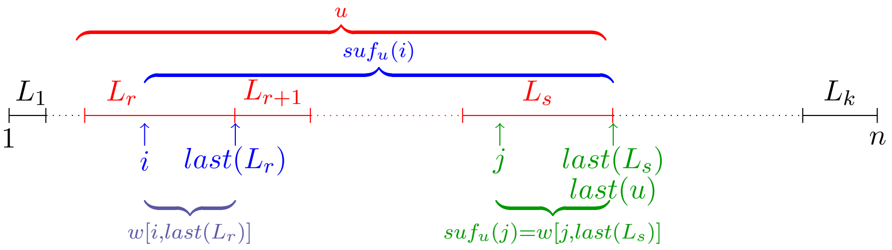
- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- It is easy to verify that the order relation between the local and the global suffixes is the same!
- We need $lcp(i, j) + 1 \leq min(|w[i, last(L_r)]|, |w[j, last(L_s)]|)$ symbol comparisons, where $lcp(i, j)$ denotes the length of the longest common prefix between the suffixes $w[i, n]$ and $w[j, n]$.

First case



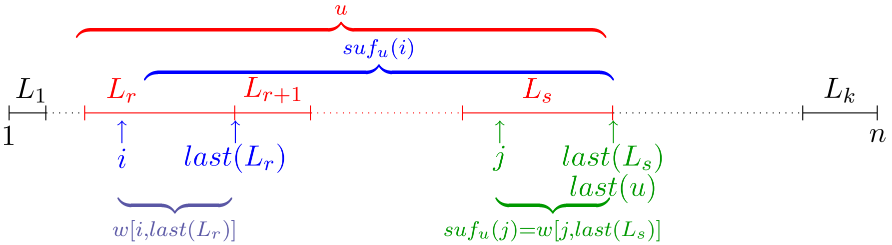
- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- It is easy to verify that the order relation between the local and the global suffixes is the same!
- We need $lcp(i, j) + 1 \leq min(|w[i, last(L_r)]|, |w[j, last(L_s)]|)$ symbol comparisons, where $lcp(i, j)$ denotes the length of the longest common prefix between the suffixes $w[i, n]$ and $w[j, n]$.

First case



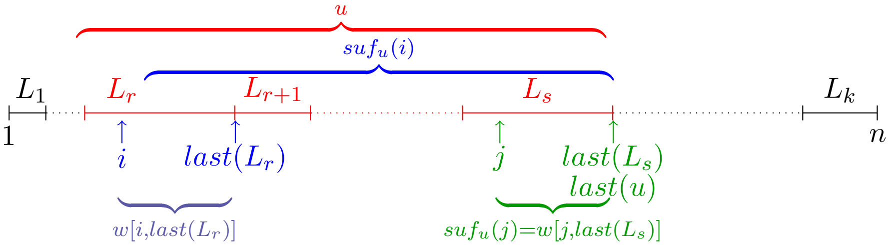
- There is a different symbol inside $w[i, last(L_r)]$ and $w[j, last(L_s)]$.
- It is easy to verify that the order relation between the local and the global suffixes is the same!
- We need $lcp(i,j) + 1 \leq min(|w[i, last(L_r)]|, |w[j, last(L_s)]|)$ symbol comparisons, where $lcp(i,j)$ denotes the length of the longest common prefix between the suffixes $w[i,n]$ and $w[j,n]$.

Second case: $w[i, last(L_r)] = w[j, last(L_s)]$



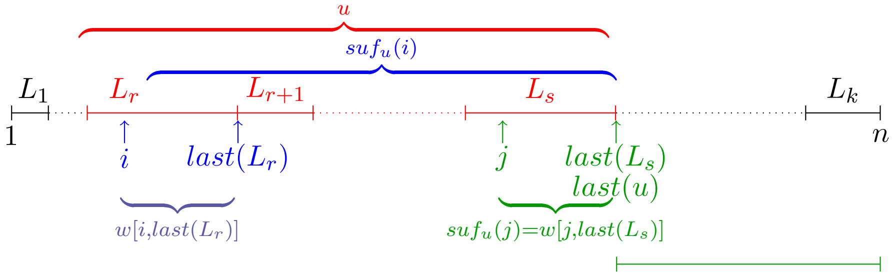
- Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]| = |w[i, last(L_r)]|$ symbol comparisons.

Second case: $w[i, last(L_r)] = w[j, last(L_s)]$



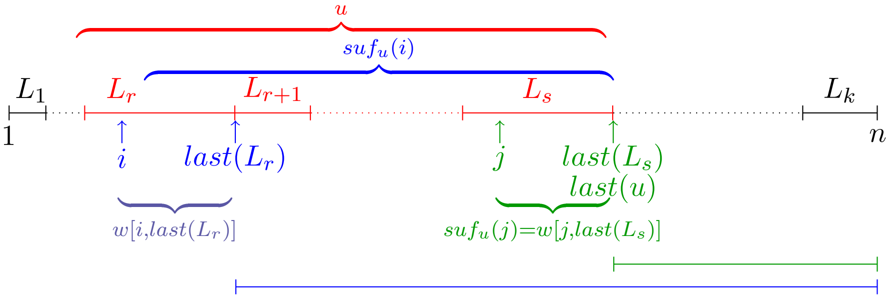
- Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]| = |w[i, last(L_r)]|$ symbol comparisons.

Second case: $w[i, last(L_r)] = w[j, last(L_s)]$



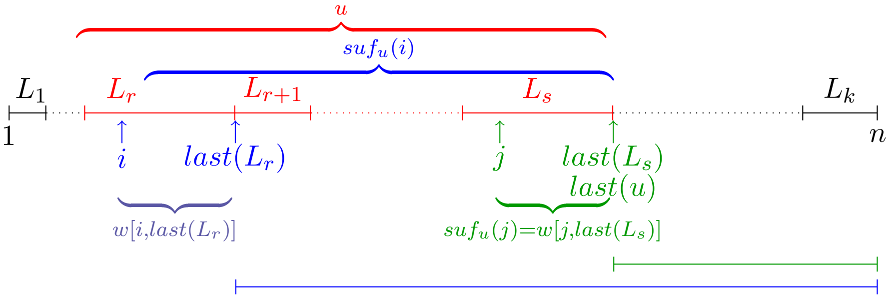
- Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]| = |w[i, last(L_r)]|$ symbol comparisons.

Second case: $w[i, last(L_r)] = w[j, last(L_s)]$



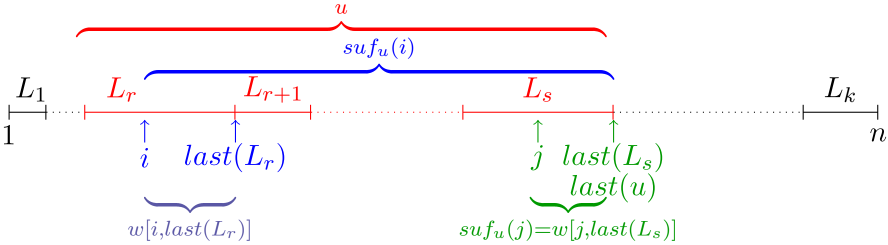
- Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]| = |w[i, last(L_r)]|$ symbol comparisons.

Second case: $w[i, last(L_r)] = w[j, last(L_s)]$



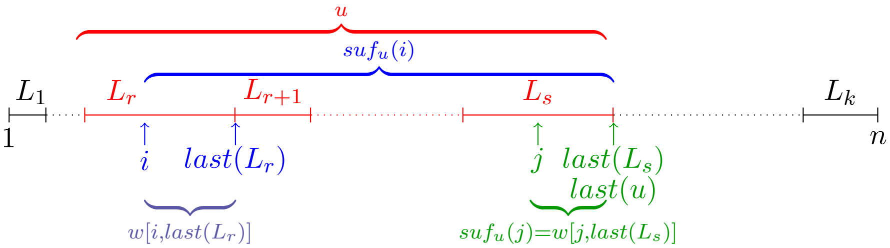
- Since $r < s$ and $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$. It is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]| = |w[i, last(L_r)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



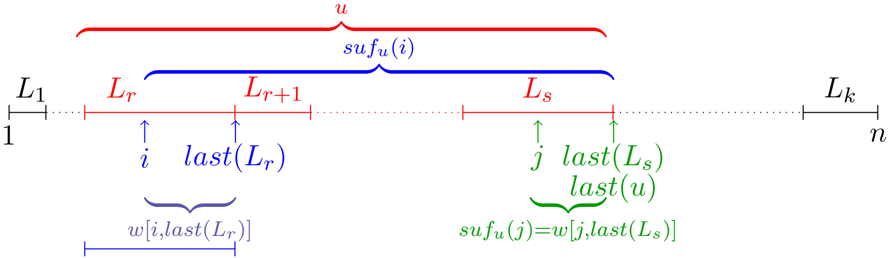
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is easy to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



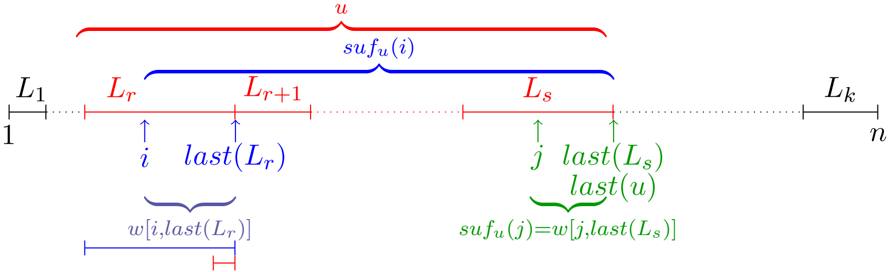
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



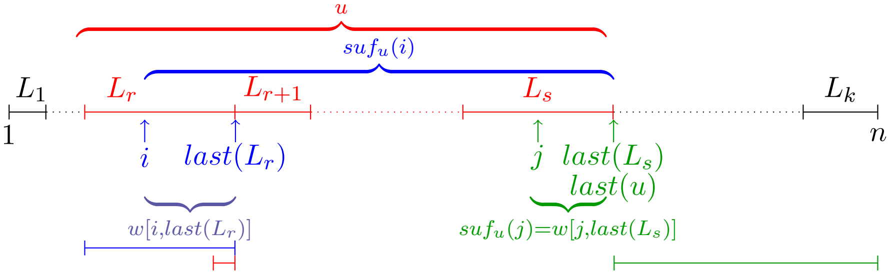
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



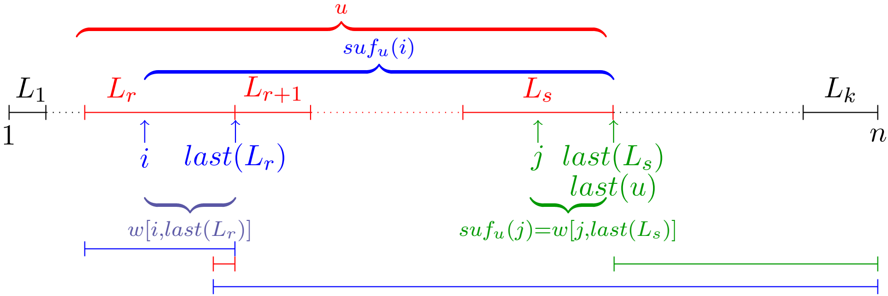
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



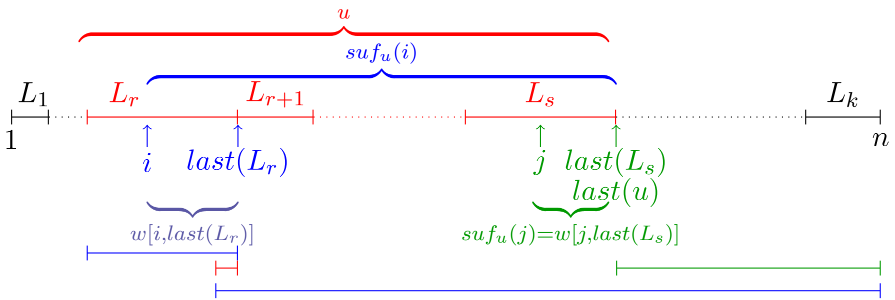
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



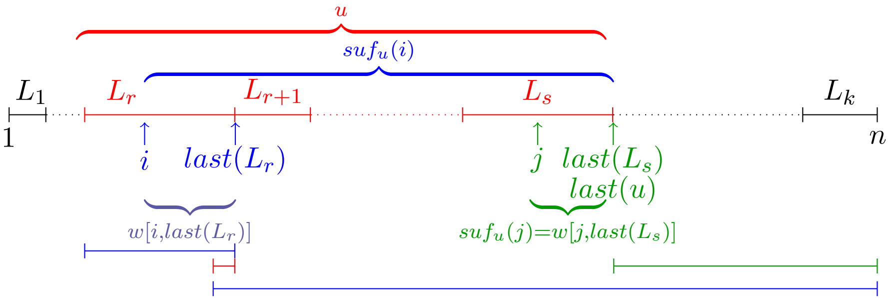
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



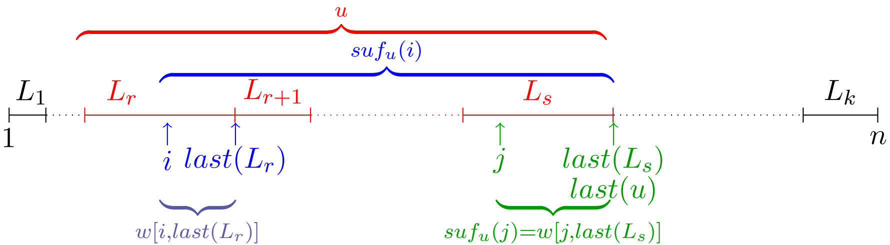
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same! So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[j, last(L_s)]$ is a prefix of $w[i, last(L_r)]$



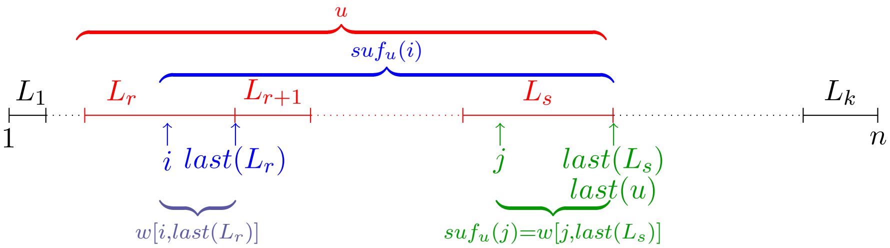
- Since $r < s$, $L_1 \geq \cdots \geq L_r \geq \cdots \geq L_s \geq \cdots \geq L_k$ and L_r is strictly less than any of its proper suffixes, it is **easy** to verify that the order relation between the local and the global suffixes is the same!
  So we don't need to compare further symbols.
- We need $l(j) = |w[j, last(L_s)]|$ symbol comparisons.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



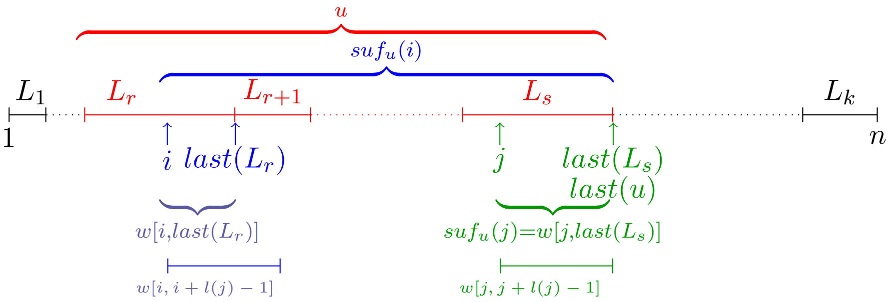
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i,j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



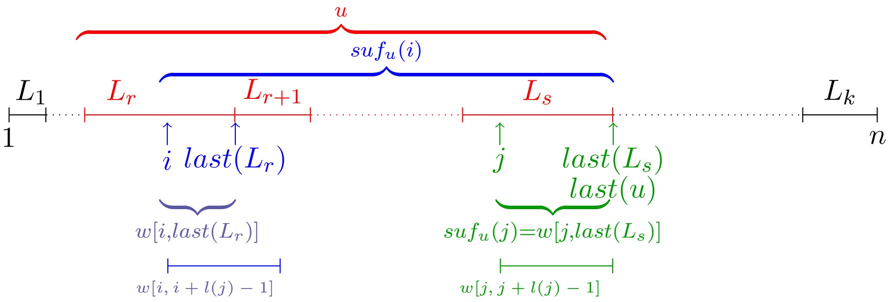
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch,

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



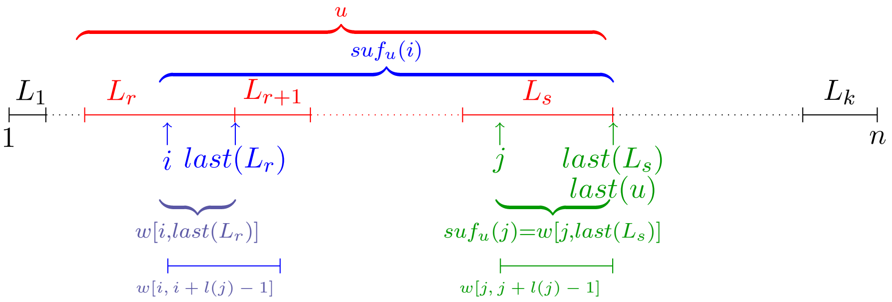
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of u and of w.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



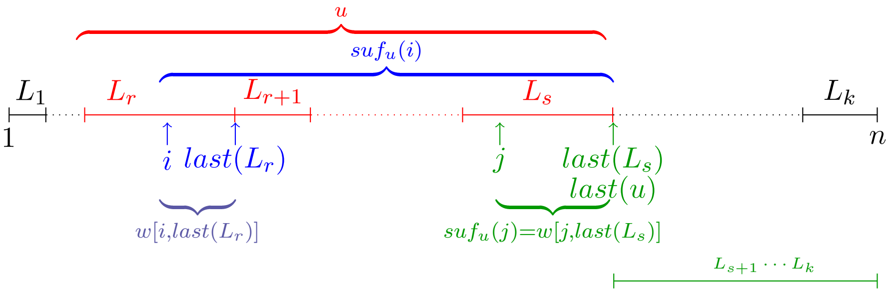
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of u and of w.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



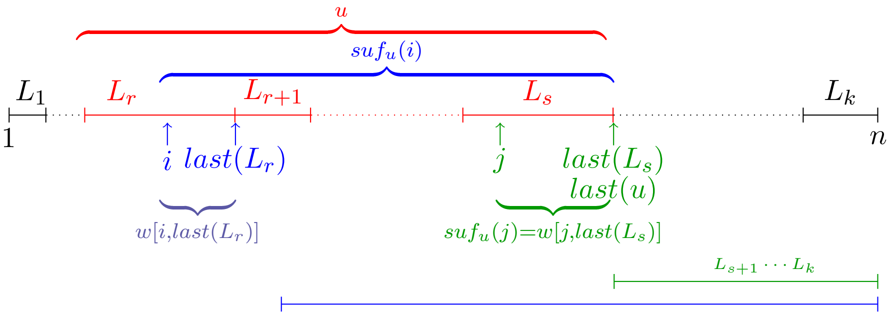
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of u and of w.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



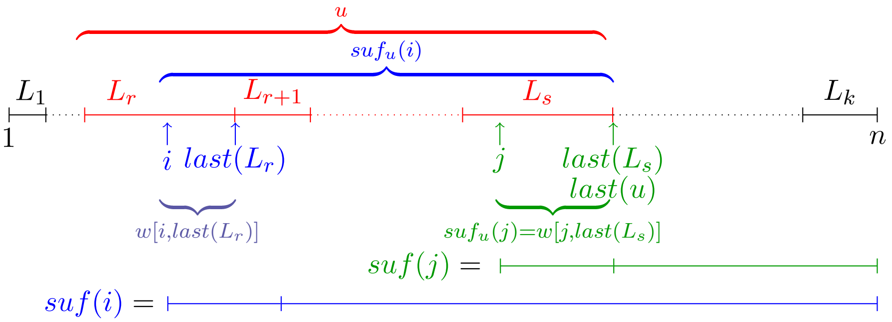
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of u and of w.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



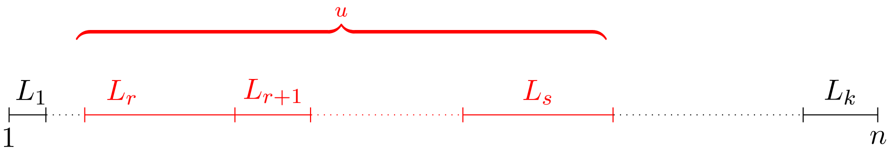
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $L_{s+1} \cdots L_k$ is smaller than any suffix of u and of w.

Second case: $w[i, last(L_r)]$ is a prefix of $w[j, last(L_s)]$



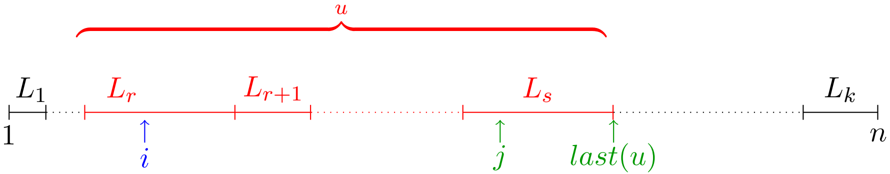
- In order to get the mutual order between $suf(i)$ and $suf(j)$, we need to compare at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Consider $w[i, i + l(j) - 1]$ and $w[j, j + l(j) - 1] = suf_u(j)$.
  - There is a mismatch, then we need $lcp(i, j) + 1 \leq l(j)$ symbol comparisons.
  - There is not a mismatch, then we use the property of the Lyndon factorization: $\underline{L_{s+1} \cdots L_k}$ is smaller than any suffix of u and of w.

How many symbol comparisons?



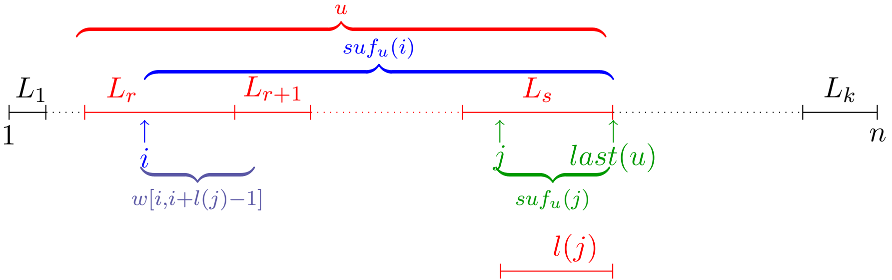
- In order to get the mutual order between $suf(i)$ and $suf(j)$ it is sufficient to execute at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Note that $l(j)$, as shown by the following example, can be smaller than $lcp(i, j) + 1$.

How many symbol comparisons?



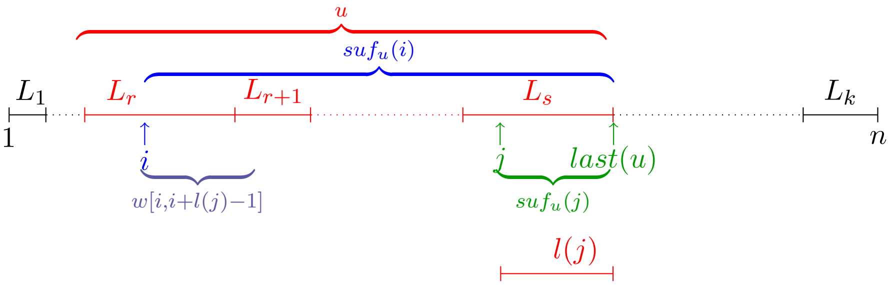
- In order to get the mutual order between $suf(i)$ and $suf(j)$ it is sufficient to execute at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Note that $l(j)$, as shown by the following example, can be smaller than $lcp(i, j) + 1$.

How many symbol comparisons?



- In order to get the mutual order between $suf(i)$ and $suf(j)$ it is sufficient to execute at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Note that $l(j)$, as shown by the following example, can be smaller than $lcp(i, j) + 1$.

How many symbol comparisons?



- In order to get the mutual order between $suf(i)$ and $suf(j)$ it is sufficient to execute at most $l(j) = |suf_u(j)|$ symbol comparisons.
- Note that $l(j)$, as shown by the following example, can be smaller than $lcp(i,j) + 1$.

Example

Let $w = abaaaabaaaaabaaaabaaaaaab$. Its Lyndon factorization is $ab|aaaab|aaaaabaaaab|aaaaaab$. Let $u = ab|aaaab|aaaaabaaaab|$.

$$
\begin{array}{ccccccccccccccccccccccccccc}
 & & i=2 & & & & & & & & & & & & j=13 & & & & & & & & & & & & \\
 & & \downarrow & & & & & & & & & & & & \downarrow & & & & & & & & & & & & \\
 & 1 & \mathbf{2} & & 3 & 4 & 5 & 6 & 7 & & 8 & 9 & 10 & 11 & 12 & \mathbf{13} & 14 & 15 & 16 & 17 & 18 & & 19 & 20 & 21 & 22 & 23 & 24 & 25 & \\
w = & a & \mathbf{b} & | & a & a & a & a & b & | & a & a & a & a & a & \mathbf{b} & a & a & a & a & b & | & a & a & a & a & a & a & b & |
\end{array}
$$

Consider the following suffixes:

$$
\begin{array}{rl}
 & \overset{2}{\downarrow} \\
suf(2) = & b\ a\ a\ a\ a\ b\ a\ a\ a\ a\ a\ b\ a\ a\ a\ a\ b\ a\ a\ a\ a\ a\ a\ b \\
suf(13) = & b\ a\ a\ a\ a\ b\ a\ a\ a\ a\ a\ a\ b \\
 & \underset{13}{\uparrow}
\end{array}
$$

We have $lcp(2, 13) = 11$ and $l(13) = 6$.

We need only 6 symbol comparisons, indeed for Lyndon properties $w[8, 25] > w[19, 25] \Rightarrow suf(2) > suf(13)$.

Example

Let $w = abaaaabaaaaabaaaabaaaaaab$. Its Lyndon factorization is $ab|aaaab|aaaaabaaaab|aaaaaab$. Let $u = ab|aaaab|aaaaabaaaab|$.

$$
\begin{array}{cccccccccccccccccccccccccccc}
 & i=2 & & & & & & & & & & & & j=13 & & & & & & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & \downarrow & & & & & & & & & & & & & \\
 1 & \mathbf{2} & & 3 & 4 & 5 & 6 & 7 & & 8 & 9 & 10 & 11 & 12 & \mathbf{13} & 14 & 15 & 16 & 17 & 18 & & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\
w = a & \mathbf{b} & | & a & a & a & a & b & | & a & a & a & a & a & \mathbf{b} & a & a & a & a & b & | & a & a & a & a & a & a & b \; | \\
 & & & & & & & & & & & & & \uparrow & & & & & & & & & & & & \uparrow & & \\
 & & & & & & & & & & & & & 2+lcp(2,13)-1 & & & & & & & & & & & & 13+lcp(2,13)-1 & &
\end{array}
$$

Consider the following suffixes:

$$
\begin{array}{rccccccccccccc}
 & 2 & & & & & & & & & & 2+lcp(2,13) & & \\
 & \downarrow & & & & & & & & & & \downarrow & & \\
suf(2) = & b & a & a & a & a & b & a & a & a & a & a & b & a \; a\,a\,a\,b\,a\,a\,a\,a\,a\,a\,b \\
suf(13) = & b & a & a & a & a & b & a & a & a & a & a & a & b \\
 & \uparrow & & & & & & & & & & \uparrow & & \\
 & 13 & & & & & & & & & & 13+lcp(2,13) & &
\end{array}
$$

We have $lcp(2, 13) = 11$ and $l(13) = 6$.

We need only 6 symbol comparisons, indeed for Lyndon properties $w[8, 25] > w[19, 25] \Rightarrow suf(2) > suf(13)$.

Example

Let $w = abaaaabaaaaabaaaabaaaaaab$. Its Lyndon factorization is $ab|aaaab|aaaaabaaaab|aaaaaab$. Let $u = ab|aaaab|aaaaabaaaab|$.

$$
\begin{array}{cccccccccccccccccccccccccccc}
 & i=2 & & & & & & & & & & & & j=13 & & & & & \mathit{last}(u)=18 & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & \downarrow & & & & & \downarrow & & & & & & & & \\
1 & \mathbf{2} & & 3 & 4 & 5 & 6 & 7 & & 8 & 9 & 10 & 11 & 12 & \mathbf{13} & 14 & 15 & 16 & 17 & 18 & & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\
w = a & \mathbf{b} & | & a & a & a & a & b & | & a & a & a & a & a & \mathbf{b} & a & a & a & a & b & | & a & a & a & a & a & a & b \; | \\
 & & & & & & & \uparrow & & & & & & & & & & & & \uparrow & & & & & & & & \\
 & & & & & & & 2+l(13)-1 & & & & & & & & & & & & 13+l(13)-1 & & & & & & & &
\end{array}
$$

Consider the following suffixes:

$$
\begin{array}{lcccccccccccc}
 & 2 & & & & & 2+l(13) & & & & & 2+lcp(2,13) & \\
 & \downarrow & & & & & \downarrow & & & & & \downarrow & \\
suf(2) = & b & a & a & a & a & b & a & a & a & a & a & b & a & a\,a\,a\,b\,a\,a\,a\,a\,a\,a\,b \\
suf(13) = & b & a & a & a & a & b & a & a & a & a & a & a & b & \\
 & \uparrow & & & & & \uparrow & & & & & \uparrow & \\
 & 13 & & & & & 13+l(13) & & & & & 13+lcp(2,13) &
\end{array}
$$

We have $lcp(2,13) = 11$ and $l(13) = 6$.

We need only 6 symbol comparisons, indeed for Lyndon properties $w[8,25] > w[19,25] \Rightarrow suf(2) > suf(13)$.

Example

Let $w = abaaaabaaaaabaaaabaaaaaab$. Its Lyndon factorization is $ab|aaaab|aaaaabaaaab|aaaaaab$. Let $u = ab|aaaab|aaaaabaaaab|$.

$$
\begin{array}{cccccccccccccccccccccccccccccc}
 & & i=2 & & & & & & & & & & & & & j=13 & & & & last(u)=18 & & & & & & & & & \\
 & & \downarrow & & & & & & & & & & & & & \downarrow & & & & \downarrow & & & & & & & & & \\
 & 1 & \mathbf{2} & & 3 & 4 & 5 & 6 & 7 & & 8 & 9 & 10 & 11 & 12 & \mathbf{13} & 14 & 15 & 16 & 17 & 18 & & 19 & 20 & 21 & 22 & 23 & 24 & 25 & \\
w = & a & \mathbf{b} & | & a & a & a & a & b & | & a & a & a & a & a & \mathbf{b} & a & a & a & a & b & | & a & a & a & a & a & a & b & | \\
 & & & & & & & & \uparrow & & & & & & & & & & & & \uparrow & & & & & & & & & \\
 & & & & & & & & 2+l(13)-1 & & & & & & & & & & & & 13+l(13)-1 & & & & & & & & &
\end{array}
$$

Consider the following suffixes:

$$
\begin{array}{rccccccccccc}
 & 2 & & & & & 2+l(13) & & & & 2+lcp(2,13) & \\
 & \downarrow & & & & & \downarrow & & & & \downarrow & \\
suf(2) = & b & a & a & a & a & b & a & a\ a\ a\ a & & b & a \quad a\ a\ a\ b\ a\ a\ a\ a\ a\ a\ b \\
suf(13) = & b & a & a & a & a & b & a & a\ a\ a\ a & & a & b \\
 & \uparrow & & & & & \uparrow & & & & \uparrow & \\
 & 13 & & & & & 13+l(13) & & & & 13+lcp(2,13) &
\end{array}
$$

We have $lcp(2,13) = 11$ and $l(13) = 6$.

We need only 6 symbol comparisons, indeed for Lyndon properties $w[8,25] > w[19,25] \Rightarrow suf(2) > suf(13)$.

Our strategy for sorting all suffixes

Let $w = L_1 L_2 \cdots L_l L_{l+1} \cdots L_k$. We propose an algorithm that is based on the following

Proposition

Let $sort(L_1 L_2 \cdots L_l)$ and $sort(L_{l+1} L_{l+2} \cdots L_k)$ denote the sorted lists of the suffixes of $L_1 L_2 \cdots L_l$ and the suffixes $L_{l+1} L_{l+2} \cdots L_k$, respectively. Then
$sort(L_1 L_2 \cdots L_k) = merge(sort(L_1 L_2 \cdots L_l), sort(L_{l+1} L_{l+2} \cdots L_k))$.

- The sorted list of the global suffixes of w can be obtained by merging the sorted lists of the local suffixes inside $L_1 L_2 \cdots L_l$ and $L_{l+1} L_{l+2} \cdots L_k$.
- Note that the mutual order of the local suffixes is preserved after the merge operation.

Our strategy for sorting all suffixes

Let $w = L_1 L_2 \cdots L_l L_{l+1} \cdots L_k$. We propose an algorithm that is based on the following

Proposition

Let $sort(L_1 L_2 \cdots L_l)$ and $sort(L_{l+1} L_{l+2} \cdots L_k)$ denote the sorted lists of the suffixes of $L_1 L_2 \cdots L_l$ and the suffixes $L_{l+1} L_{l+2} \cdots L_k$, respectively. Then
$sort(L_1 L_2 \cdots L_k) = merge(sort(L_1 L_2 \cdots L_l), sort(L_{l+1} L_{l+2} \cdots L_k))$.

- The sorted list of the global suffixes of w can be obtained by merging the sorted lists of the local suffixes inside $L_1 L_2 \cdots L_l$ and $L_{l+1} L_{l+2} \cdots L_k$.
- Note that the mutual order of the local suffixes is preserved after the merge operation.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

Our algorithm

This proposition suggests a possible strategy for sorting the list of the suffixes of some word w:

- find the Lyndon decomposition of w: $L_1 L_2 \cdots L_k$;
- find the sorted list of the suffixes of L_1 and, separately, the sorted list of the suffixes of L_2;
- merge the sorted lists in order to obtain the sorted list of the suffixes of $L_1 L_2$;
- find the sorted list of the suffixes of L_3 and merge it to the previous sorted list;
- repeat until all the Lyndon factors are processed;

One can use this strategy for computing the suffix array and for constructing the Burrows-Wheeler Transform.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol $\$$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append $\$$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$$

$$M$$

$$\begin{array}{rcccccccccccc}
1 & \$ & m & a & t & h & e & m & a & t & i & c & s \\
2 & a & t & h & e & m & a & t & i & c & s & \$ & m \\
3 & a & t & i & c & s & \$ & m & a & t & h & e & m \\
4 & c & s & \$ & m & a & t & h & e & m & a & t & i \\
5 & e & m & a & t & i & c & s & \$ & m & a & t & h \\
6 & h & e & m & a & t & i & c & s & \$ & m & a & t \\
7 & i & c & s & \$ & m & a & t & h & e & m & a & t \\
8 & m & a & t & h & e & m & a & t & i & c & s & \$ \\
9 & m & a & t & i & c & s & \$ & m & a & t & h & e \\
10 & s & \$ & m & a & t & h & e & m & a & t & i & c \\
11 & t & h & e & m & a & t & i & c & s & \$ & m & a \\
12 & t & i & c & s & \$ & m & a & t & h & e & m & a
\end{array}$$

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol $\$$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol $\$$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append $\$$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$

M

1	$\$$	m	a	t	h	e	m	a	t	i	c	s
2	a	t	h	e	m	a	t	i	c	s	$\$$	m
3	a	t	i	c	s	$\$$	m	a	t	h	e	m
4	c	s	$\$$	m	a	t	h	e	m	a	t	i
5	e	m	a	t	i	c	s	$\$$	m	a	t	h
6	h	e	m	a	t	i	c	s	$\$$	m	a	t
7	i	c	s	$\$$	m	a	t	h	e	m	a	t
8	m	a	t	h	e	m	a	t	i	c	s	$\$$
9	m	a	t	i	c	s	$\$$	m	a	t	h	e
10	s	$\$$	m	a	t	h	e	m	a	t	i	c
11	t	h	e	m	a	t	i	c	s	$\$$	m	a
12	t	i	c	s	$\$$	m	a	t	h	e	m	a

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol $\$$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol \$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append \$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

m	a	t	h	e	m	a	t	i	c	s	\$
a	t	h	e	m	a	t	i	c	s	\$	m
t	h	e	m	a	t	i	c	s	\$	m	a
h	e	m	a	t	i	c	s	\$	m	a	t
e	m	a	t	i	c	s	\$	m	a	t	h
m	a	t	i	c	s	\$	m	a	t	h	e
a	t	i	c	s	\$	m	a	t	h	e	m
t	i	c	s	\$	m	a	t	h	e	m	a
i	c	s	\$	m	a	t	h	e	m	a	t
c	s	\$	m	a	t	h	e	m	a	t	i
s	\$	m	a	t	h	e	m	a	t	i	c
\$	m	a	t	h	e	m	a	t	i	c	s

M

1	\$	m	a	t	h	e	m	a	t	i	c	s
2	a	t	h	e	m	a	t	i	c	s	\$	m
3	a	t	i	c	s	\$	m	a	t	h	e	m
4	c	s	\$	m	a	t	h	e	m	a	t	i
5	e	m	a	t	i	c	s	\$	m	a	t	h
6	h	e	m	a	t	i	c	s	\$	m	a	t
7	i	c	s	\$	m	a	t	h	e	m	a	t
8	m	a	t	h	e	m	a	t	i	c	s	\$
9	m	a	t	i	c	s	\$	m	a	t	h	e
10	s	\$	m	a	t	h	e	m	a	t	i	c
11	t	h	e	m	a	t	i	c	s	\$	m	a
12	t	i	c	s	\$	m	a	t	h	e	m	a

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol \$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol \$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append \$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$$
\begin{array}{cccccccccccc}
m & a & t & h & e & m & a & t & i & c & s & \$ \\
a & t & h & e & m & a & t & i & c & s & \$ & m \\
t & h & e & m & a & t & i & c & s & \$ & m & a \\
h & e & m & a & t & i & c & s & \$ & m & a & t \\
e & m & a & t & i & c & s & \$ & m & a & t & h \\
m & a & t & i & c & s & \$ & m & a & t & h & e \\
a & t & i & c & s & \$ & m & a & t & h & e & m \\
t & i & c & s & \$ & m & a & t & h & e & m & a \\
i & c & s & \$ & m & a & t & h & e & m & a & t \\
c & s & \$ & m & a & t & h & e & m & a & t & i \\
s & \$ & m & a & t & h & e & m & a & t & i & c \\
\$ & m & a & t & h & e & m & a & t & i & c & s
\end{array}
\Rightarrow
\begin{array}{c|cccccccccccc}
 & & & & & & & M & & & & & \\
1 & \$ & m & a & t & h & e & m & a & t & i & c & s \\
2 & a & t & h & e & m & a & t & i & c & s & \$ & m \\
3 & a & t & i & c & s & \$ & m & a & t & h & e & m \\
4 & c & s & \$ & m & a & t & h & e & m & a & t & i \\
5 & e & m & a & t & i & c & s & \$ & m & a & t & h \\
6 & h & e & m & a & t & i & c & s & \$ & m & a & t \\
7 & i & c & s & \$ & m & a & t & h & e & m & a & t \\
8 & m & a & t & h & e & m & a & t & i & c & s & \$ \\
9 & m & a & t & i & c & s & \$ & m & a & t & h & e \\
10 & s & \$ & m & a & t & h & e & m & a & t & i & c \\
11 & t & h & e & m & a & t & i & c & s & \$ & m & a \\
12 & t & i & c & s & \$ & m & a & t & h & e & m & a
\end{array}
$$

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol \$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol \$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append \$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$$
\begin{array}{cccccccccccc}
m & a & t & h & e & m & a & t & i & c & s & \$ \\
a & t & h & e & m & a & t & i & c & s & \$ & m \\
t & h & e & m & a & t & i & c & s & \$ & m & a \\
h & e & m & a & t & i & c & s & \$ & m & a & t \\
e & m & a & t & i & c & s & \$ & m & a & t & h \\
m & a & t & i & c & s & \$ & m & a & t & h & e \\
a & t & i & c & s & \$ & m & a & t & h & e & m \\
t & i & c & s & \$ & m & a & t & h & e & m & a \\
i & c & s & \$ & m & a & t & h & e & m & a & t \\
c & s & \$ & m & a & t & h & e & m & a & t & i \\
s & \$ & m & a & t & h & e & m & a & t & i & c \\
\$ & m & a & t & h & e & m & a & t & i & c & s
\end{array}
\Rightarrow
\begin{array}{c|cccccccccccc}
 & F & & & & & M & & & & & & L \\
 & \downarrow & & & & & & & & & & & \downarrow \\
1 & \$ & m & a & t & h & e & m & a & t & i & c & s \\
2 & a & t & h & e & m & a & t & i & c & s & \$ & m \\
3 & a & t & i & c & s & \$ & m & a & t & h & e & m \\
4 & c & s & \$ & m & a & t & h & e & m & a & t & i \\
5 & e & m & a & t & i & c & s & \$ & m & a & t & h \\
6 & h & e & m & a & t & i & c & s & \$ & m & a & t \\
7 & i & c & s & \$ & m & a & t & h & e & m & a & t \\
8 & m & a & t & h & e & m & a & t & i & c & s & \$ \\
9 & m & a & t & i & c & s & \$ & m & a & t & h & e \\
10 & s & \$ & m & a & t & h & e & m & a & t & i & c \\
11 & t & h & e & m & a & t & i & c & s & \$ & m & a \\
12 & t & i & c & s & \$ & m & a & t & h & e & m & a
\end{array}
$$

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol \$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol $\$$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append $\$$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$$
\begin{array}{cccccccccccc}
m & a & t & h & e & m & a & t & i & c & s & \$ \\
a & t & h & e & m & a & t & i & c & s & \$ & m \\
t & h & e & m & a & t & i & c & s & \$ & m & a \\
h & e & m & a & t & i & c & s & \$ & m & a & t \\
e & m & a & t & i & c & s & \$ & m & a & t & h \\
m & a & t & i & c & s & \$ & m & a & t & h & e \\
a & t & i & c & s & \$ & m & a & t & h & e & m \\
t & i & c & s & \$ & m & a & t & h & e & m & a \\
i & c & s & \$ & m & a & t & h & e & m & a & t \\
c & s & \$ & m & a & t & h & e & m & a & t & i \\
s & \$ & m & a & t & h & e & m & a & t & i & c \\
\$ & m & a & t & h & e & m & a & t & i & c & s
\end{array}
\Rightarrow
\begin{array}{rcccccccccccc}
 & F & & & & & M & & & & & & L \\
 & \downarrow & & & & & & & & & & & \downarrow \\
1 & \$ & m & a & t & h & e & m & a & t & i & c & s \\
2 & a & t & h & e & m & a & t & i & c & s & \$ & m \\
3 & a & t & i & c & s & \$ & m & a & t & h & e & m \\
4 & c & s & \$ & m & a & t & h & e & m & a & t & i \\
5 & e & m & a & t & i & c & s & \$ & m & a & t & h \\
6 & h & e & m & a & t & i & c & s & \$ & m & a & t \\
7 & i & c & s & \$ & m & a & t & h & e & m & a & t \\
\rightarrow 8 & m & a & t & h & e & m & a & t & i & c & s & \$ \\
9 & m & a & t & i & c & s & \$ & m & a & t & h & e \\
10 & s & \$ & m & a & t & h & e & m & a & t & i & c \\
11 & t & h & e & m & a & t & i & c & s & \$ & m & a \\
12 & t & i & c & s & \$ & m & a & t & h & e & m & a
\end{array}
$$

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol $\$$ in L.

The Burrows-Wheeler Transform

A possible definition of BWT consists in adding an end-marker symbol \$ at the end of the word. Start with word $w \in \Sigma^*$.

- Append \$ symbol, which is lexicographically before all other characters in the alphabet Σ.
- Generate all of the conjugates of $w\$$ and sort them lexicographically, forming a matrix M with rows and columns equal to $|w\$| = |w| + 1$.
- Construct L, the transformed text of $w\$$, by taking the last column of M.

Example: $w = mathematics$

$$
\begin{array}{cccccccccccc}
m & a & t & h & e & m & a & t & i & c & s & \$ \\
a & t & h & e & m & a & t & i & c & s & \$ & m \\
t & h & e & m & a & t & i & c & s & \$ & m & a \\
h & e & m & a & t & i & c & s & \$ & m & a & t \\
e & m & a & t & i & c & s & \$ & m & a & t & h \\
m & a & t & i & c & s & \$ & m & a & t & h & e \\
a & t & i & c & s & \$ & m & a & t & h & e & m \\
t & i & c & s & \$ & m & a & t & h & e & m & a \\
i & c & s & \$ & m & a & t & h & e & m & a & t \\
c & s & \$ & m & a & t & h & e & m & a & t & i \\
s & \$ & m & a & t & h & e & m & a & t & i & c \\
\$ & m & a & t & h & e & m & a & t & i & c & s
\end{array}
\Rightarrow
\begin{array}{rcccccccccccc}
 & F & & & & & M & & & & & & L \\
 & \downarrow & & & & & & & & & & & \downarrow \\
1 & \$ & m & a & t & h & e & m & a & t & i & c & s \\
2 & a & t & h & e & m & a & t & i & c & s & \$ & m \\
3 & a & t & i & c & s & \$ & m & a & t & h & e & m \\
4 & c & s & \$ & m & a & t & h & e & m & a & t & i \\
5 & e & m & a & t & i & c & s & \$ & m & a & t & h \\
6 & h & e & m & a & t & i & c & s & \$ & m & a & t \\
7 & i & c & s & \$ & m & a & t & h & e & m & a & t \\
\rightarrow 8 & m & a & t & h & e & m & a & t & i & c & s & \$ \\
9 & m & a & t & i & c & s & \$ & m & a & t & h & e \\
10 & s & \$ & m & a & t & h & e & m & a & t & i & c \\
11 & t & h & e & m & a & t & i & c & s & \$ & m & a \\
12 & t & i & c & s & \$ & m & a & t & h & e & m & a
\end{array}
$$

Output: $bwt(w\$) = L = smmihtt\$ecaa$.

To recover the original word, it is enough to know the position of the symbol \$ in L.

BWT and SA

This is equivalent to sort the suffixes of $w\$$.

$$\begin{array}{ccccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ w = & m & a & t & h & e & m & a & t & i & c & s & \$ \end{array}$$

$$\begin{array}{cccccccccccc} & & & & M & & & & & & & L \\ & & & & & & & & & & & \downarrow \\ \$ & m & a & t & h & e & m & a & t & i & c & s \\ a & t & h & e & m & a & t & i & c & s & \$ & m \\ a & t & i & c & s & \$ & m & a & t & h & e & m \\ c & s & \$ & m & a & t & h & e & m & a & t & i \\ e & m & a & t & i & c & s & \$ & m & a & t & h \\ h & e & m & a & t & i & c & s & \$ & m & a & t \\ i & c & s & \$ & m & a & t & h & e & m & a & t \\ m & a & t & h & e & m & a & t & i & c & s & \$ \\ m & a & t & i & c & s & \$ & m & a & t & h & e \\ s & \$ & m & a & t & h & e & m & a & t & i & c \\ t & h & e & m & a & t & i & c & s & \$ & m & a \\ t & i & c & s & \$ & m & a & t & h & e & m & a \end{array} \Leftrightarrow \begin{array}{cl} BWT & \text{Sorted Suffixes} \\ s & \$ \\ m & athematics\$ \\ m & atics\$ \\ i & cs\$ \\ h & ematics\$ \\ t & hematics\$ \\ t & ics\$ \\ \$ & mathematics\$ \\ e & matics\$ \\ c & s\$ \\ a & thematics\$ \\ a & tics\$ \end{array}$$

Note that one can build the BWT of a string without needing to compute its suffix array.

BWT and SA

This is equivalent to sort the suffixes of $w\$$.
In order to obtain it, one can compute the suffix array.

- $SA[i]$: The starting position of the ith smallest suffix of $w\$$.
- $BWT[i]$: The symbol that (circularly) precedes the first symbol of the ith smallest suffix.

$$\begin{array}{cccccccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ w & = & m & a & t & h & e & m & a & t & i & c & s & \$ \end{array}$$

M											L ↓
$\$$	m	a	t	h	e	m	a	t	i	c	s
a	t	h	e	m	a	t	i	c	s	$\$$	m
a	t	i	c	s	$\$$	m	a	t	h	e	m
c	s	$\$$	m	a	t	h	e	m	a	t	i
e	m	a	t	i	c	s	$\$$	m	a	t	h
h	e	m	a	t	i	c	s	$\$$	m	a	t
i	c	s	$\$$	m	a	t	h	e	m	a	t
m	a	t	h	e	m	a	t	i	c	s	$\$$
m	a	t	i	c	s	$\$$	m	a	t	h	e
s	$\$$	m	a	t	h	e	m	a	t	i	c
t	h	e	m	a	t	i	c	s	$\$$	m	a
t	i	c	s	$\$$	m	a	t	h	e	m	a

$\Leftrightarrow$

SA	BWT	Sorted Suffixes
12	s	$\$$
2	m	$a\,t\,h\,e\,m\,a\,t\,i\,c\,s\,\$$
7	m	$a\,t\,i\,c\,s\,\$$
9	i	$c\,s\,\$$
5	h	$e\,m\,a\,t\,i\,c\,s\,\$$
4	t	$h\,e\,m\,a\,t\,i\,c\,s\,\$$
9	t	$i\,c\,s\,\$$
1	$\$$	$m\,a\,t\,h\,e\,m\,a\,t\,i\,c\,s\,\$$
6	e	$m\,a\,t\,i\,c\,s\,\$$
11	c	$s\,\$$
3	a	$t\,h\,e\,m\,a\,t\,i\,c\,s\,\$$
8	a	$t\,i\,c\,s\,\$$

Note that one can build the BWT of a string without needing to compute its suffix array.

Compute the BWT and the SA

- the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
- the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

Let $w = aabcabbaabaabdabbaaabbdc$. Its Lyndon factorization is $aabcabb|aabaabdabb|aaabbdc$.

$w\$ = $ $L_1 = aabcabb$ $L_2 = aabaabdabb$ $L_3 = aaabbdc$ $L_4 = \$$

Compute the BWT and the SA

- the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
- the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

Let $w = aabcabbaabaabdabbaaabbdc$. Its Lyndon factorization is $aabcabb|aabaabdabb|aaabbdc$.

$$w\$ = \underbrace{\vdash\!\!\text{———}\!\!\dashv}^{L_1 = aabcabb}\underbrace{\vdash\!\!\text{———}\!\!\dashv}^{L_2 = aabaabdabb}\underbrace{\vdash\!\!\text{———}\!\!\dashv}^{L_3 = aaabbdc}\underbrace{\vdash\!\!\text{—}\!\!\dashv}^{L_4 = \$}$$

Compute the BWT and the SA

- the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
- the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

Let $w = aabcabbaabaabdabbaaabbdc$. Its Lyndon factorization is $aabcabb|aabaabdabb|aaabbdc$.

$$w\$ = \overbrace{aabcabb}^{L_1} \; \overbrace{aabaabdabb}^{L_2} \; \overbrace{aaabbdc}^{L_3} \; \overbrace{\$}^{L_4}$$

$L_1 = aabcabb$, $L_2 = aabaabdabb$, $L_3 = aaabbdc$, $L_4 = \$$

Consider: $L_1\$ = aabcabb\$$

Compute the BWT and the SA

- the Suffix Array (SA): the array containing the starting positions of the suffixes of a word, sorted in lexicographic order;
- the Burrows-Wheeler Transform (BWT): the array containing a permutation of the symbols of a word according to the sorting of its suffixes.

Let $w = aabcabbaabaabdabbaaabbdc$. Its Lyndon factorization is $aabcabb|aabaabdabb|aaabbdc$.

$$w\$ = \underbrace{\vphantom{|}}_{L_1 = aabcabb}\underbrace{\vphantom{|}}_{L_2 = aabaabdabb}\underbrace{\vphantom{|}}_{L_3 = aaabbdc}\underbrace{\vphantom{|}}_{L_4 = \$}$$

Consider: $L_1\$ = aabcabb\$$

Compute the $BWT(L_1\$)$ and $SA(L_1\$)$:

$L_1\$$

SA	BWT	Sorted Suffixes
8	$\underline{b}$	$\$$
1	$\$$	$aabcabb\$$
5	c	$abb\$$
2	a	$abcabb\$$
7	b	$b\$$
6	a	$bb\$$
3	a	$bcabb\$$
4	b	$cabb\$$

Compute the BWT and the SA

$$w = \underbrace{L_1 = aabcabb}\ \underbrace{L_2 = aabaabdabb}\ \underbrace{L_3 = aaabbdc}\ \underbrace{L_4 = \$}$$

$$L_1\$ = aabcabb\$$$

$L_1\$$

SA	BWT	Sorted Suffixes
8	$\underline{b}$	$\$$
1	$\$$	$aabcabb\$$
5	c	$abb\$$
2	a	$abcabb\$$
7	b	$b\$$
6	a	$bb\$$
3	a	$bcabb\$$
4	b	$cabb\$$

Compute the BWT and the SA

$$w = \underbrace{L_1 = aabcabb}\ \underbrace{L_2 = aabaabdabb}\ \underbrace{L_3 = aaabbdc}\ \underbrace{L_4 = \$}$$

$L_1\$ = aabcabb\$$

Consider: $L_2\$ = aabaabdabb\$$

Note that $|L_1| = j_1 = 7$. Compute the $BWT(L_2\$)$ and $SA(L_2\$)$.

$L_1\$$

SA	BWT	Sorted Suffixes
8	$\underline{b}$	$\$$
1	$\$$	$aabcabb\$$
5	c	$abb\$$
2	a	$abcabb\$$
7	b	$b\$$
6	a	$bb\$$
3	a	$bcabb\$$
4	b	$cabb\$$

$L_2\$$

SA	BWT	Sorted Suffixes
$11+7=18$	b	$\$$
$1+7=8$	$\$$	$\underline{aabaabdabb\$}$
$4+7=11$	b	$aabdabb\$$
$2+7=9$	a	$abaabdabb\$$
$8+7=15$	d	$abb\$$
$5+7=12$	a	$abdabb\$$
$10+7=17$	b	$b\$$
$3+7=10$	a	$baabdabb\$$
$9+7=16$	a	$bb\$$
$6+7=13$	a	$bdabb\$$
$7+7=14$	b	$dabb\$$

Compute the BWT and the SA

$L_1\$$

	SA	BWT	Sorted Suffixes
	8	$\underline{b}$	$\$$
	1	$\$$	$aabcabb\$$
	5	c	$abb\$$
	2	a	$abcabb\$$
	7	b	$b\$$
	6	a	$bb\$$
	3	a	$bcabb\$$
	4	b	$cabb\$$

$L_2\$$

G	SA	BWT	Sorted Suffixes
0	$11+7=18$	b	$\$$
0	$1+7=8$	$\$$	$\underline{aabaabdabb\$}$
2	$4+7=11$	b	$aabdabb\$$
2	$2+7=9$	a	$abaabdabb\$$
2	$8+7=15$	d	$abb\$$
4	$5+7=12$	a	$abdabb\$$
4	$10+7=17$	b	$b\$$
5	$3+7=10$	a	$baabdabb\$$
5	$9+7=16$	a	$bb\$$
7	$6+7=13$	a	$bdabb\$$
8	$7+7=14$	b	$dabb\$$

merge $\Rightarrow$

$L_1L_2\$$

SA	BWT	Sorted Suffixes
18	b	$\$$
$\underline{8}$	$\underline{b}$	$\underline{aabaabdabb\$}$
1	$\$$	$aabcabbaabaabdabb\$$
11	b	$aabdabb\$$
9	a	$abaabdabb\$$
15	d	$abb\$$
5	c	$abbaabaabdabb\$$
2	a	$abcabbaabaabdabb\$$
12	a	$abdabb\$$
17	b	$b\$$
7	b	$baabaabdabb\$$
10	a	$baabdabb\$$
16	a	$bb\$$
6	a	$bbaabaabdabb\$$
3	a	$bcabbaabaabdabb\$$
13	a	$bdabb\$$
4	b	$cabbaabaabdabb\$$
14	b	$dabb\$$

Compute the BWT and the SA

$w = \underbrace{}_{}$ $L_1 = aabcabb$ $\color{red}{L_2 = aabaabdabb}$ $\color{blue}{L_3 = aaabbdc}$ $L_4 = \$$

$L_1L_2\$ = aabcabbaabaabdabb\$$

By merging the sorted list of the suffixes of $L_1L_2\$$ and of $L_3\$$, we obtain the SA/BWT of $w\$ = L_1L_2L_3\$$.

Compute the BWT and the SA

$$w = \underbrace{}_{L_1 = aabcabb}\underbrace{}_{L_2 = aabaabdabb}\underbrace{}_{L_3 = aaabbdc}\underbrace{}_{L_4 = \$}$$

$$L_1L_2\$ = aabcabbaabaabdabb\$$$

Consider: $L_3\$ = aaabbdc\$$

Compute the $BWT(L_3\$)$ and $SA(L_3\$)$.

SA	$L_3\$$ BWT	Sorted Suffixes
$17 + 8 = 25$	c	$\$$
$17 + 1 = 18$	$\$$	$aaabbdc\$$
$17 + 2 = 19$	a	$aabbdc\$$
$17 + 3 = 20$	a	$abbdc\$$
$17 + 4 = 21$	a	$bbdc\$$
$17 + 5 = 22$	b	$bdc\$$
$17 + 7 = 24$	d	$c\$$
$17 + 6 = 23$	b	$dc\$$

By merging the sorted list of the suffixes of $L_1L_2\$$ and of $L_3\$$, we obtain the SA/BWT of $w\$ = L_1L_2L_3\$$.

Compute the BWT and the SA

$w = \underbrace{}_{L_1 = aabcabb} \underbrace{}_{L_2 = aabaabdabb} \underbrace{}_{L_3 = aaabbdc} \underbrace{}_{L_4 = \$}$

$L_1 L_2\$ = aabcabbaabaabdabb\$$

Consider: $L_3\$ = aaabbdc\$$

Compute the $BWT(L_3\$)$ and $SA(L_3\$)$.

SA	$L_3\$$ BWT	Sorted Suffixes
$17+8=25$	c	$\$$
$17+1=18$	$\$$	$aaabbdc\$$
$17+2=19$	a	$aabbdc\$$
$17+3=20$	a	$abbdc\$$
$17+4=21$	a	$bbdc\$$
$17+5=22$	b	$bdc\$$
$17+7=24$	d	$c\$$
$17+6=23$	b	$dc\$$

By merging the sorted list of the suffixes of $L_1 L_2\$$ and of $L_3\$$, we obtain the SA/BWT of $w\$ = L_1 L_2 L_3\$$.

Further work: Parallel sorting

- The word could be partitioned into several sequences of consecutive blocks of Lyndon words, and the sorting algorithm can be applied *in parallel* to each of those sequences. Then one should merge the sorted lists.
- Furthermore, also the Lyndon factorization can be performed in parallel, as shown in [*Apostolico and Crochemore*, 1989] and [*Daykin, Iliopoulos and Smyth*, 1994].

Further work

One can compute the BWT without the SA by using our strategy and the strategies already used in the following papers:

- *Hon, Lam, Sadakane, Sung and Yiu*, 2007;
- *Ferragina, Gagie and Manzini*, 2010 and 2012;
- *Bauer, Cox and R.*, 2011 and 2013;
- *Crochemore, Grossi, Kärkkäinen and Landau*, 2013.

In this way, one could obtain algorithms that work:

- in external memory;
- in place.

One could use efficient dynamic data structures for the rank and insert operations, for instance by using Navarro and Nekrich's recent results on optimal representations of dynamic sequences.

Further work: linear algorithm

Does there exist a linear algorithm that uses the Lyndon Factorization in order to sort (implicity or explicity) the suffixes?

Open problem!

Thank you for your attention!