

Constructing the Bijective BWT

Hideo Bannai (Kyushu University),

Juha Kärkkäinen (Helsinki Institute of Information Technology),

Dominik Köppl (Kyushu University),

Marcin Piątkowski (Nicolaus Copernicus University)

the BBWT is
the BWT of
the Lyndon factorization
of an input text
with respect to \prec_{ω}

the BBWT is
the BWT of

the Lyndon factorization

1.

of an input text

with respect to

\prec_{ω}

2.

Lyndon words

- a
- aabab

Lyndon word is smaller than

- any proper suffix
- any rotation

Lyndon words

- a
- aabab

Lyndon word is smaller than

- any proper suffix
- any rotation

not Lyndon words:

- abaab (rotation aabab smaller)
- abab (abab not smaller than suffix ab)

Lyndon factorization [Chen+ '58]

- input: text $T = \boxed{T_1} \boxed{T_2} \dots \boxed{T_t}$
- output: factorization $T_1 \dots T_t$ with
 - T_x is Lyndon word
 - $T_x \geq_{\text{lex}} T_{x+1}$
 - factorization uniquely defined
 - linear time [Duval'88]

(Chen-Fox-Lyndon Theorem)

example

$T = \text{senescence}$

Lyndon factorization: $s \mid \text{enes} \mid \text{cen} \mid \text{ce}$

- $s, \text{enes}, \text{cen},$ and ce are Lyndon

- $s >_{\text{lex}} \text{enes} >_{\text{lex}} \text{cen} \geq_{\text{lex}} \text{ce}$

\prec_{ω} order

- $u \prec_{\omega} w \iff uuuu\dots \prec_{\text{lex}} wwww\dots$
- $ab \prec_{\text{lex}} aba$
- $aba \prec_{\omega} ab$

\prec_{ω} order

• $u \prec_{\omega} w \iff uuuu\dots \prec_{\text{lex}} www\dots$

• $ab \prec_{\text{lex}} aba$

ab**a**babab...

• $aba \prec_{\omega} ab$

aba**a**baaba...

bijjective BWT of senescence

s | enes | cen | ce

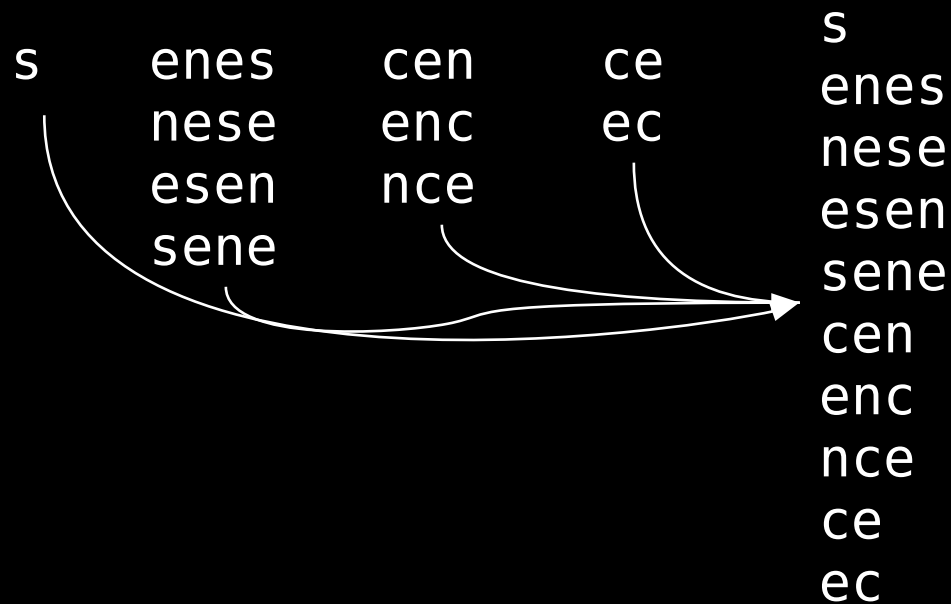
bijective BWT of senescence

s | enes | cen | ce

s	enes	cen	ce
	nese	enc	ec
	esen	nce	
	sene		

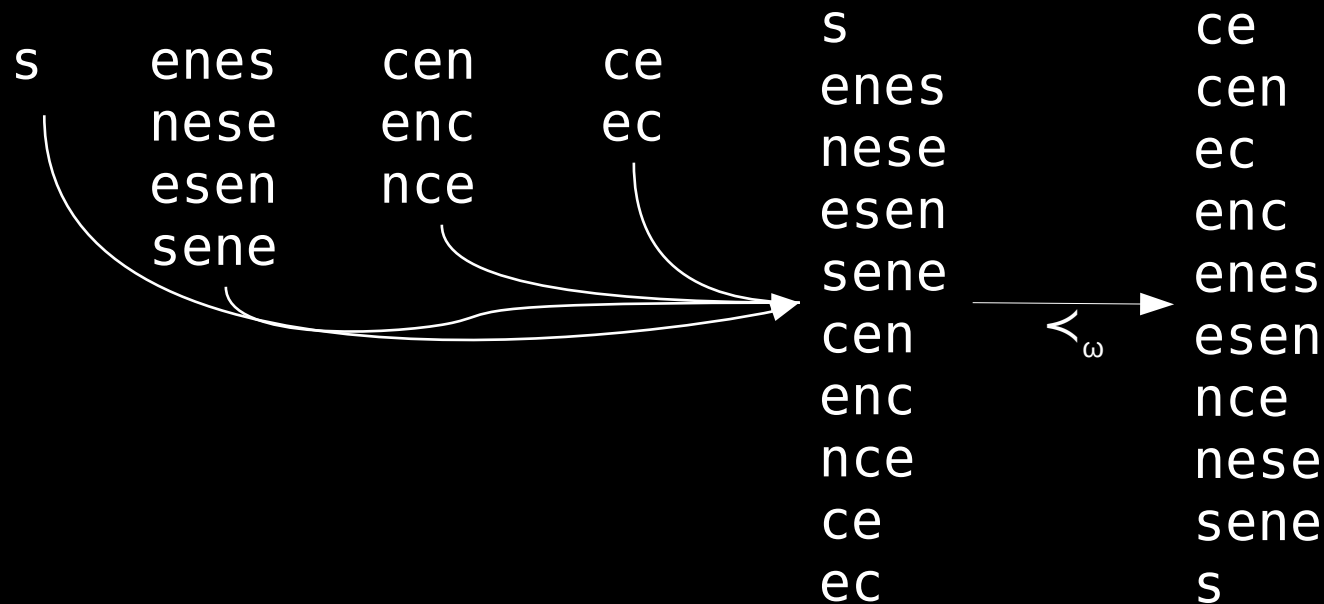
bijective BWT of senescence

s | enes | cen | ce



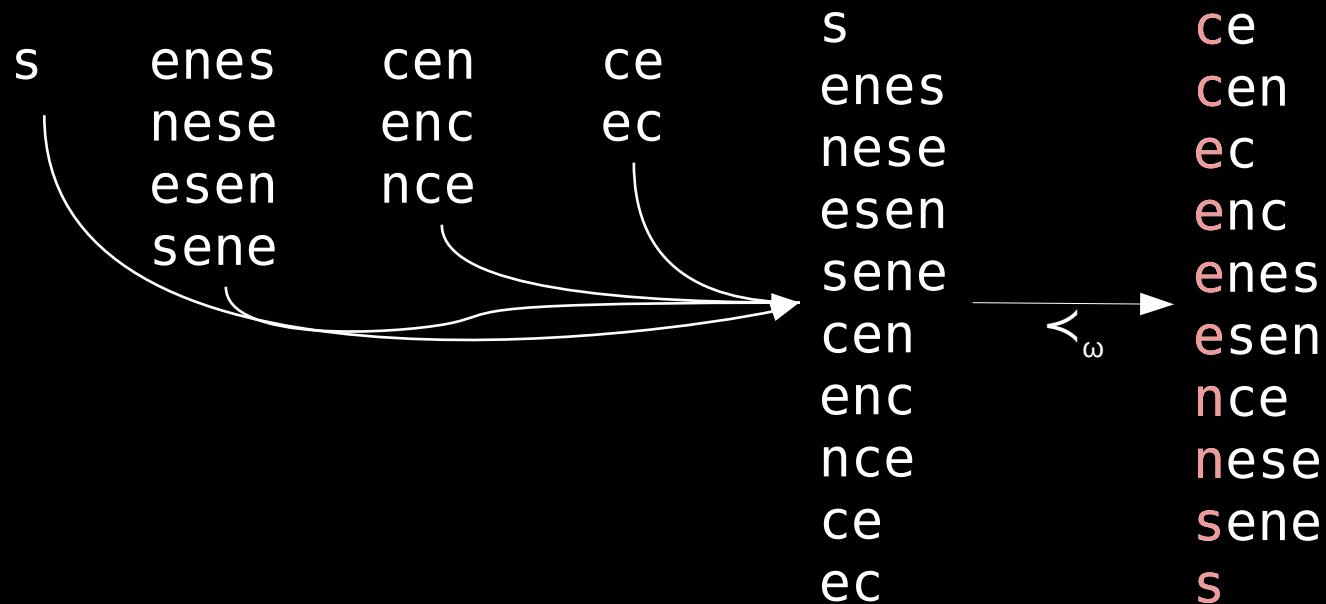
bijjective BWT of senescence

s | enes | cen | ce



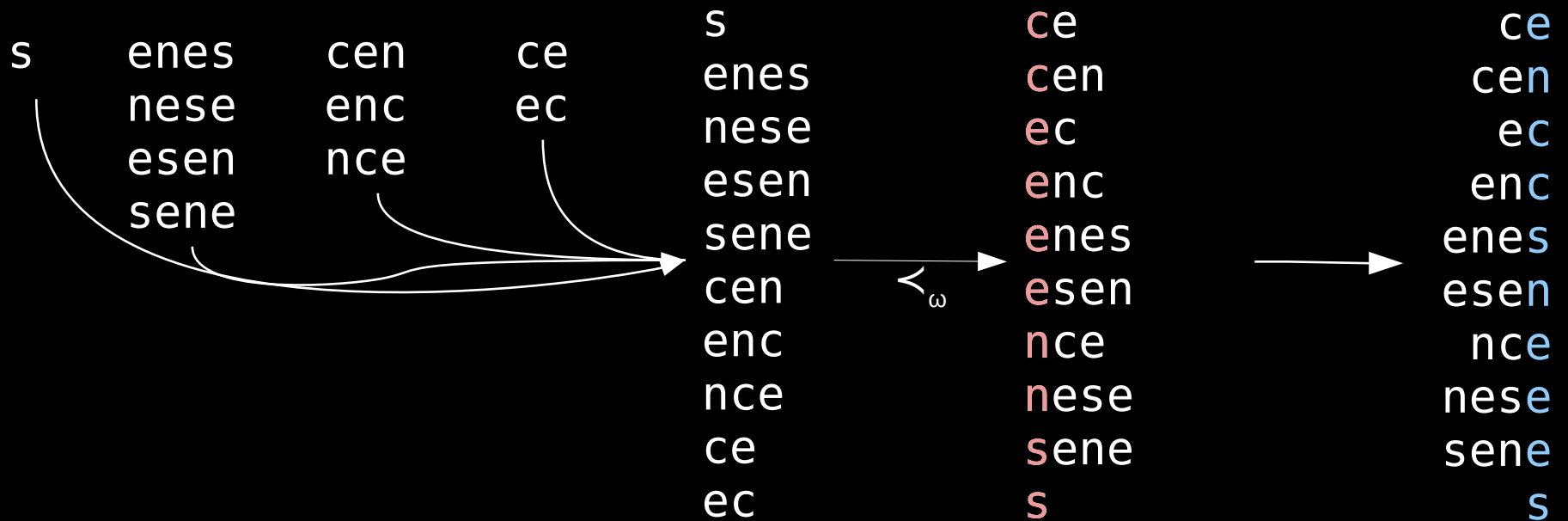
bijjective BWT of senescence

s | enes | cen | ce



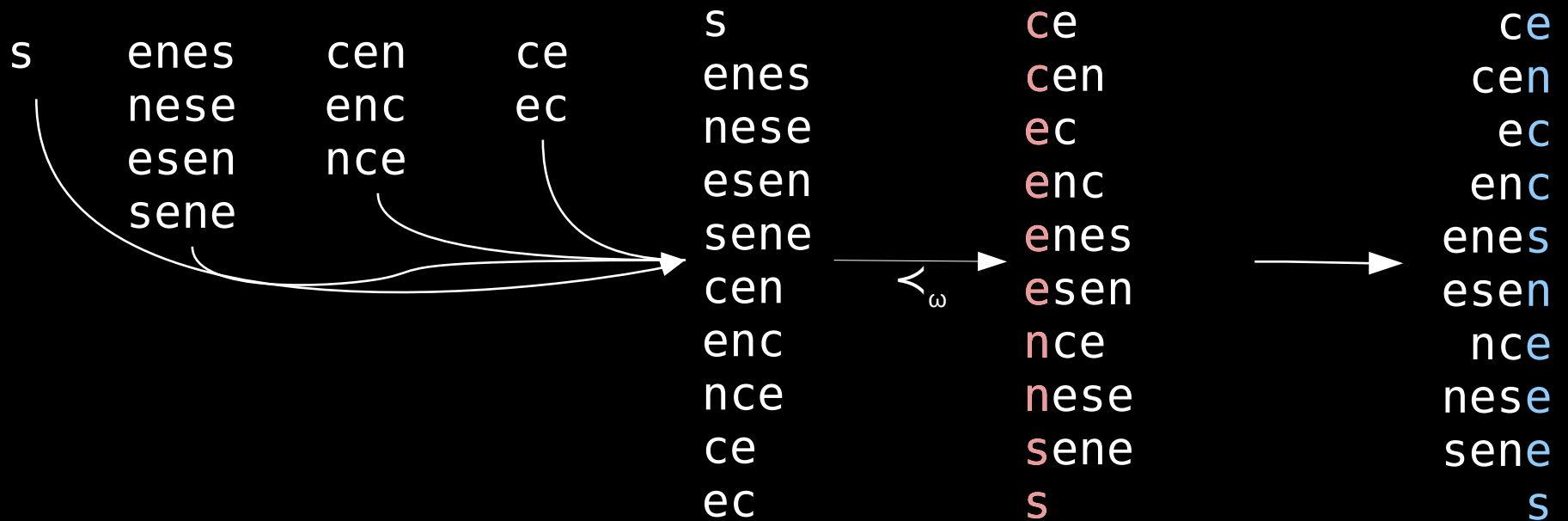
bijjective BWT of senescence

s | enes | cen | ce



bijjective BWT of senescence

s | enes | cen | ce



result: encsneees

motivation

properties of BBWT :

- no ϵ necessary
- BBWT is more compressible than BWT for various inputs

[Scott and Gill '12]

- BBWT is indexable (full text index)

[Bannai+ '19]

however, linear time construction was only conjectured!

time

- how much time does it take to sort?
- #conjugates = number of text positions = n
⇒ naively: $O(n^2)$ time
- can we use $O(n)$ time suffix sorting?

senescence の BWT

senescence

senescence の BWT

senescence

senescence

enescence

nescence

escence

scence

cence

ence

nce

ce

e

senescence の BWT

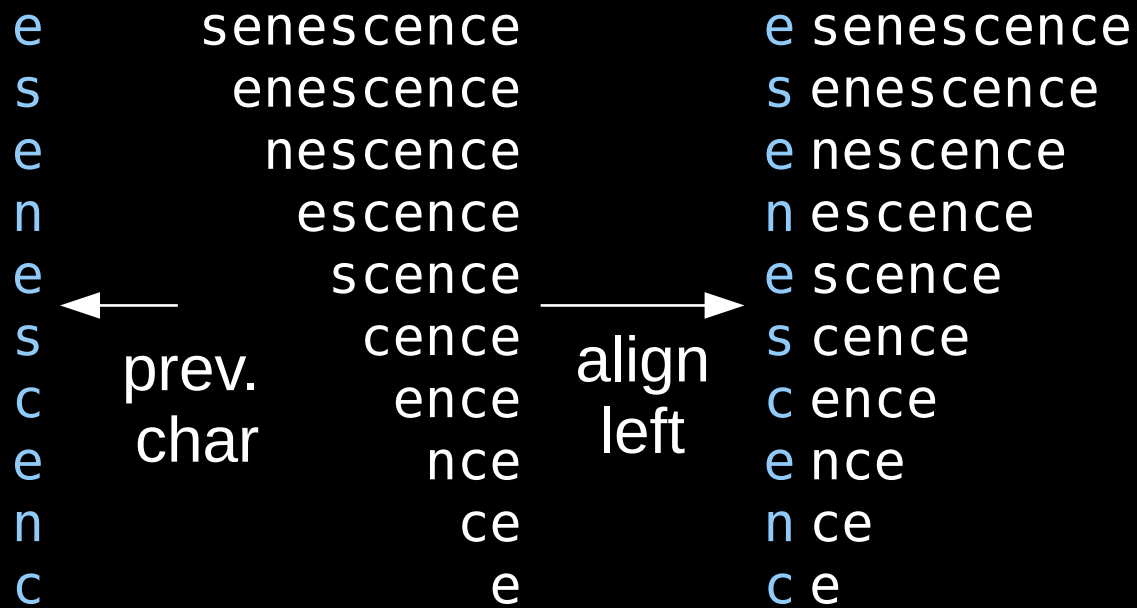
senescence

e	senescence
s	enescence
e	nescence
n	escence
e	science
s	cence
c	ence
e	nce
n	ce
c	e

← prev. char

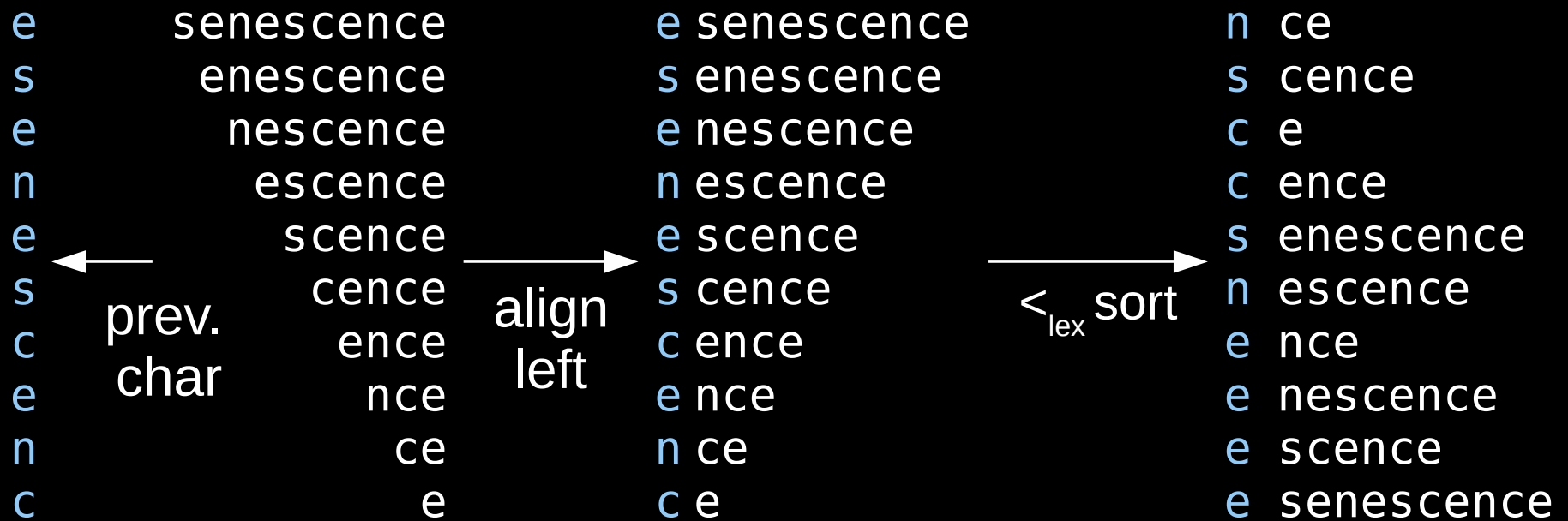
senescence の BWT

senescence



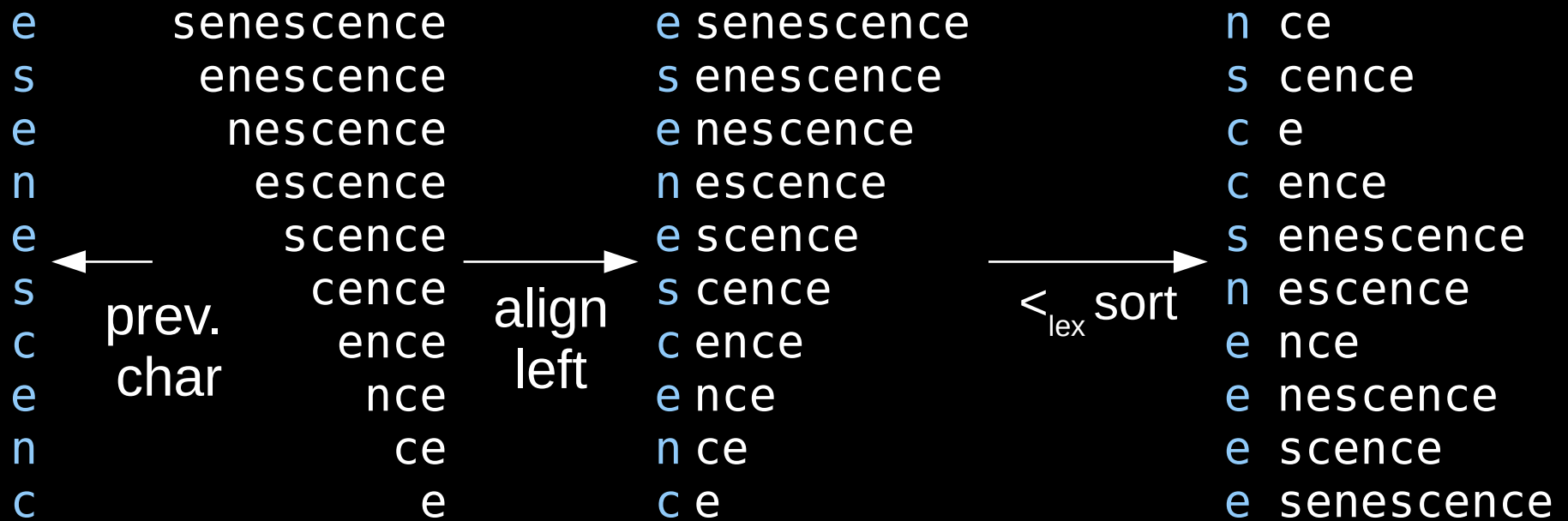
senescence の BWT

senescence



senescence の BWT

senescence



output: nscsneeee (BBWT: encsneees)

senescence の BBWT

s | enes | cen | ce

senescence の BBWT

s | enes | cen | ce

s
enes
nese
esen
sene
cen
enc
nce
ce
ec

senescence の BBWT

s | enes | cen | ce

s		s
s		enes
e		nese
n		esen
e	←	sene
n	last char.	cen
c		enc
e		nce
e		ce
c		ec

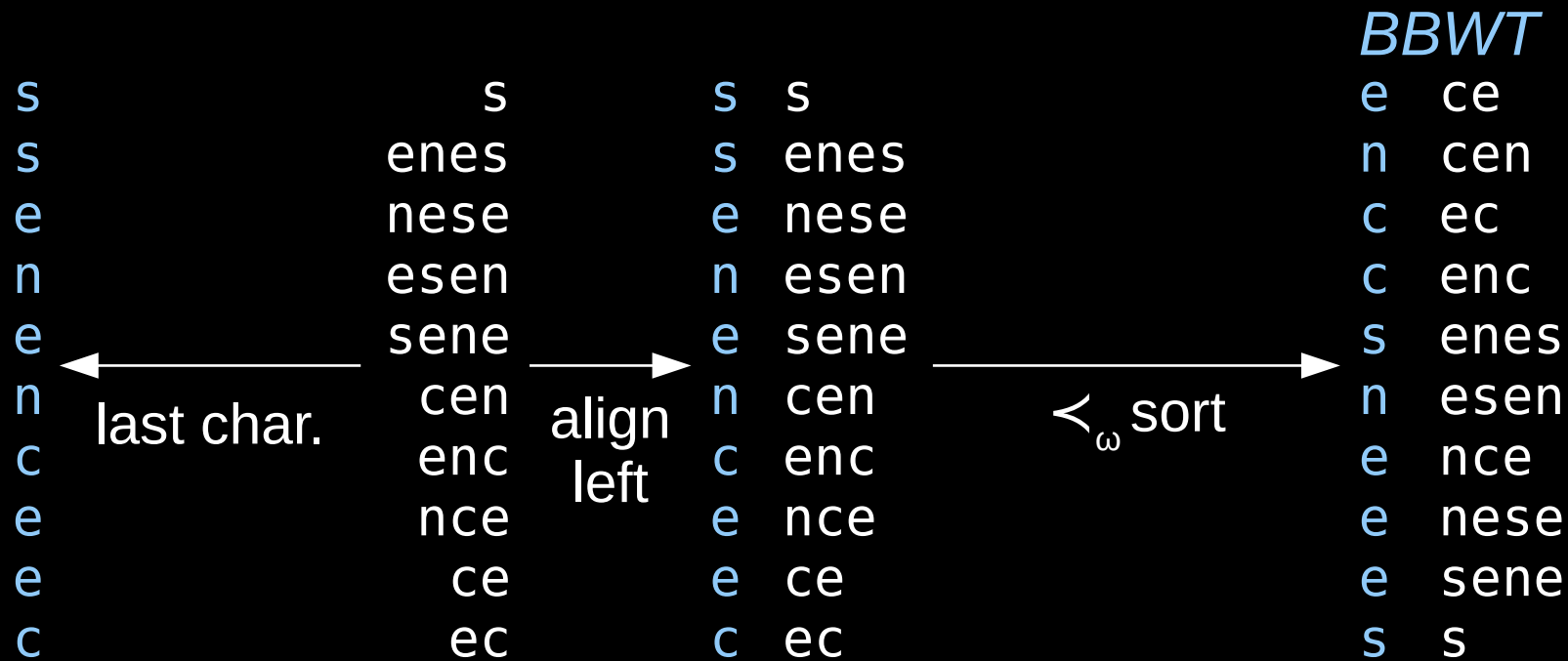
senescence の BBWT

s | enes | cen | ce

s		s		s	s
s		enes		s	enes
e		nese		e	nese
n		esen		n	esen
e	←	sene	→	e	sene
n	last char.	cen	align	n	cen
c		enc	left	c	enc
e		nce		e	nce
e		ce		e	ce
c		ec		c	ec

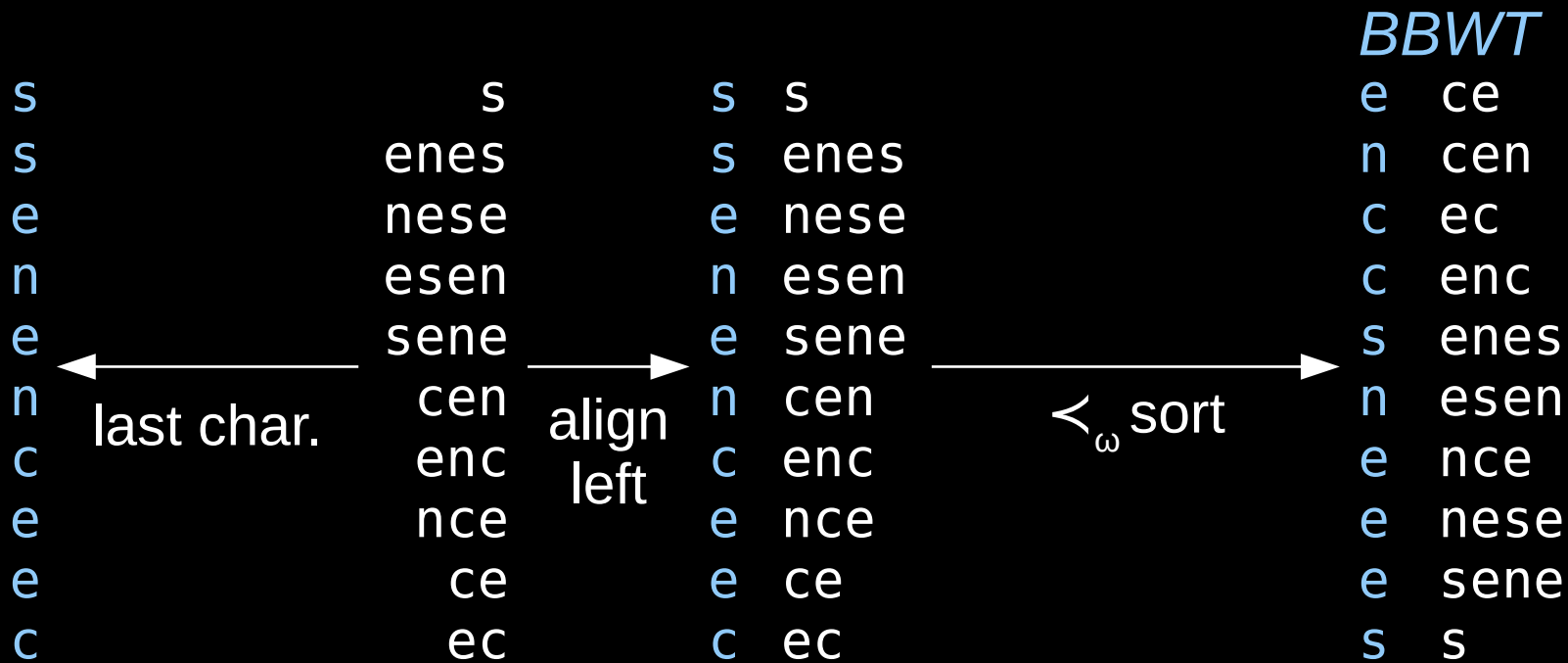
senescence の BBWT

s | enes | cen | ce



senescence の BBWT

s | enes | cen | ce



output: encsneees

SA \Rightarrow BWT

- $BWT[i] = T[SA[i]-1]$
- $SA[i]$ = starting position of the i th smallest suffix respective to lexicographic order

SAIS:

- famous linear time SA construction algorithm
- $O(n)$ time for integer alphabets

LSA \Rightarrow BBWT

- $BBWT[i] = T[LSA[i]-1]$
- $LSA[i]$ = starting position of i th smallest conjugate respective to \prec_{ω} order

[Hon+ '11]

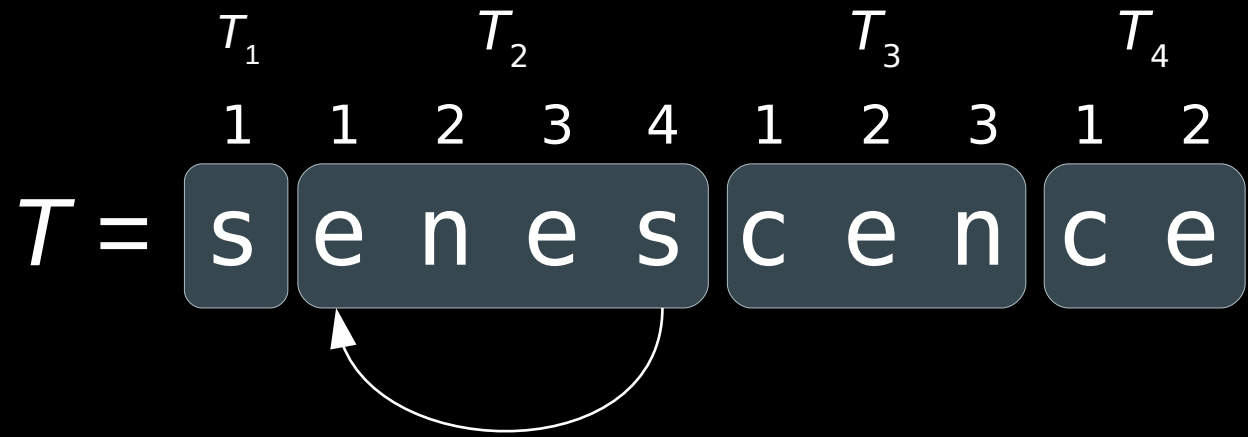
LSAIS:

- LSA construction algorithm
- sorts conjugates instead of suffixes

rewinding

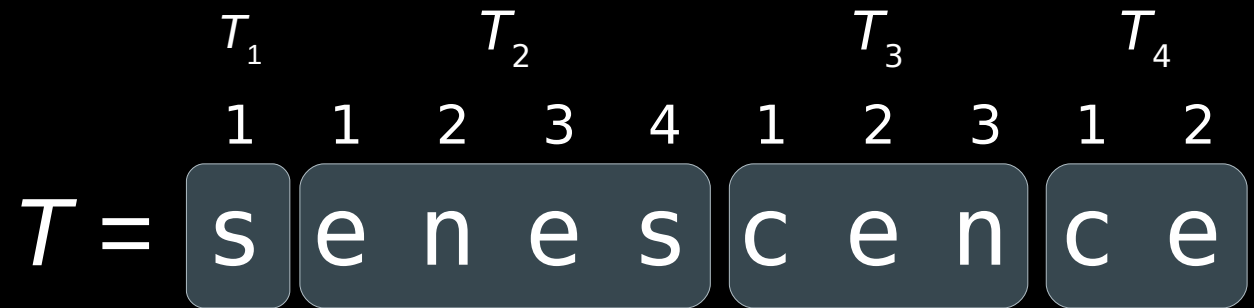
$T =$ T_1 T_2 T_3 T_4
1 1 2 3 4 1 2 3 1 2
s e n e s c e n c e

rewinding



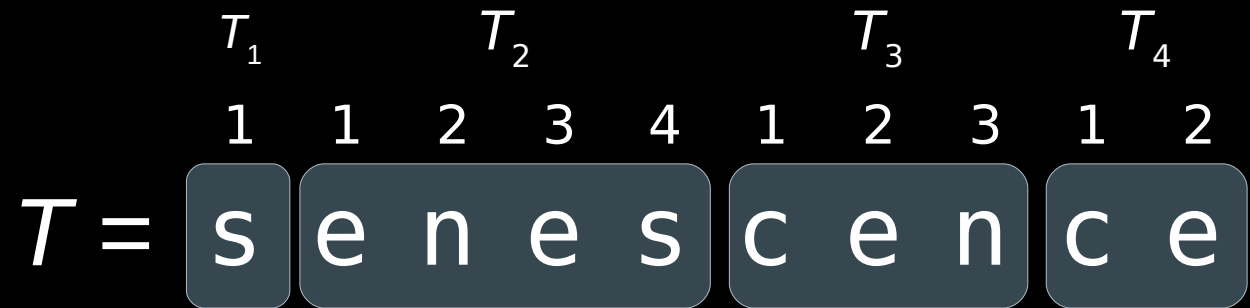
- $T_2[5] := T_2[1]$

rewinding



- $T_2[5] := T_2[1]$
- $T_2[4..] := T_2[4]T_2[1]T_2[2]T_2[3]T_2[4]T_2[1]...$

rewinding

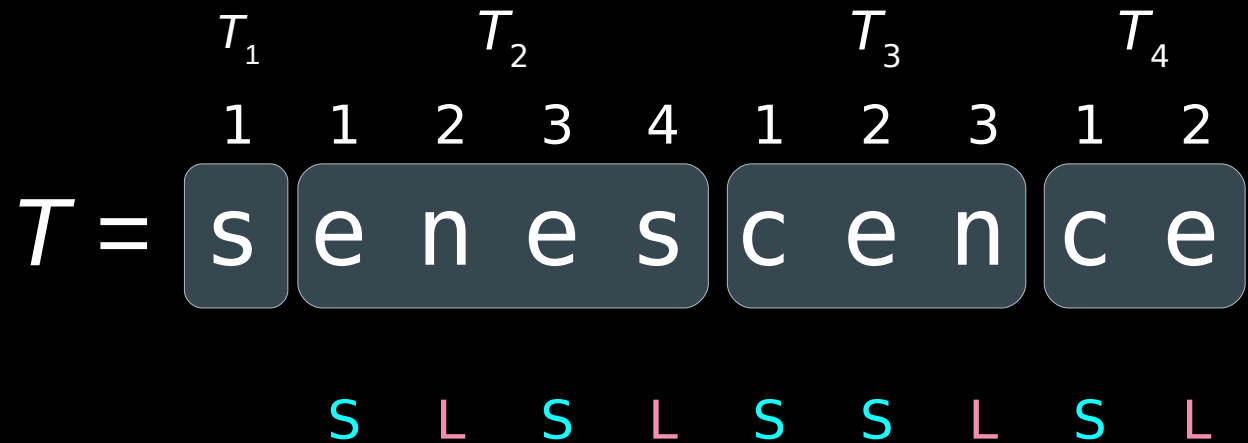


- $T_2[5] := T_2[1]$
- $T_2[4..] := T_2[4]T_2[1]T_2[2]T_2[3]T_2[4]T_2[1]...$

in general, for any x define:

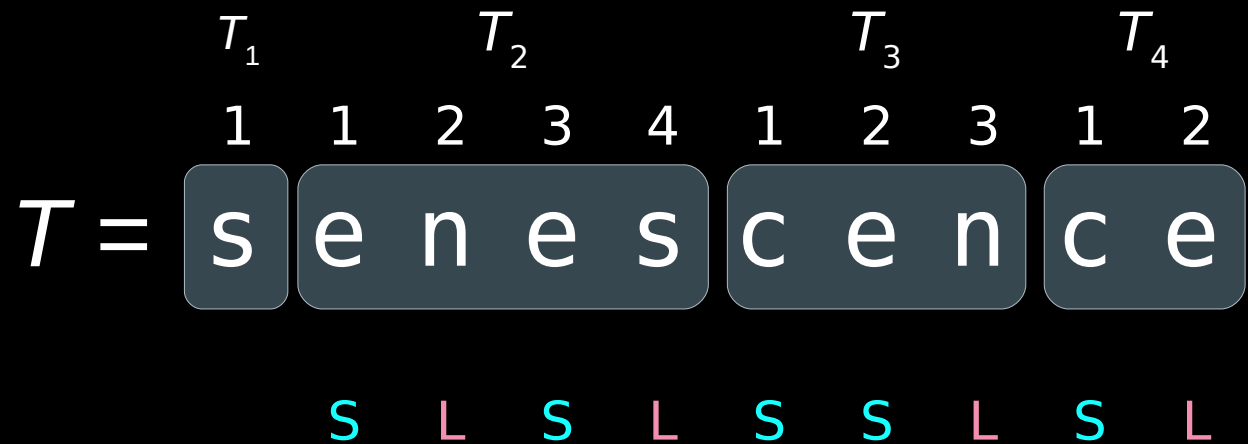
- $T_x[|T_x|+j] := T_x[(|T_x|+j-1) \bmod |T_x|+1]$, $j \geq 0$
- $T_x[0] := T_x[|T_x|]$
- $T_x[i..] := T_x[i] \dots T_x[|T_x|] T_x[1] \dots$

L / S type



- $T_x[i] <_{\text{lex}} T_x[i+1] \Rightarrow T_x[i]$ is S type
- $T_x[i] >_{\text{lex}} T_x[i+1] \Rightarrow T_x[i]$ is L type
- $T_x[i] = T_x[i+1] \Rightarrow T_x[i]$ and $T_x[i+1]$ are same type

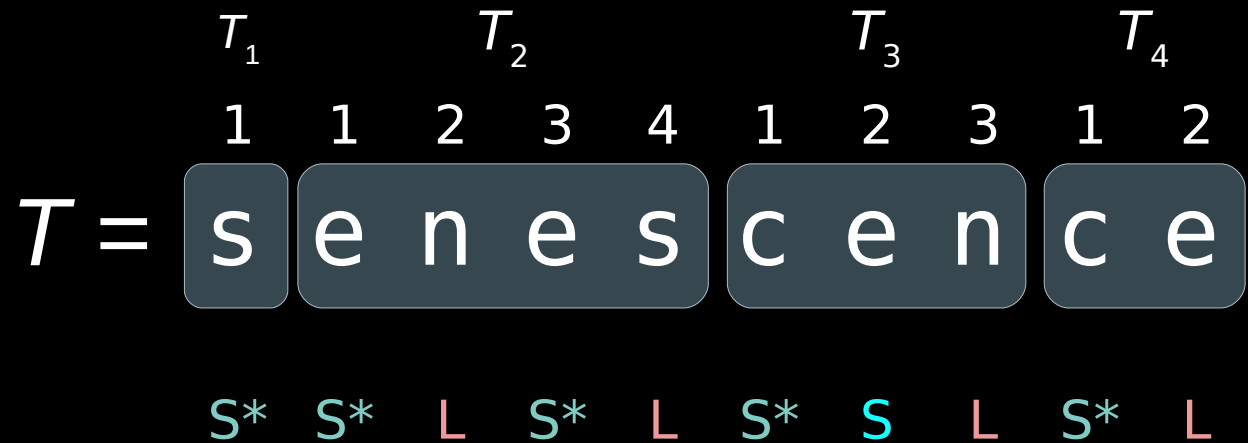
L / S type



- $T_x[i] <_{\text{lex}} T_x[i+1] \Rightarrow T_x[i]$ is S type
- $T_x[i] >_{\text{lex}} T_x[i+1] \Rightarrow T_x[i]$ is L type
- $T_x[i] = T_x[i+1] \Rightarrow T_x[i]$ and $T_x[i+1]$ are same type

thanks to the Lyndon factorization the S / L types are the same as in the original SAIS

S* type



- If $T_x[i]$ is S type and $T_x[i-1]$ is L type, then $T[i]$ is S* type
- $T_x[1]$ is always S* type $\forall x$

LMS (substrings)

$T =$

1	2	3	4	5	6	7	8	9	10
s	e	n	e	s	c	e	n	c	e
S^*	S^*	L	S^*	L	S^*	S	L	S^*	L

for $1 \leq i < j \leq |T_x|+1$:

$T_x[i..j]$ is called LMS if

- $T_x[i]$ and $T_x[j]$ are S^* type and
- $T_x[k]$ is S^* type

$\forall k \in [i+1 .. j-1]$

start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec

LSAIS algorithm

- 1) sort all LMS
- 2) place S^* types
- 3) induce L types
- 4) induce S types

sort all LMS

start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec

sort all LMS

start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec



$<_{\text{lex}}$ sort

start pos.	LMS
9	cec
6	cenc
2	ene
4	ese
1	ss

S/L type bucket allocation

$T =$

1	2	3	4	5	6	7	8	9	10
s	e	n	e	s	c	e	n	c	e
S*	S*	L	S*	L	S*	S	L	S*	L

$LSA =$

S	L	S			L		L	S	
c	e				n		s		

place S^* types

pos. LMS

9 cec

6 cenc

2 ene

4 ese

1 ss

$T =$

	1	2	3	4	5	6	7	8	9	10
	s	e	n	e	s	c	e	n	c	e
	S^*	S^*	L	S^*	L	S^*	S	L	S^*	L

$LSA =$

9	6		2	4					1	
S		L		S			L		L	S
c				e			n			s

induce L types



$LSA =$

9	6	10	2	4		8	3	5	1
S		L	S			L		L	S
c		e				n		s	

induce **S** types



$LSA =$

9	6	10	7	2	4	8	3	5	1
S		L	S			L		L	S
c		e				n		s	

$$\text{BBWT}[i] = T[\text{LSA}[i]-1]$$

$T =$

1	2	3	4	5	6	7	8	9	10
s	e	n	e	s	c	e	n	c	e
s*	s*	L	s*	L	s*	s	L	s*	L

$LSA =$

9	6	10	7	2	4	8	3	5	1
---	---	----	---	---	---	---	---	---	---

$LSA-1 =$

10	8	9	6	5	3	7	2	4	1
----	---	---	---	---	---	---	---	---	---

$T[LSA-1] =$

e	n	c	c	s	n	e	e	e	s
---	---	---	---	---	---	---	---	---	---

time analysis

SAIS

- sorts LMS recursively
- given SAIS needs $\tau(n)$ time,
 $\tau(n) = O(n) + \tau(n/2) = O(n)$ since
the number of LMS $\leq n/2$

LSAIS

- number of LMS can be $\Theta(n)$ in all recursion steps
- still $O(n^2)$ time? ☹️

example

$T = b \dots baababaabab \dots aabab$

$b \mid \dots \mid b \mid aabab \mid aabab \mid \dots \mid aabab$

example

$T = b \dots baababaabab \dots aabab$

$b \mid \dots \mid b \mid aabab \mid aabab \mid \dots \mid aabab$

LMS : _____ assign ranks according to
lexicographic order

– $b \rightarrow 3$

– $aab \rightarrow 1$

– $ab \rightarrow 2$

example

$T = b \dots baababaabab \dots aabab$

$b \mid \dots \mid b \mid aabab \mid aabab \mid \dots \mid aabab$

LMS : _____ assign ranks according to
lexicographic order

– $b \rightarrow 3$

– $aab \rightarrow 1$

– $ab \rightarrow 2$

$3 \mid \dots \mid 3 \mid 12 \mid 12 \mid \dots \mid 12$ (recursion step)

example

$T = b \dots baababaabab \dots aabab$

b | . . . | b | aabab | aabab | . . . | aabab

LMS :

- b → 3

- aab → 1

- ab → 2

assign ranks according to
lexicographic order

same amount

3 | . . . | 3 | 12 | 12 | . . . | 12 (recursion step)

example

$T = b \dots baababaabab \dots aabab$

$b \mid \dots \mid b \mid aabab \mid aabab \mid \dots \mid aabab$

LMS:

- $b \rightarrow 3$

- $aab \rightarrow 1$

- $ab \rightarrow 2$

assign ranks according to
lexicographic order

same amount

$3 \mid \dots \mid 3 \mid 12 \mid 12 \mid \dots \mid 12$ (recursion step)

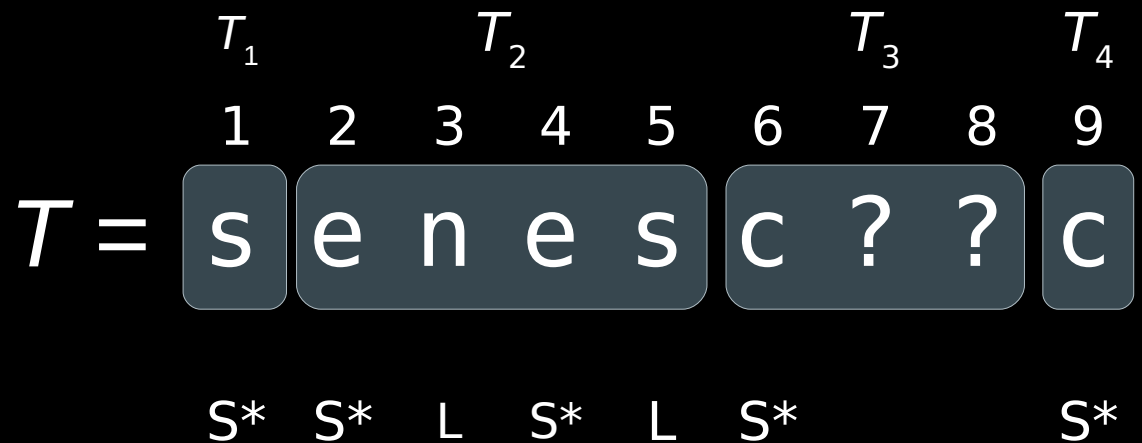
same Lyndon factorization

lemma

- the string on the ranked LMS subtring has the same **Lyndon** factorization
- compared to any character c , all LMS starting with c are larger with respect to $<_{\omega}$

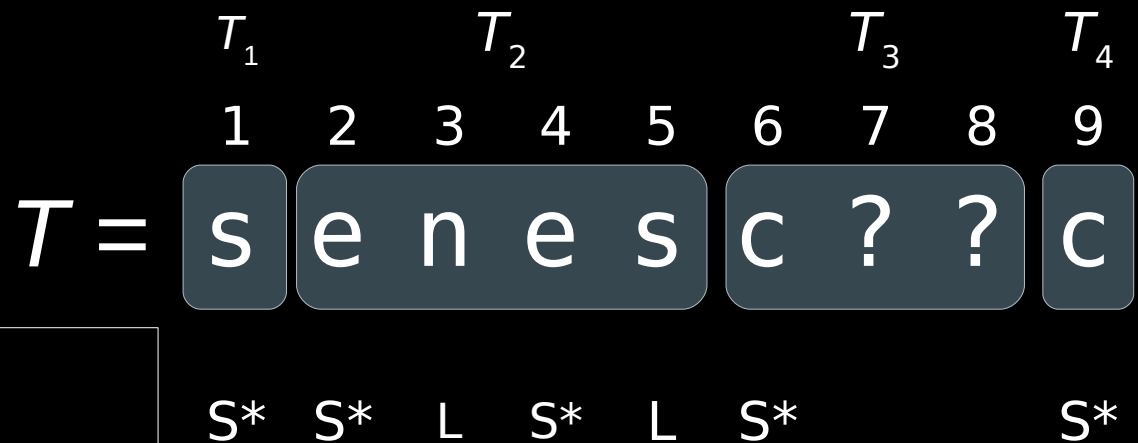
lemma

- the string on the ranked LMS subtring has the same **Lyndon** factorization
- compared to any character c , all LMS starting with c are larger with respect to $<_{\omega}$



lemma

- the string on the ranked LMS subtring has the same **Lyndon** factorization
- compared to any character c , all LMS starting with c are larger with respect to $<_{\omega}$



\Rightarrow can omit LMS of length 1 in the recursion

summary

- bijective BWT construction
 - $O(n)$ time on integer alphabets
- methods
 - adapt SAIS
 - sort conjugates in \prec_{ω} instead of suffixes in lex. order
 - skip LMS of length 1 in recursion

summary

- bijective BWT construction
 - $O(n)$ time on integer alphabets
- methods
 - adapt SAIS
 - sort conjugates in \prec_{ω} instead of suffixes in lex. order
 - skip LMS of length 1 in recursion

all questions are welcome!