Dynamic Suffix Array with experiments and a specific emphasis on LCP arrays

Mikaël Salson, Thierry Lecroq, Martine Léonard, Laurent Mouchard

LITIS, University of Rouen

Stringology 2009
A research workshop of the Israel Science Foundation

Motivation

Recent Advances in High-Throughput Sequencing

Indexing large dynamic texts (genomes) to speed up approximate pattern matching

Motivation

Recent Advances in High-Throughput Sequencing

Indexing large dynamic texts (genomes) to speed up approximate pattern matching

Our choice

- Updating the FM-Index:
  - updating the Burrows-Wheeler Transform,
  - updating the suffix array.

Motivation

Recent Advances in High-Throughput Sequencing

Indexing large dynamic texts (genomes) to speed up approximate pattern matching

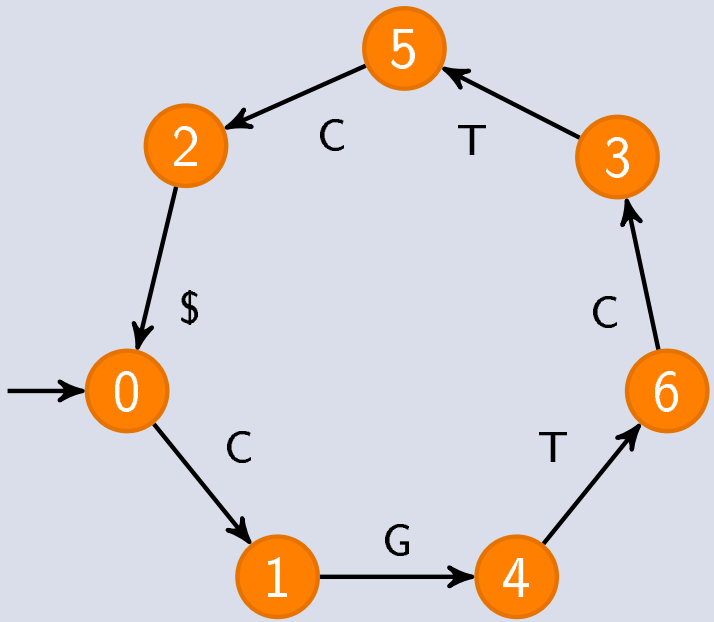
Our choice

- Updating the FM-Index:
  - updating the Burrows-Wheeler Transform, 
  - updating the suffix array.

Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{\mathtt{CTCTGC\$}} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{\mathtt{CTGCTGC\$}}$

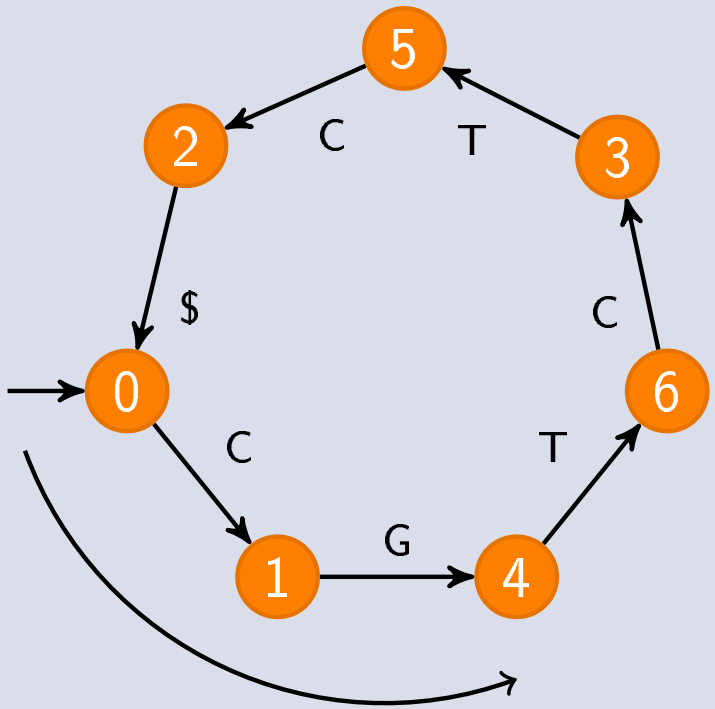
i	F	L	SA	ISA
0	\$	C	6	2
1	C	G	5	5
2	C	\$	0	3
3	C	T	2	6
4	G	T	4	4
5	T	C	1	1
6	T	C	3	0



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

i	F	L	SA	ISA
0	\$	C	6	2
1	C	G	5	5
2	C	\$	0	3
3	C	T	2	6
4	G	T	4	4
5	T	C	1	1
6	T	C	3	0

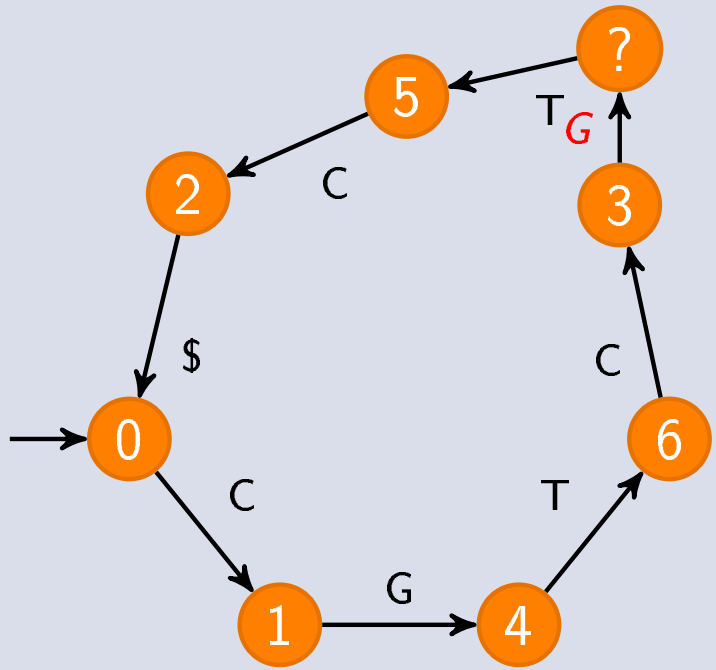


We can recover ISA backwards by reading the number in each state (LF).

Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{\mathtt{CTCTGC\$}} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{\mathtt{CTGCTGC\$}}$

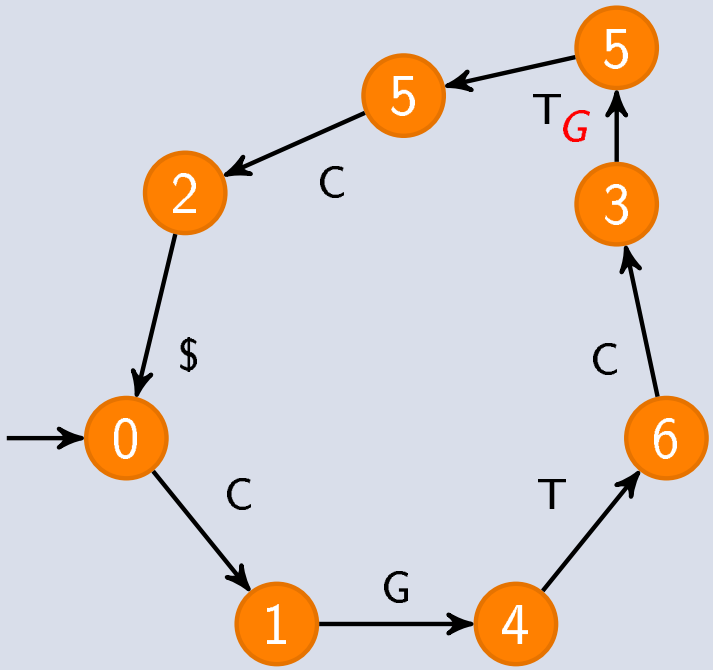
i	F	L	SA	ISA
0	\$	C	6	2
1	C	G	5	5
2	C	\$	0	3
3	C	*G*	2	6
4	G	T	4	4
5	T	C	1	1
6	T	C	3	0



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

i	F	L	SA	ISA
0	\$	C	6	2
1	C	G	5	5
2	C	\$	0	5
3	C	G	2	3
4	G	T	4	6
5	G	T	2	4
6	T	C	1	1
7	T	C	3	0

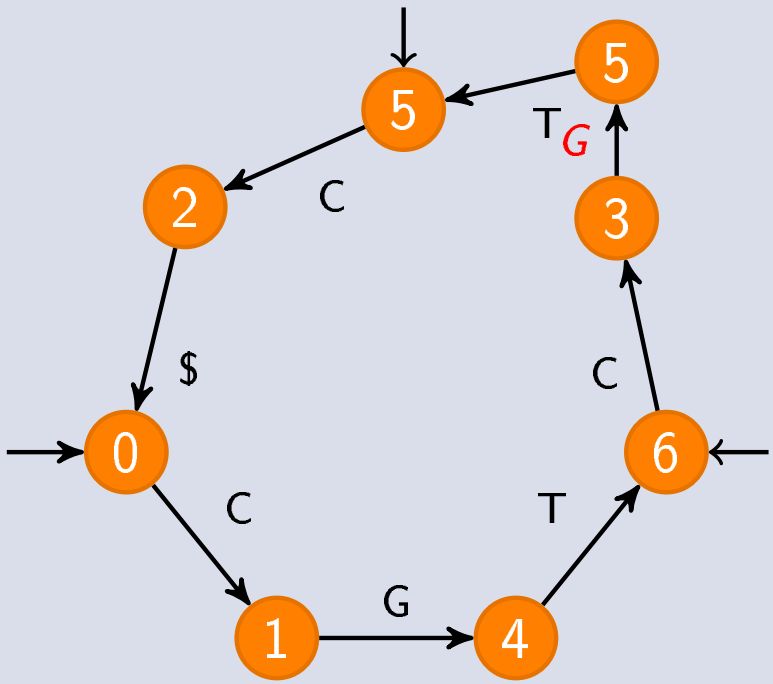


New element is inserted at position 5 in the *BWT* and *SA*.

Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{\mathrm{C\,T\,C\,T\,G\,C\,\$}} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{\mathrm{C\,T\,G\,C\,T\,G\,C\,\$}}$

i	F	L	SA	ISA
0	\$	C	→ 6	2
1	C	G	→ 5	5 ←
2	C	\$	0	5
3	C	G	→ 2	3
4	G	T	→ 4	6 ←
5	G	T	2	4
6	T	C	1	1
7	T	C	→ 3	0

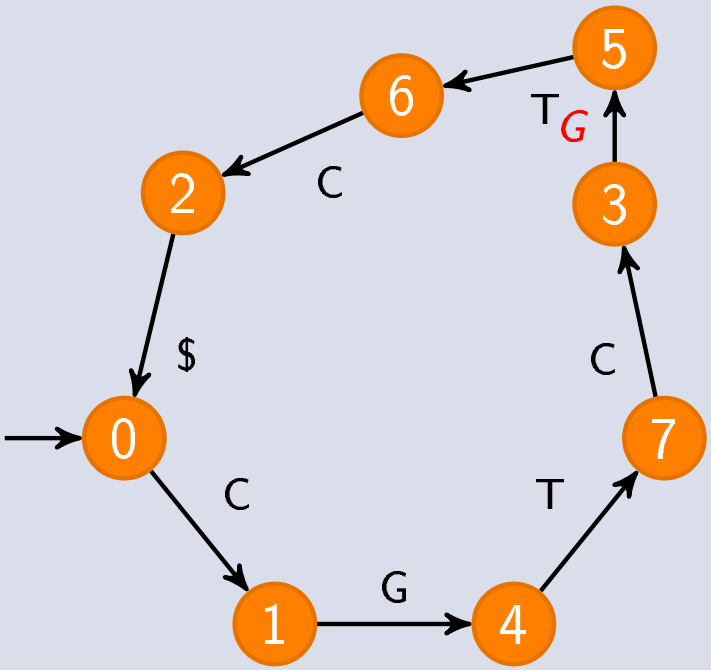


Because of the insertion, positions are shifted.
We have to increment those values.

Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

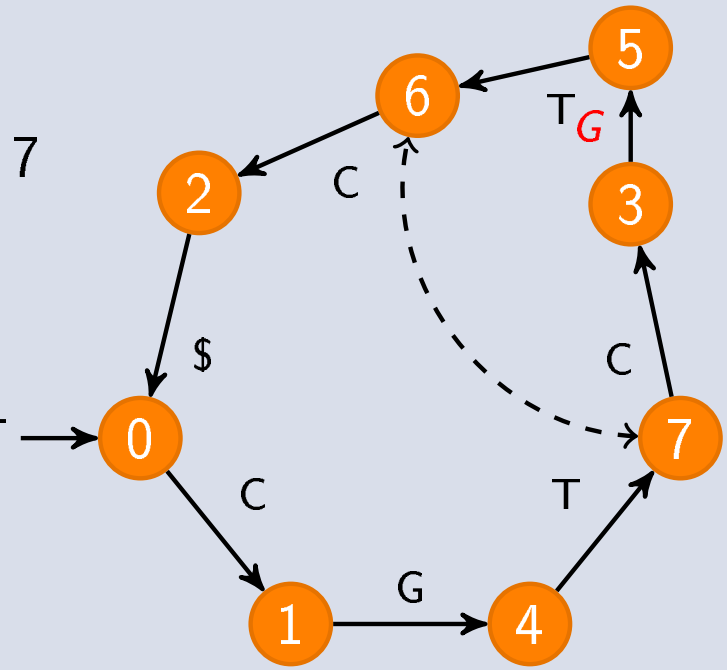
i	F	L	SA	ISA
0	\$	C	7	2
1	C	G	6	6
2	C	\$	0	5
3	C	*G*	3	3
4	G	T	5	7
5	*G*	T	2	4
6	T	C	1	1
7	T	C	4	0



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

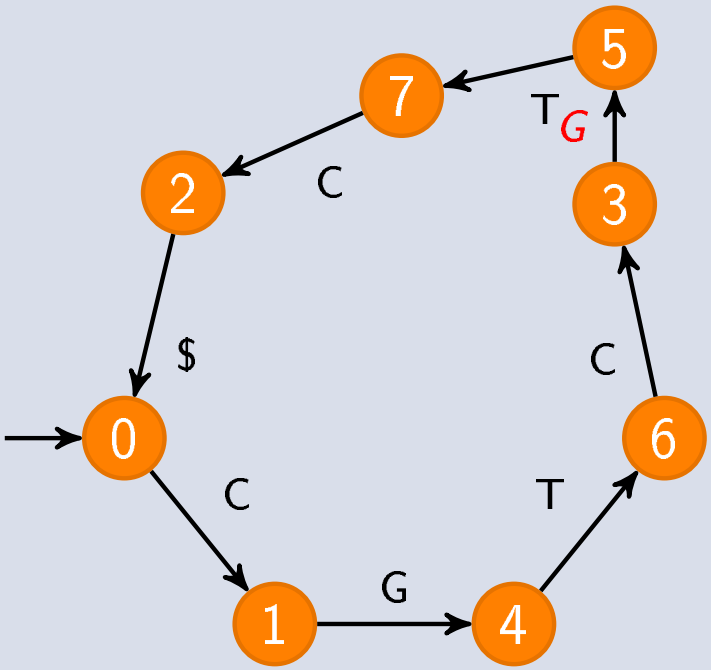
i	F	L	SA	ISA	
0	\$	C	7	2	
1	C	G	6	6	$\rightarrow 7$
2	C	\$	0	5	
3	C	G	3	3	
4	G	T	5	7	-1
5	G	T	2	4	
6	T	C	1	1	
7	T	C	4	0	



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

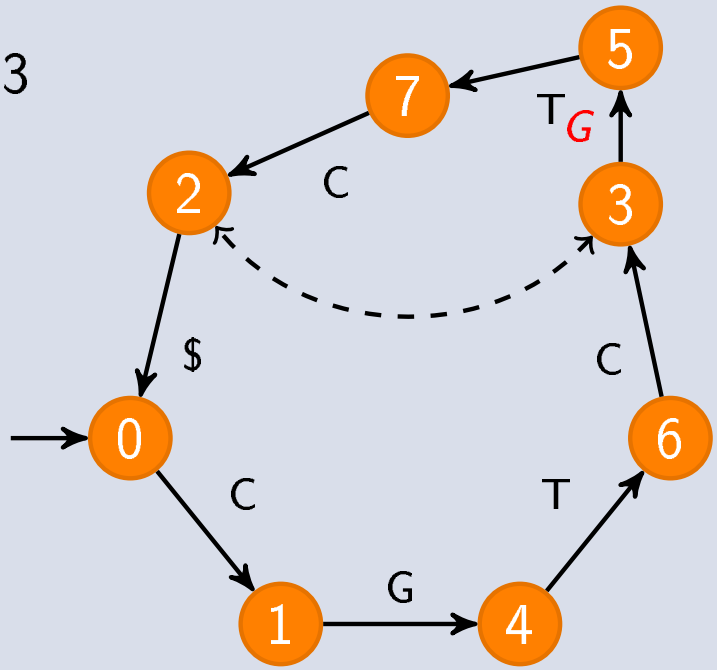
i	F	L	SA	ISA
0	\$	C	7	2
1	C	G	6	7
2	C	\$	0	5
3	C	*G*	3	3
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

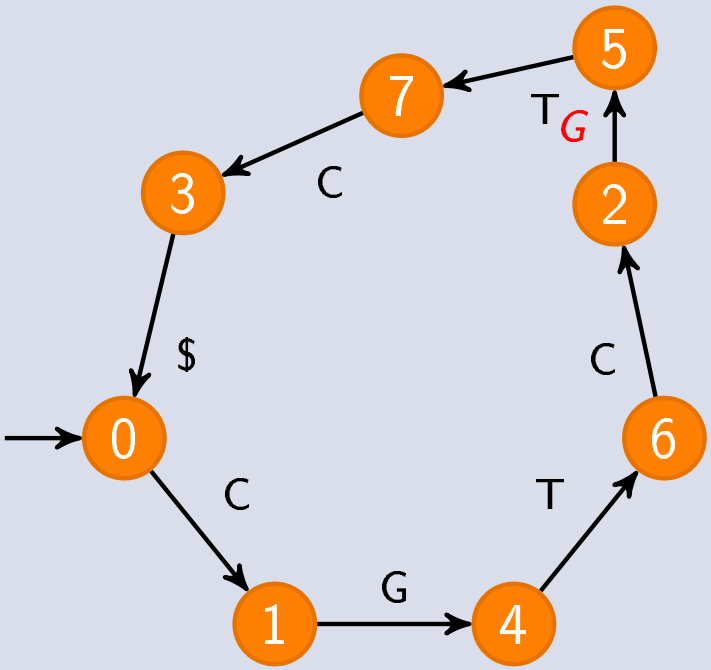
i	F	L	SA	ISA
0	\$	C	7	$2 \rightarrow 3$
1	C	G	6	7
2	C	\$	0	5
3	C	G	3	$3 \quad -1$
4	G	T	5	6
5	G	T	2	4
6	T	C	4	1
7	T	C	1	0



Updating the suffix array

$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6}{C\ T\ C\ T\ G\ C\ \$} \rightarrow T' = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{C\ T\ G\ C\ T\ G\ C\ \$}$

i	F	L	SA	ISA
0	$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0



Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Sorted, very limited space consumption.

Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Sorted, very limited space consumption.

Compressed, space related to the entropy of the text.

Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Sorted, very limited space consumption.

Compressed, space related to the entropy of the text.

$8n$ bytes.

Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Sorted, very limited space consumption.

Compressed, space related to the entropy of the text.

$8n$ bytes.

Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

Storage

i	F	L	SA	ISA
0	\$	C	7	3
1	C	G	6	7
2	C	*G*	3	5
3	C	\$	0	2
4	G	T	5	6
5	*G*	T	2	4
6	T	C	4	1
7	T	C	1	0

Sorted, very limited space consumption.

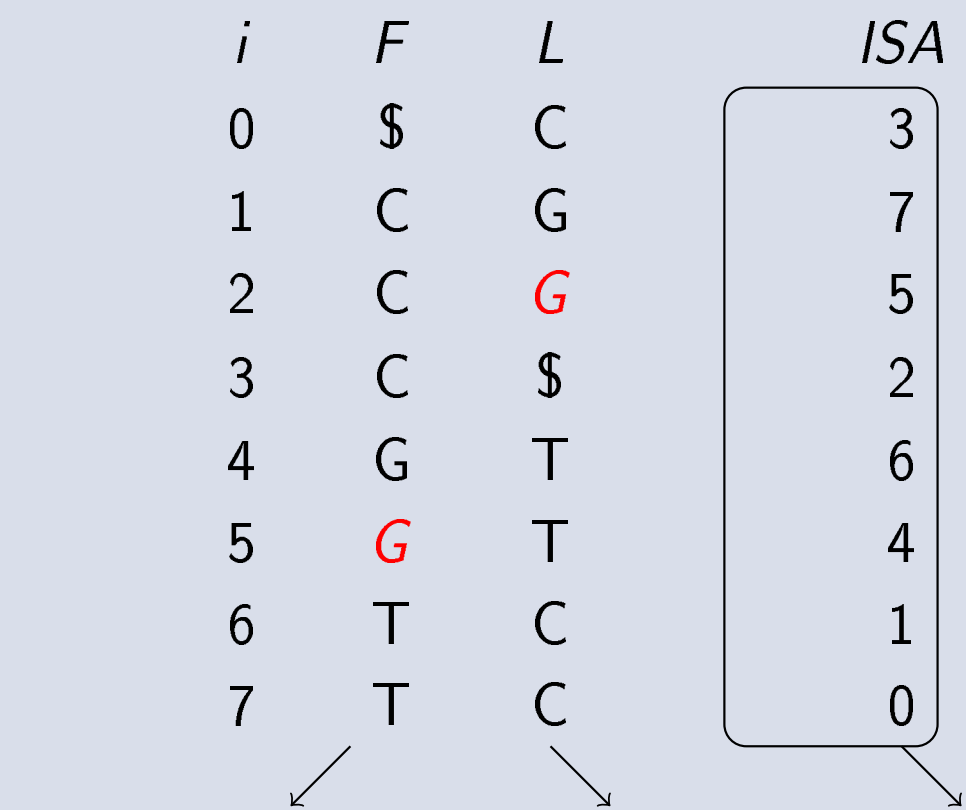
Compressed, space related to the entropy of the text.

$8n$ bytes.

Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

Storage



Sorted, very limited space consumption.

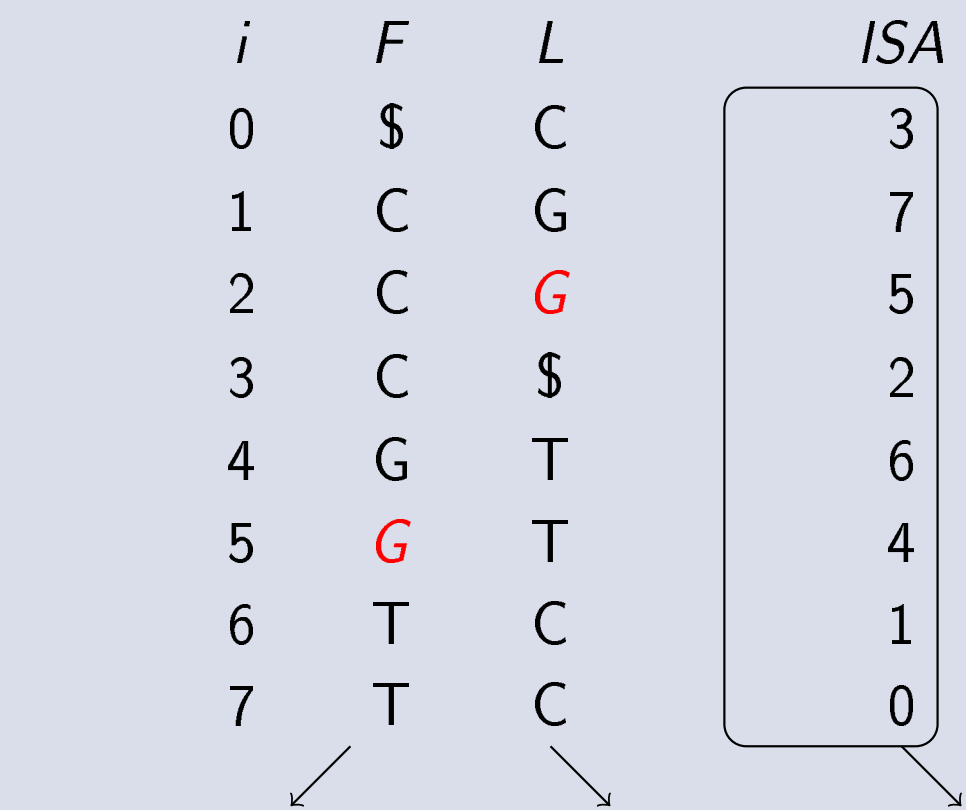
Compressed, space related to the entropy of the text.

$4n$ bytes.

Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

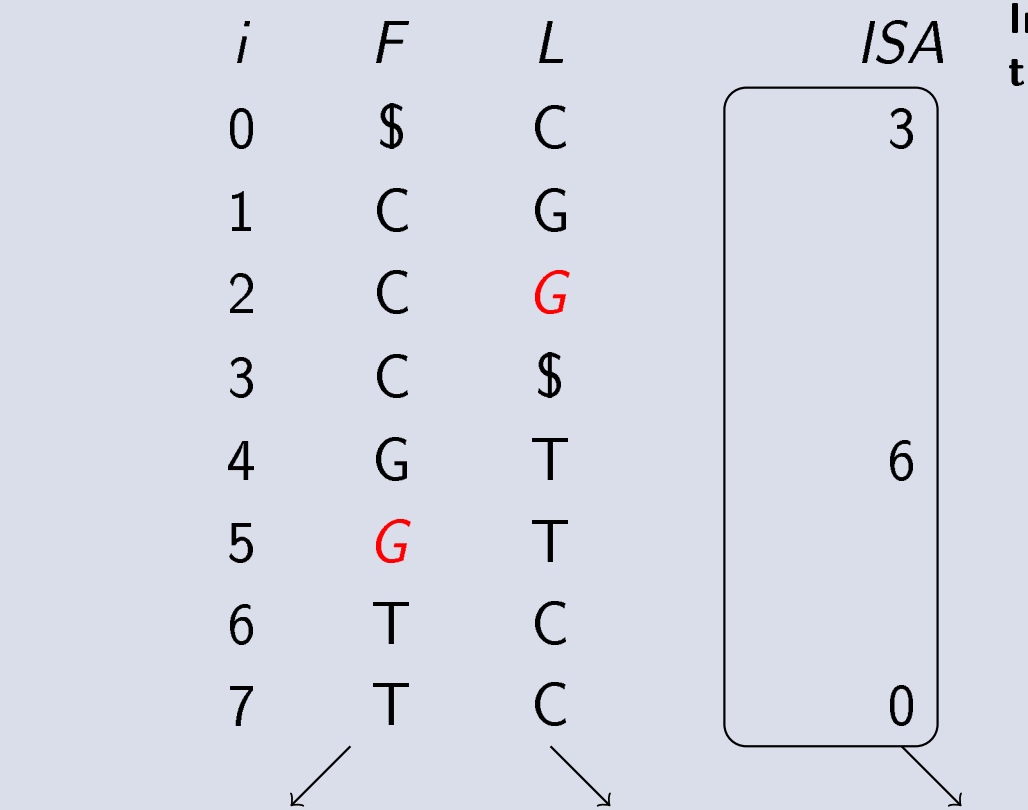
Storage



Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

Storage



Sorted, very limited space consumption.

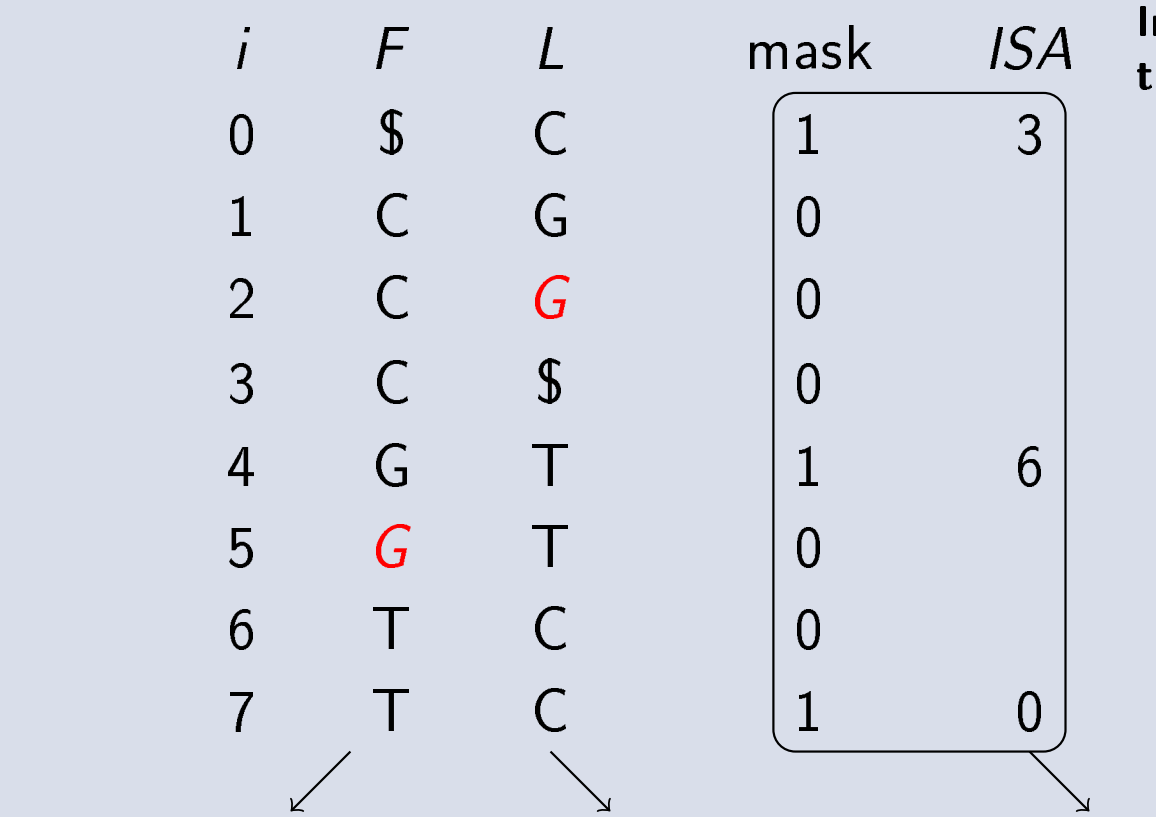
Compressed, space related to the entropy of the text.

$o(n)$.

Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

Storage



Improving space consumption:

- Keep only one array
- LF and ISA are strongly related
  $\Rightarrow$ we can recover ISA using LF
  $\Rightarrow$ keep only few values, e.g. one out of $\log n \log\log n$, to guarantee a quick computation of any value.

Underlying structures

Original

i	F
0	\$
1	C
2	C
3	C
4	G
5	G
6	T
7	T

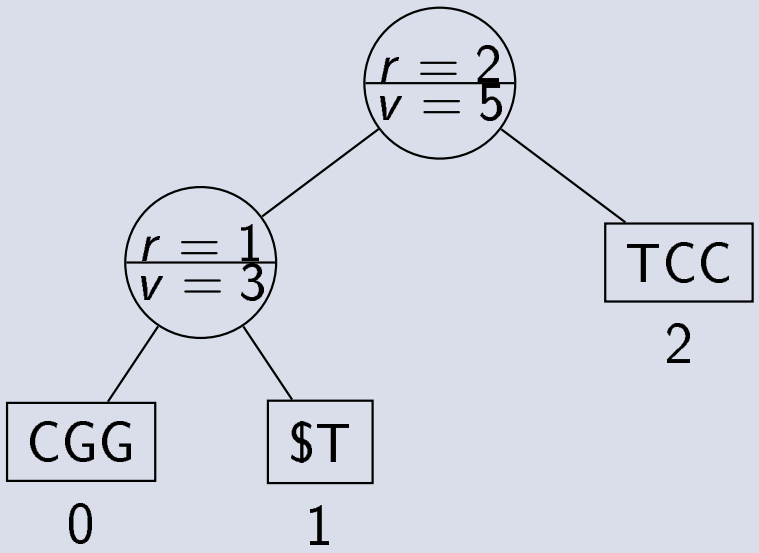
Storage of F

c	$Count$
\$	0
C	1
G	4
T	6

Original

i	L
0	C
1	G
2	G
3	\$
4	T
5	T
6	C
7	C

Storage of L — González and Navarro, 2008



	Leaves		
	0	1	2
$S_\$$	0	1	0
S_C	1	0	2
S_G	2	0	0
S_T	0	1	1

r: number of leaves in left subtree.

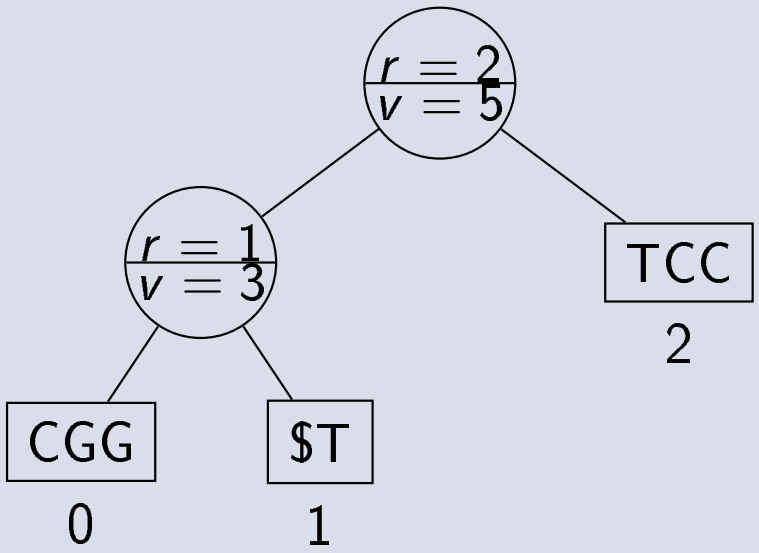
v: number of letters in left subtree.

Underlying structures

Original

i	L	
0	C	0
1	G	
2	G	
3	\$	1
4	T	
5	T	2
6	C	
7	C	

Storage of L — González and Navarro, 2008



	Leaves		
	0	1	2
$S_\$$	0	1	0
S_C	1	0	2
S_G	2	0	0
S_T	0	1	1

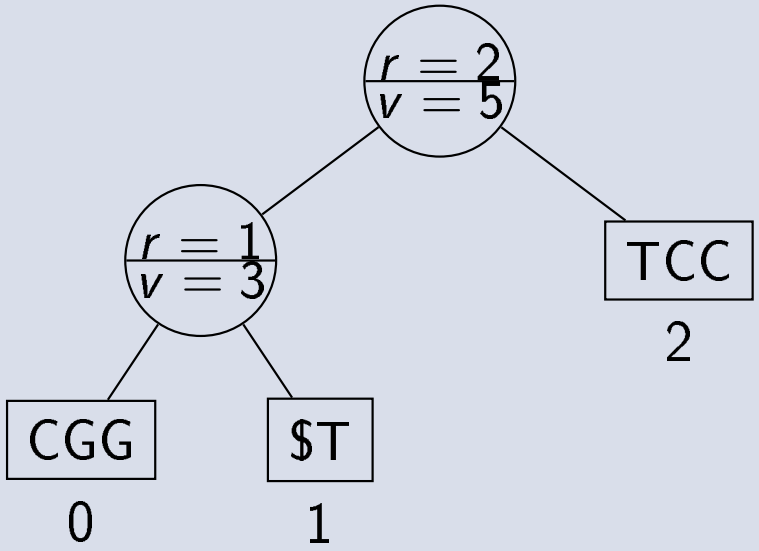
r: number of leaves in left subtree.

v: number of letters in left subtree.

Original

i	L	
0	C	0
1	G	
2	G	
3	\$	1
4	T	
5	T	2
6	C	
7	C	

Storage of L — González and Navarro, 2008



Leaves

	0	1	2
$S_\$$	0	1	0
S_C	1	0	2
S_G	2	0	0
S_T	0	1	1

2 Gs in leaf 0

r: number of leaves in left subtree.

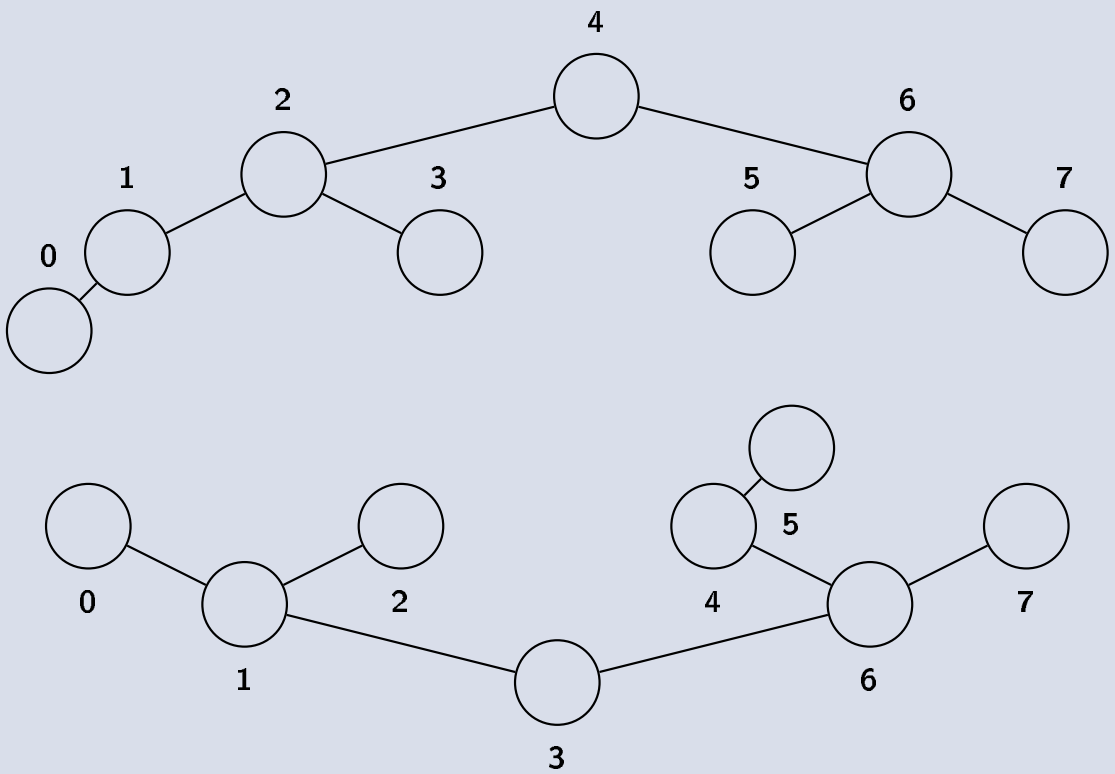
v: number of letters in left subtree.

Underlying structures

Original

i	ISA
0	3
1	7
2	5
3	2
4	6
5	4
6	1
7	0

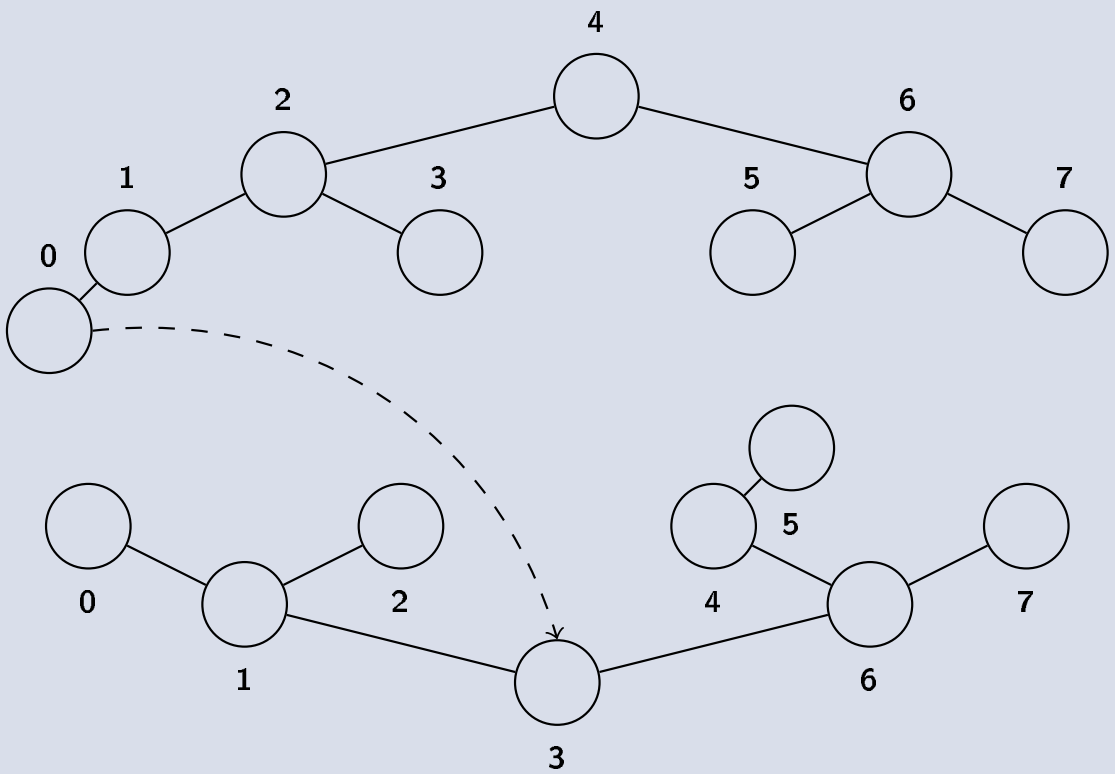
Storage of *ISA*



Original

i	ISA
0	3
1	7
2	5
3	2
4	6
5	4
6	1
7	0

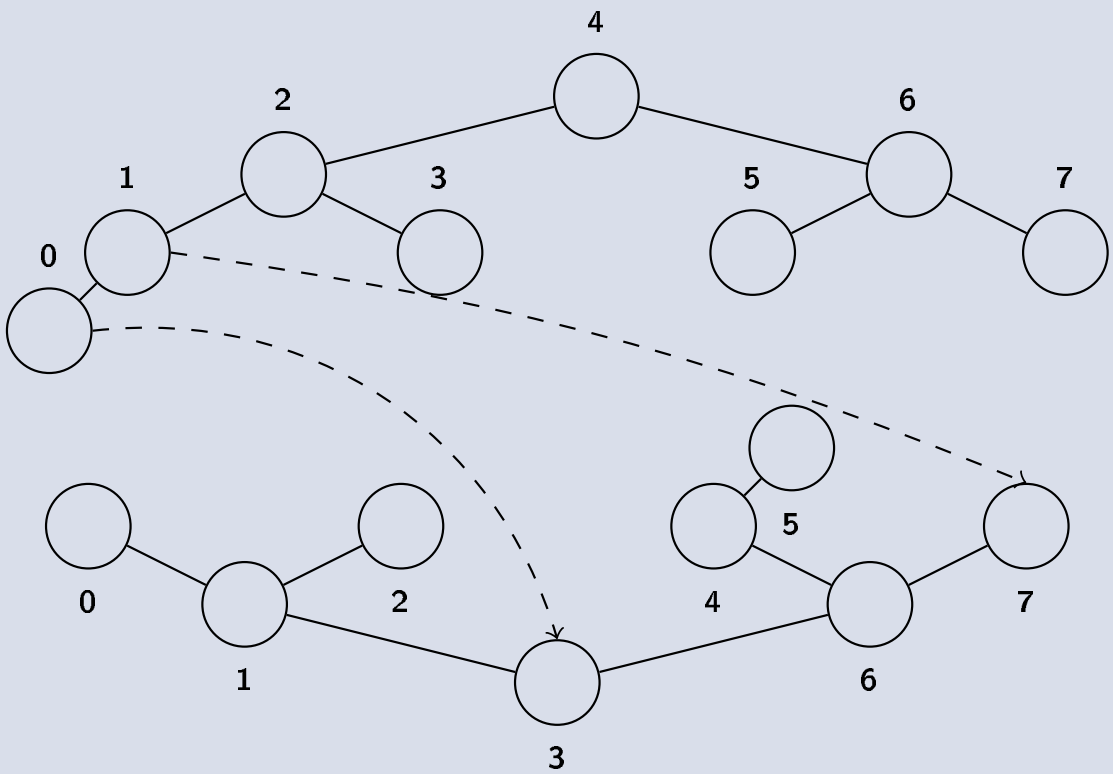
Storage of *ISA*



Original

i	ISA
0	3
1	7
2	5
3	2
4	6
5	4
6	1
7	0

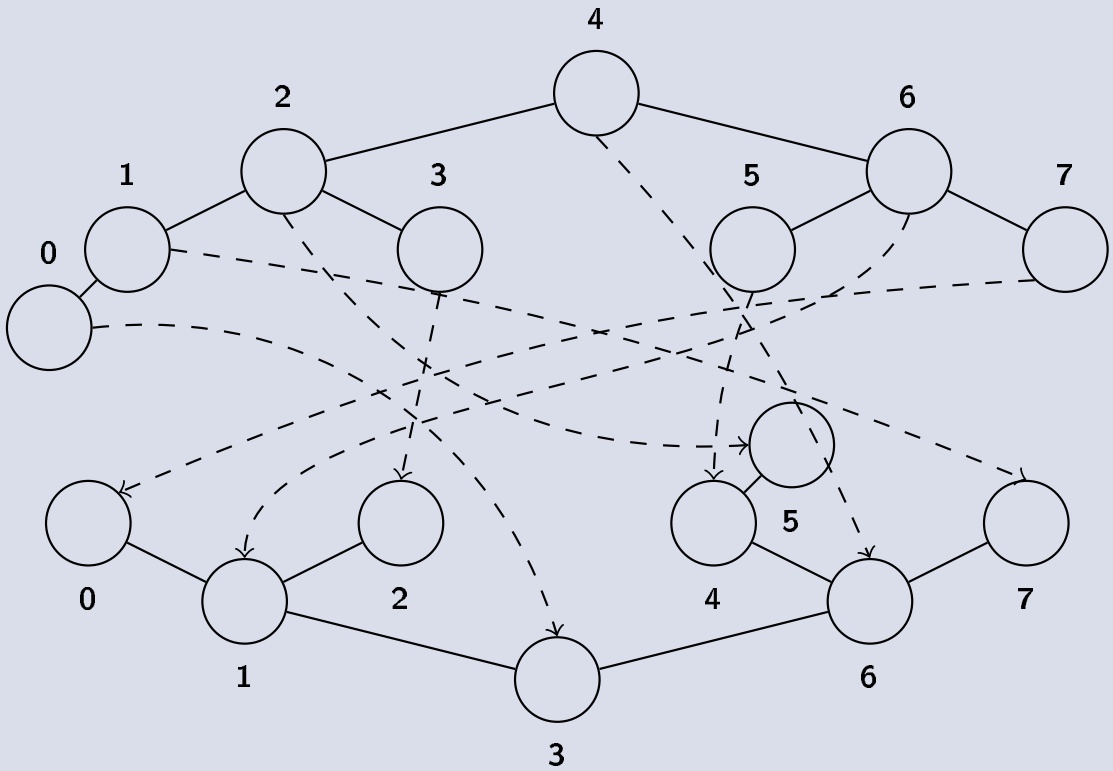
Storage of *ISA*



Original

i	ISA
0	3
1	7
2	5
3	2
4	6
5	4
6	1
7	0

Storage of *ISA*



Experiments

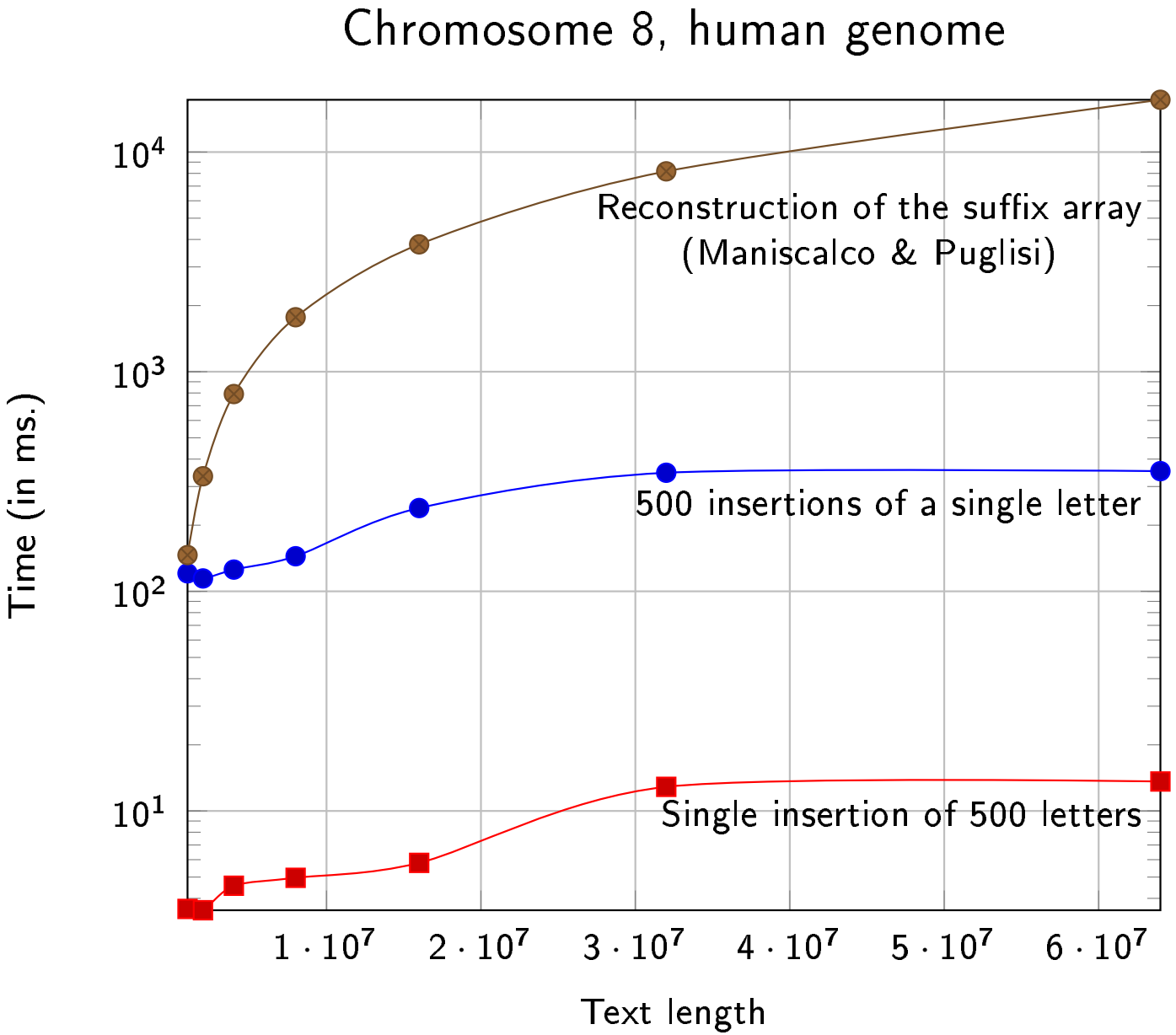
Remarks

- We can handle deletions and substitutions as well.
- We can extend our approach to factors.
- Structures are dynamic and suffer from a logarithmic factor in comparison to static structures.
- The reordering stage of the algorithm may be time-consuming.

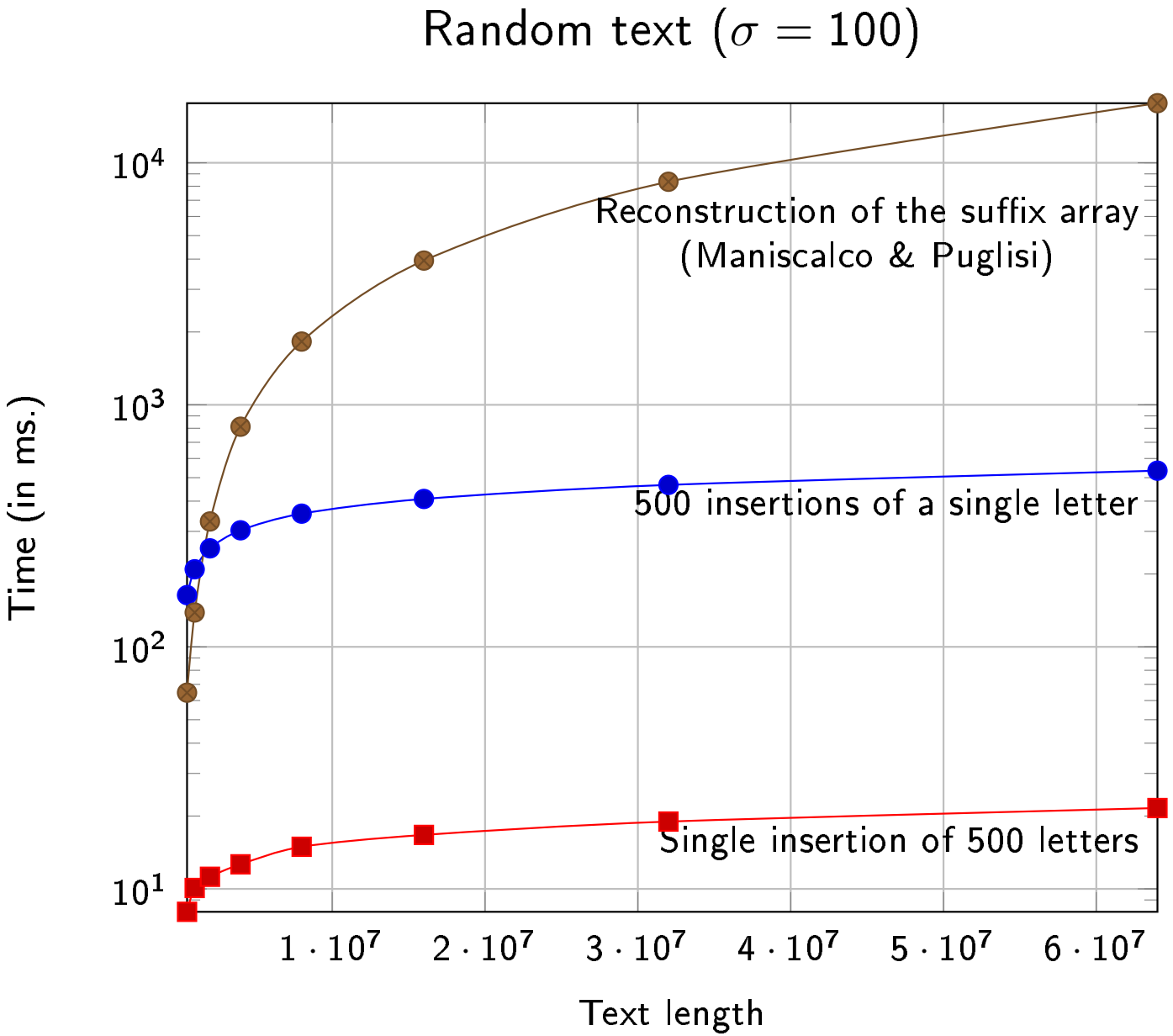
Question

Is the algorithm efficient in practice?

Experiments



Experiments



About the time complexity...

Reminder

In the worst-case, on very specific texts, we may have $O(n)$ elements to reorder.

Remarks

The previous experiments have been conducted on several thousands random examples on a given text.

It's not sufficient to conclude our algorithm is efficient for those types of text.

$\rightarrow$ Let's go back to the general case.

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof

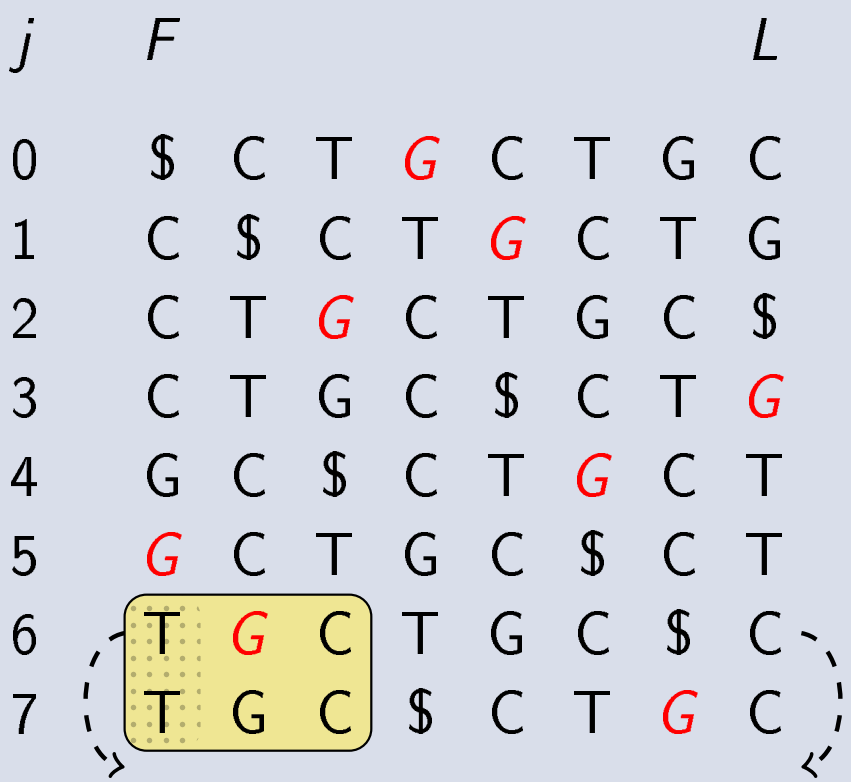
j	F							L
0	\$	C	T	*G*	C	T	G	C
1	C	\$	C	T	*G*	C	T	G
2	C	T	*G*	C	T	G	C	\$
3	C	T	G	C	\$	C	T	*G*
4	G	C	\$	C	T	*G*	C	T
5	*G*	C	T	G	C	\$	C	T
6	T	*G*	C	T	G	C	\$	C
7	T	G	C	\$	C	T	*G*	C

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof



$j = 6,\ i = 1$

$LCP[6] = 3$

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof

j	F							L
0	\$	C	T	*G*	C	T	G	C
1	C	\$	C	T	*G*	C	T	G
2	C	T	*G*	C	T	G	C	\$
3	C	T	G	C	\$	C	T	*G*
4	G	C	\$	C	T	*G*	C	T
5	*G*	C	T	G	C	\$	C	T
6	T	G	C	\$	C	T	*G*	C
7	T	*G*	C	T	G	C	\$	C

$j = 6,\ i = 1$

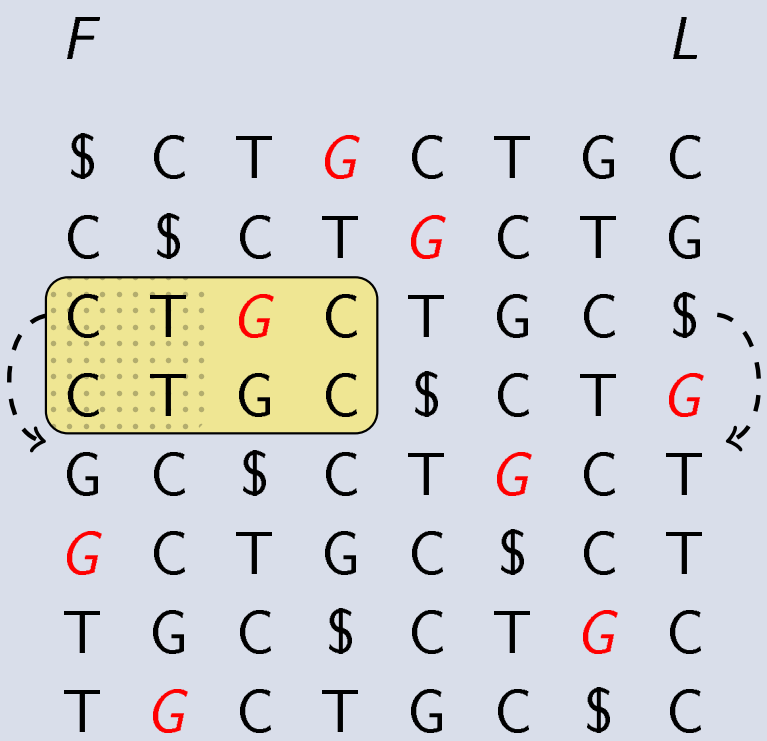
$LCP[6] = 3$

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof



$j = 2,\ i = 2$

$LCP[2] = 4$

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof

j	F							L
0	\$	C	T	*G*	C	T	G	C
1	C	\$	C	T	*G*	C	T	G
2	C	T	G	C	\$	C	T	*G*
3	C	T	*G*	C	T	G	C	\$
4	G	C	\$	C	T	*G*	C	T
5	*G*	C	T	G	C	\$	C	T
6	T	G	C	\$	C	T	*G*	C
7	T	*G*	C	T	G	C	\$	C

$j = 2,\ i = 2$

$LCP[2] = 4$

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Sketch of the proof

j	F							L
0	$	C	T	G	C	T	G	C
1	C	$	C	T	G	C	T	G
2	C	T	G	C	$	C	T	G
3	C	T	G	C	T	G	C	$
4	G	C	$	C	T	G	C	T
5	G	C	T	G	C	$	C	T
6	T	G	C	$	C	T	G	C
7	T	G	C	T	G	C	$	C

$j = 0,\ i = 3$

$LCP[0] = 0$

About the time complexity...

Property

If the element at position j in the table is the i-th one to be moved down, $LCP[j] \geq i$.

Remarks

- The *median LCP* value is an upper-bound of the number of reorderings in 50% of the cases.
- Let LCP_{ave} and LCP_{max} be the *average* and the *maximal LCP* values.

	Our algorithm's upper bounds
Average case	$LCP_{\text{ave}} \times \log n(1 + \log \sigma / \log \log n)$
Worst case	$LCP_{\text{max}} \times \log n(1 + \log \sigma / \log \log n)$

LCP values

Human genome

Texts	Size	Median	Average	Max
chr1	226,212,984	14	40	67,631
chr2	237,898,220	14	26	43,034
chr8	143,330,736	14	74	124,099
chr10	131,738,012	14	32	30,751
chr14	88,290,585	13	16	1,292
chr15	81,926,261	13	40	25,713
chr17	79,617,833	13	29	15,692
chr19	56,037,509	13	21	3,412
chr20	59,505,253	13	15	866
chr22	35,058,650	12	19	2,331
chrX	152,577,922	14	52	51,821
chrY	25,652,954	13	98	11,501

LCP values

Natural languages

Texts	Size	Median	Average	Max
Gutenberg	91,349,446	11	14	2,971
Wikipedia (afrikaans)	68,989,658	13	66	34,205
Wikipedia (occitan)	70,250,160	17	64	11,003

Conclusion

Result

Efficient update for suffix arrays and FM-Index.

Publications

M. Salson, T. Lecroq, M. Léonard, and L. Mouchard (2009). Dynamic extended suffix arrays. *JDA*. doi:10.1016/j.jda.2009.02.007.

Current work

Compressed bit vectors for improving the overall space consumption of the resulting index (Mäkinen and Navarro 2008).

Perspectives

LZ factorization

Indexing large sequences subject to local modifications.