Simple Linear-Time Off-Line Text Compression by Longest-First Substitution

Ryosuke Nakamura, Hideo Bannai,
Shunsuke Inenaga, Masayuki Takeda

Kyushu University, Japan

Lossless Compression

Uncompressed

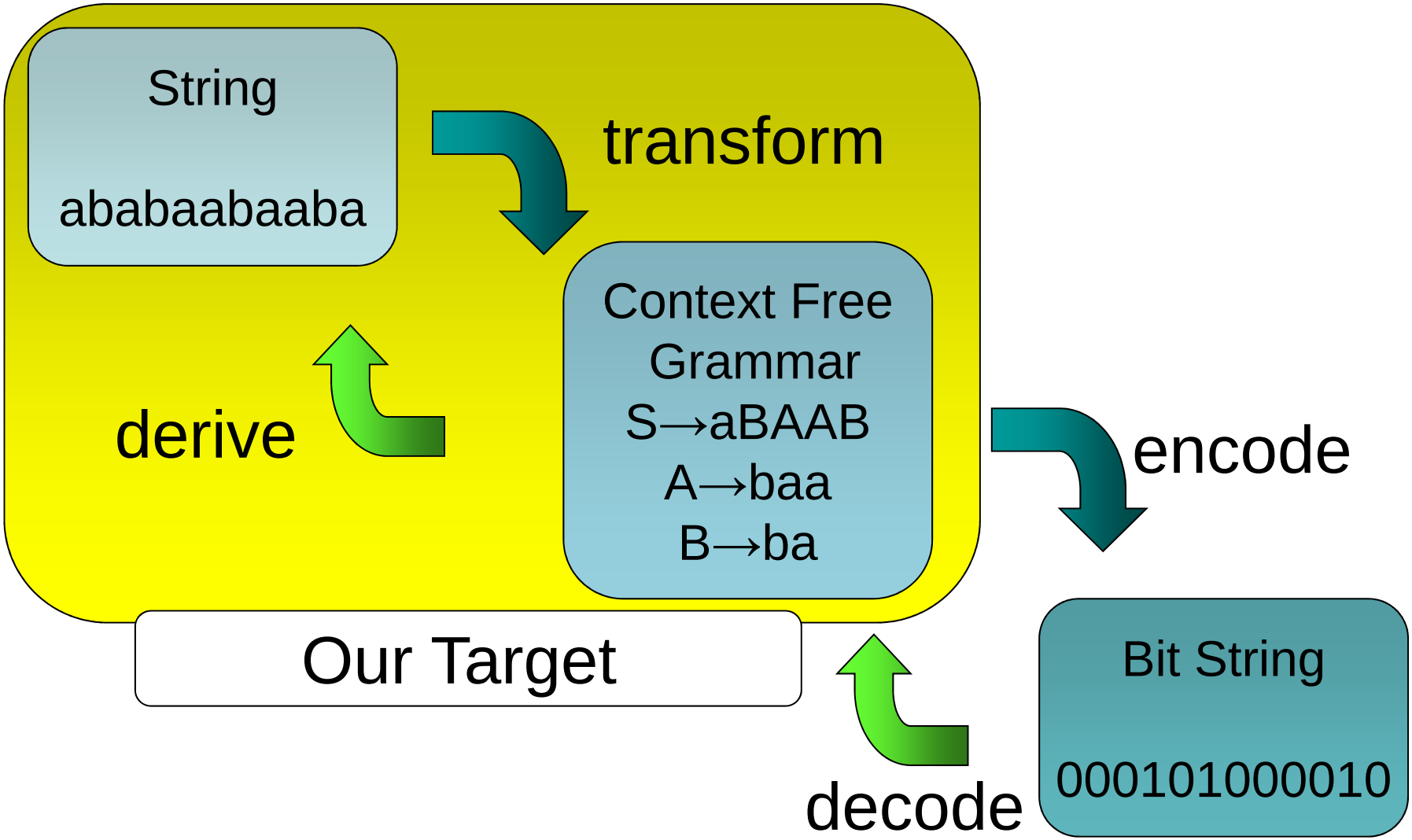
Compressed

Lossless Data Compression

- Dictionary-based Compression
  - LZ77 [Ziv and Lemple, ‘77]
- BW Transform-based Compression [Burrows and Wheeler, ‘94]
  - Bzip
- Grammar-based Compression
  - SEQUITUR [Nevill-Manning and Witten ‘97]
  - RE-PAIR [Larrson and Moffat ‘00]

Fast compressed pattern matching [Kida et al. ‘03]

Grammar-based Compression



Grammar-based Compression

- There exist more than one grammar generating the same string.
- Input string : aaaaaaaabaaaaaaaa

$S \rightarrow aaaaaaaabaaaaaaaa$

$S \rightarrow AbA,\ A \rightarrow BB,\ B \rightarrow aaaa$

$S \rightarrow AAbAA,\ A \rightarrow aaaa$

$S \rightarrow AAAAbAAAA,\ A \rightarrow aa$

- We want a small grammar.

Grammar-based Compression

- There exist more than one grammar generating the same string
- Input string: a

S → [illegible]

S → AAb[illegible]

S → AAAAAbAA[illegible]A, A → [illegible]a

Finding the smallest grammar is NP-hard [Storer '77]

- We want a small grammar.

Greedy Method

- Recursively replaces frequent and/or long factors by non-terminal symbols.

  Input string: aaaaaaaabaaaaaaaa

$$\begin{cases} S \rightarrow XXXXbXXXX \\ X \rightarrow aa \end{cases}$$

  Replaced string : XXXXbXXXX

$$\begin{cases} S \rightarrow YbY \\ X \rightarrow aa \\ Y \rightarrow XXXX \end{cases}$$

Greedy Method

- Most Frequent First Substitution (MFFS) recursively replaces the most frequent factors by non-terminal symbols

- Longest First Substitution (LFS) recursively replaces the longest repeating factors by non-terminal symbols

Greedy Method

- Most Frequent First Substitution (MFFS) recursively replaces the most frequent factors by non-terminal symbols

RE-PAIR : linear time algorithm for MFFS
[Larsson and Moffat ‘00]

- Longest First Substitution (LFS) recursively replaces the longest repeating factors by non-terminal symbols

Text Compression by Longest First Substitution [Bentley and McIlroy '99]

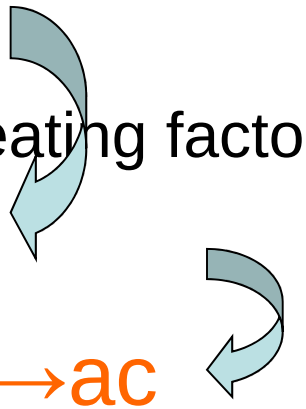
- Recursively replaces *L*ongest *R*epeating *F*actors (*LRFs*) by non-terminal symbols

S → acdaabcacabdcaababdcaad

aab ac abdcaa ← longest repeating factor

S → acdaabcacXbXd, X→abdcaa

S → YdaabcYXbXd, X→abdcaa, Y→ac



Time Complexities of Longest First Substitution Approach

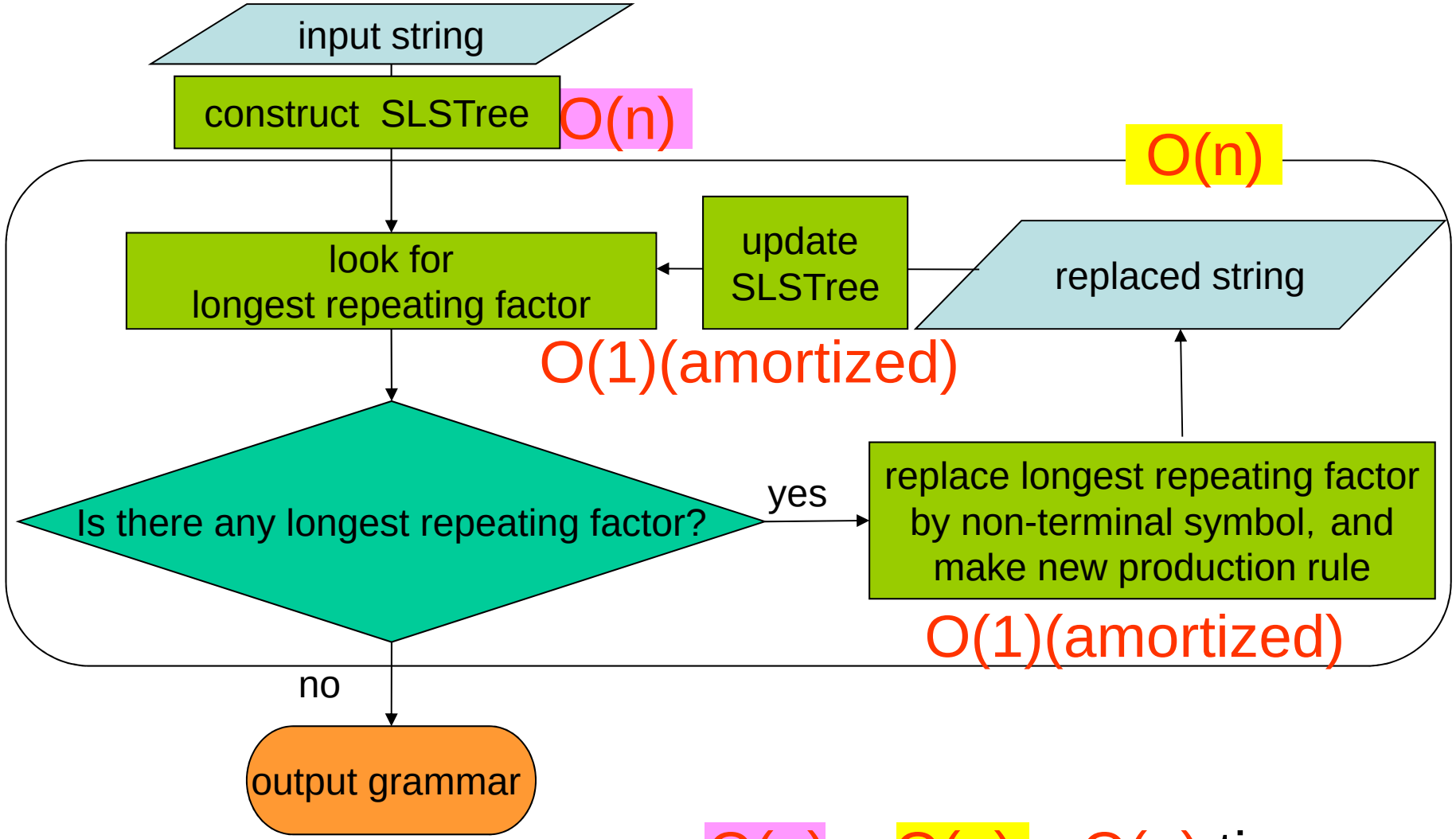
- n : length of input string
  - naive algorithm : $O(n^4)$ time ☹
  - algorithm using minimal augmented suffix trees [Brodal et al. '02] : $O(n^2 \log n)$: time 😐
  - our algorithm : $O(n)$ time ☺

Our algorithm

- Uses a new data structure sparse lazy suffix trees (SLSTrees) based on suffix trees [Weiner ’73];

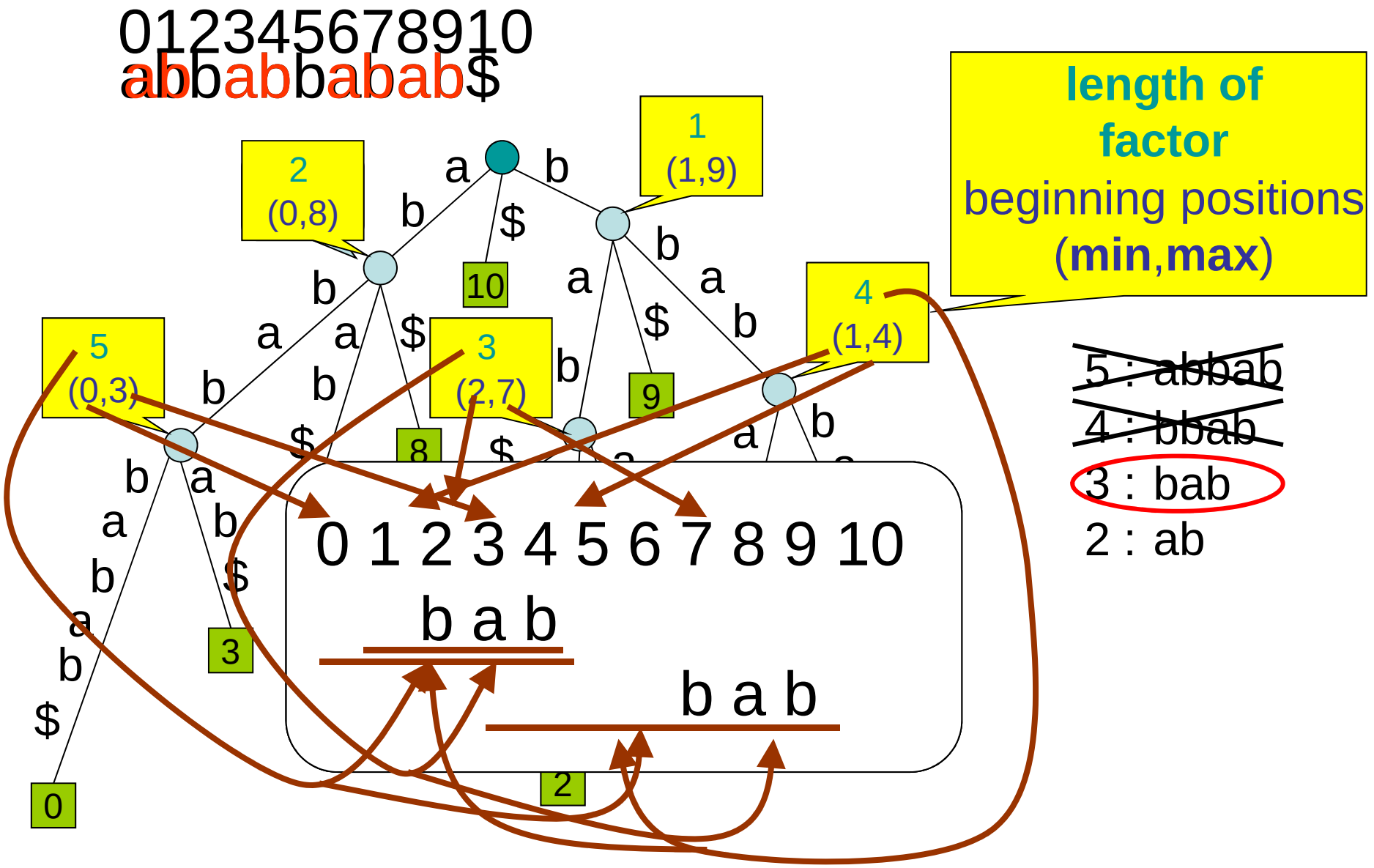
- Runs in linear time and space.

Algorithm Flow Chart

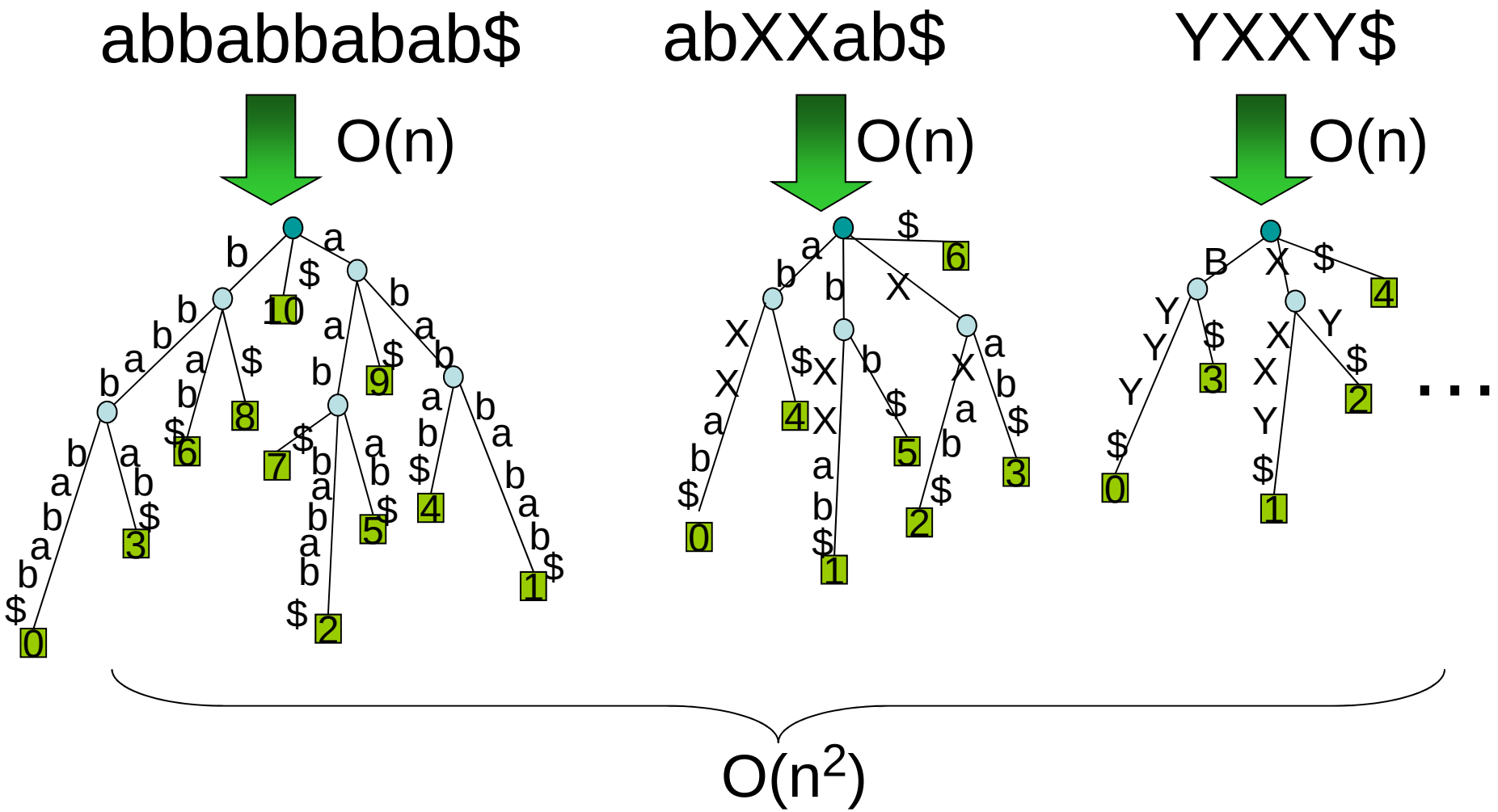


O(n) + O(n) = O(n) time

How to find Longest Repeating Factor

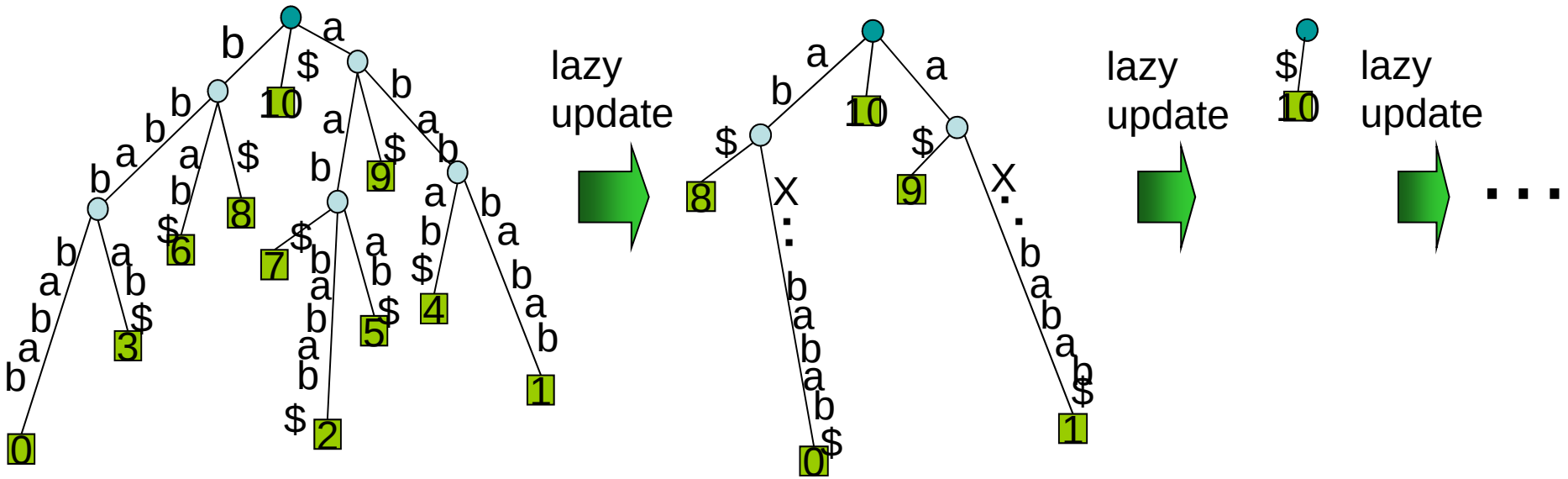


If we use suffix tree…



Using sparse lazy suffix tree

abbabbabab$ abXXab$ YXXY$

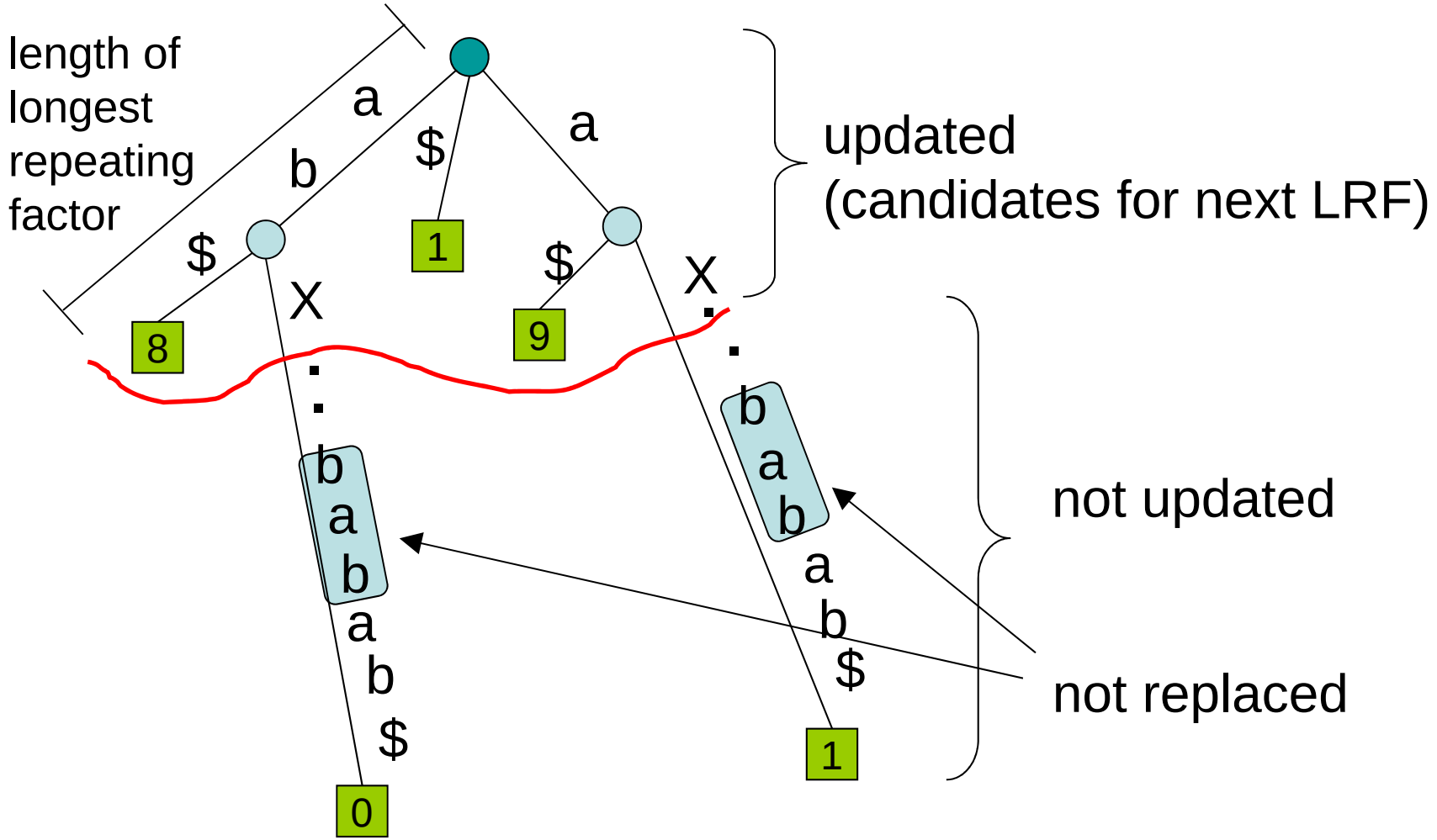


O(n)

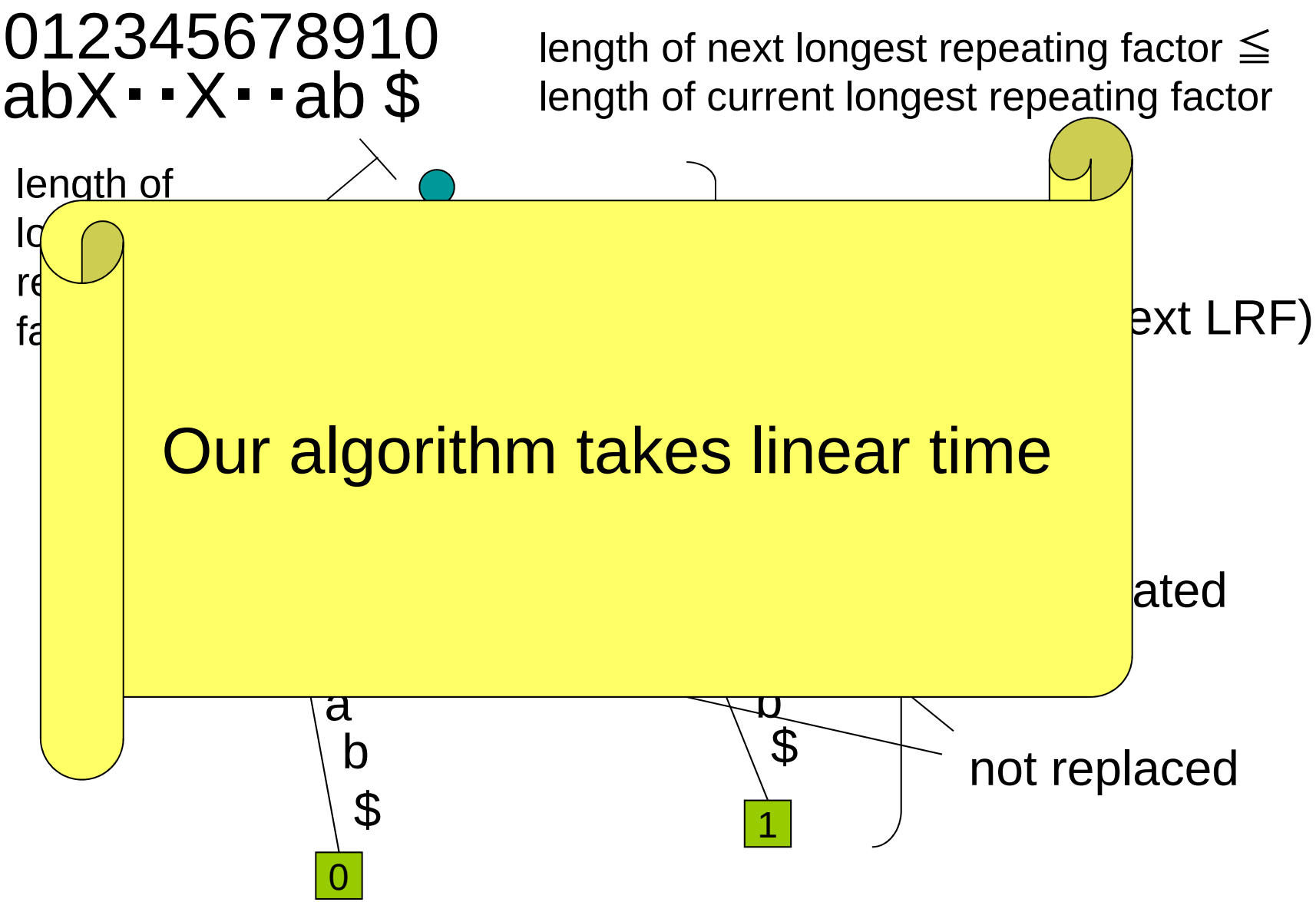
What is laziness?

012345678910
abX▪▪X▪▪ab $

length of next longest repeating factor ≦
length of current longest repeating factor



What is laziness?



Smaller grammar with Longest First Substitution (LFS2)

- Consider the right side of production rules constructed by the LFS method.

abbabbbabbbbbabbbab$

$$\begin{cases} S \rightarrow XYbYX\$ \\ X \rightarrow abbab \\ Y \rightarrow bbab \end{cases}$$

want to replace with Y

$$\begin{cases} S \rightarrow Xbbabbbbbab X\$ \\ X \rightarrow abbab \end{cases}$$

$Xbbabbbbbab X\$_1 bbab\$_2$

$$\begin{cases} S \rightarrow XYbYX\$ \\ X \rightarrow aY \\ Y \rightarrow bbab \end{cases}$$

Smaller grammar with Longest First Substitution (LFS2)

- Consider the right side of production rules constructed by the LFS method.

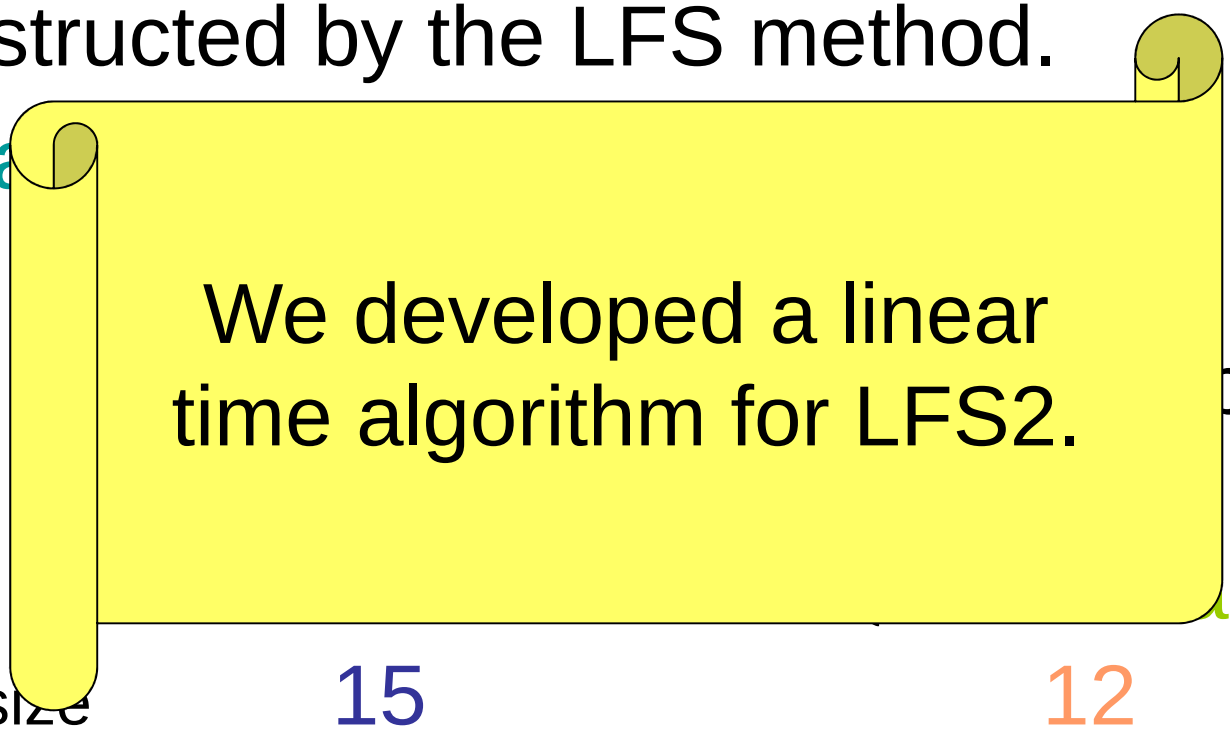
  abbabbbabbbbbababbab$

	LFS	LFS2
	$S \rightarrow XYbYX\$$ $X \rightarrow abbab$ $Y \rightarrow bbab$	$S \rightarrow XYbYX\$$ $X \rightarrow aY$ $Y \rightarrow bbab$
grammar size	15	12

grammar size : total length of the right side of production rules.

Smaller grammar with Longest First Substitution (LFS2)

- Consider the right side of production rules constructed by the LFS method.



grammar size : total length of the right side of production rules.

Comparison of grammar sizes

- Input texts : Canterbury Corpus

File	Size (Bytes)	total grammar size		
		MFFS	LFS	LFS2
alice29.txt	152090	**38750**	88333	45225
asyoulik.txt	125179	**35245**	74747	41755
cp.html	24603	8006	14559	**7977**
fields.c	11150	3535	6525	**3307**
grammar.lsp	3721	1597	2332	**1431**
kennedy.xls	1029744	**165589**	291536	166250
lcet10.txt	426754	**84923**	235112	103602
plrabn12.txt	481861	**116128**	276714	144078
ptt5	513216	**42813**	266040	47885
sum	38240	13023	20846	**12103**
xargs.1	4227	2096	2772	**1906**

Conclusions

- We developed
  - a linear time algorithm for longest first substitution (LFS);
  - a linear time algorithm for LFS2.
- LFS2 generates smaller grammars than MFFS for small files.

Future Work

- Efficient encoding of LFS grammars to bit strings.

- Compressed pattern matching specialized for LFS compressed texts.
  - The right side of most production rules consists only of terminal symbols (not LFS2). e.g., S → YdaabcYXbXd, X→abdcaa, Y→ac
    There should be efficient pattern matching algorithms for this.