# Computing LZ78-Derivates with Suffix Trees

Dominik Köppl

Faculty of Engineering, University of Yamanashi

(1,b) (1,2) (1,a)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,b)(1,b)(1,2)(1,a)

# setting

## text factorization

- input: text $T$ with length $n$
- output: factorization of $T$

examples of factorizations

- LZ77
- LZ78
- Lyndon factorization

goal: compute factorization in $\mathcal{O}(n)$ time

## substring compression

- index $T$ in a preprocessing step
- query: interval $[i..j] \subset [1..n]$
- output: factorization of $T[i..j]$

goal:

- query time linear to output size (output sensitive)
- index time linear in input size ($\mathcal{O}(n)$ time)

## why restricting index time?

trivial solution for substring compression:

- compute and store the factorizations of all $\Theta(n^2)$ substrings
- answer a query in $\mathcal{O}(1)$ via lookup
- however: index space is $\Omega(n^2)$ (hence time is also $\Omega(n^2)$)

# work on substring factorization

| factorization | construction time | query time | reference |
|---|---|---|---|
| LZ77 | $\mathcal{O}(n \lg n)$ | $\mathcal{O}(z \lg n \lg \lg n)$ | Cormode+'05 |
| LZ77 | $\mathcal{O}(n \lg n)$ | $\mathcal{O}(z \lg \lg n)$ | Keller+'14 |
| Lyndon | $\mathcal{O}(n \lg n)$ | $\mathcal{O}(z)$ | Babenko+'14 |
| Lyndon | $\mathcal{O}(n)$ | $\mathcal{O}(z)$ | Kociumaka'16 |
| LZ78 | $\mathcal{O}(n)$ | $\mathcal{O}(z)$ | Köppl'21 |
| LZD/LZMW | $\mathcal{O}(n)$ | $\mathcal{O}(z)$ | this talk |

$z$ : output size of respective factorization

# factorizations in this talk

LZ78 derivations

- Lempel–Ziv Double (LZD) Goto'15
- Lempel–Ziv-Miller–Wegman (LZMW) Miller+'85

why?

- number of LZ78 factors is lower bounded by $\Omega(\sqrt{n})$
- in contrast, the lower bound for LZD and LZMW is $\Omega(\lg n)$

# lower bound for LZ78

$T = \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a}\ \mathtt{a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZ78

$$T = \texttt{a a a a a a a a a a a a} \cdots$$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

coding: a

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZ78

(1,a)

$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a(1,a)

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZ78

(1,a)

(2,a)

$T =$ `a a a a a a a a a a a a` $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: `a(1,a)(2,a)`

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZ78

(1,a)

(2,a)

(3,a)

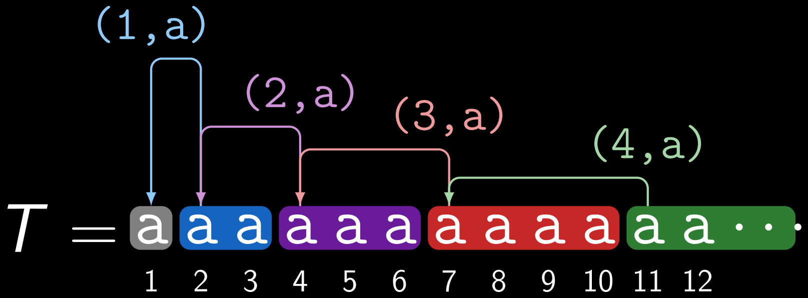
$T = \texttt{a a a a a a a a a a a a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a(1,a)(2,a)(3,a)

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZ78




coding: a(1,a)(2,a)(3,a)(4,a)

since the length $|F_x|$ of the $x$-th factor is $x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

# lower bound for LZD

$$T = \texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\cdots$$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

since the length $|F_x|$ of the $x$-th factor is $2^x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZD

$T = \texttt{a a a a a a a a a a a a} \cdots$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|

coding: (a,a)

since the length $|F_x|$ of the $x$-th factor is $2^x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZD

`(1,1)`

$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,a) `(1,1)`

since the length $|F_x|$ of the $x$-th factor is $2^x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZD

(1,1)

(2,2)

$T =$ a a a a a a a a a a a a ⋯

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,a) (1,1) (2,2)

since the length $|F_x|$ of the $x$-th factor is $2^x$, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

## lower bound for LZMW

$$T = \texttt{a a a a a a a a a a a a} \cdots$$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

since the length $|F_x|$ of the $x$-th factor is the $x-1$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZMW

$T = $ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a

since the length $|F_x|$ of the $x$-th factor is the $x - 1$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZMW

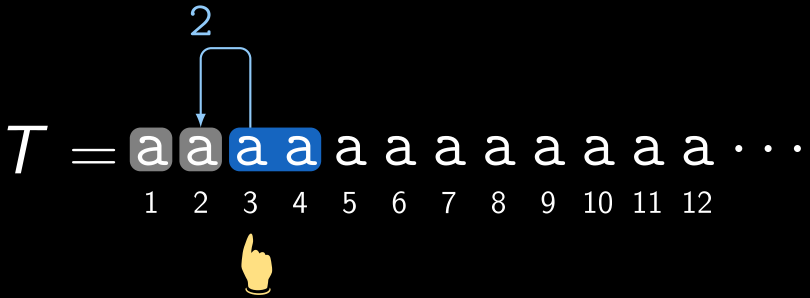
$T =$ a a a a a a a a a a a a $\cdots$

| a | a | a | a | a | a | a | a | a | a | a | a |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

coding: aa

since the length $|F_x|$ of the $x$-th factor is the $x - 1$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

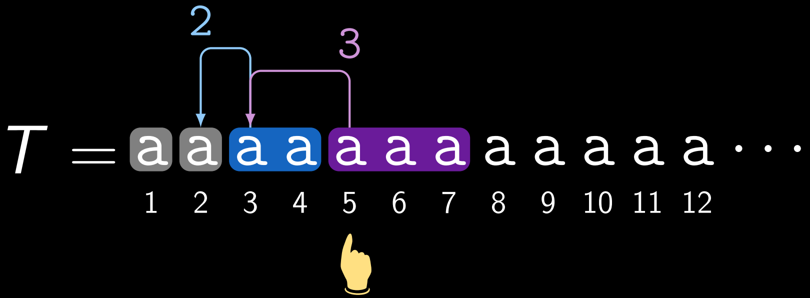
# lower bound for LZMW




coding: aa2

since the length $|F_x|$ of the $x$-th factor is the $x - 1$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZMW




coding: aa23

since the length $|F_x|$ of the $x$-th factor is the $x - 1$st Fibonacci number, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# lower bound for LZMW

$T = \texttt{a a a a a a a a a a a a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: aa234

since the length $|F_x|$ of the $x$-th factor is the $x-1$st Fibonacci number, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

# definition of LZD

each factor represented as a pair

- element is either a character or the index of a former factor
- greedily maximize the length by the first element first

let $\mathrm{dst}_x$ denote the starting position of $F_x$ in $T$.

## formal definition

A factorization $F_1 \cdots F_z$ of $T$ is LZD if

- $F_x = G_1 \cdot G_2$ with
- $G_1, G_2 \in \{F_1, \ldots, F_{x-1}\} \cup \Sigma$ such that
- $G_1$ and $G_2$ are respectively the longest possible prefixes of $T[\mathrm{dst}_x..]$ and of $T[\mathrm{dst}_x + |G_1|..]$.

## example for LZD

$T = $ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding:

## example for LZD

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,b)

# example for LZD

(1,b)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,b)(1,b)

# example for LZD

(1,b)

(1,2)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,b) (1,b) (1,2)

# example for LZD

(1,b)

(1,2)

(1,a)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,b)(1,b)(1,2)(1,a)

# definition of LZMW

- has like LZD two references
- however references need to be successive
- thus needs to store only one reference to a former factor index

## formal definition

A factorization $F_1 \cdots F_z$ of $T$ is LZMW if $F_x$ is the longest prefix of $T[\mathrm{dst}_x..]$ with $F_x \in \{F_{y-1}F_y : y \in [2..\mathrm{dst}_x - 1]\} \cup \Sigma$, for every $x \in [1..z]$.

# example for LZMW

$$T = \texttt{a b a b b a b a b b a b b}$$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

## example for LZMW

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: a

# example for LZMW

$T = $ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: ab

# example for LZMW

2

$T = \texttt{a b a b b a b a b b a b b}$

1 2 3 4 5 6 7 8 9 10 11 12

coding: ab2

# example for LZMW

2

3

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: ab23

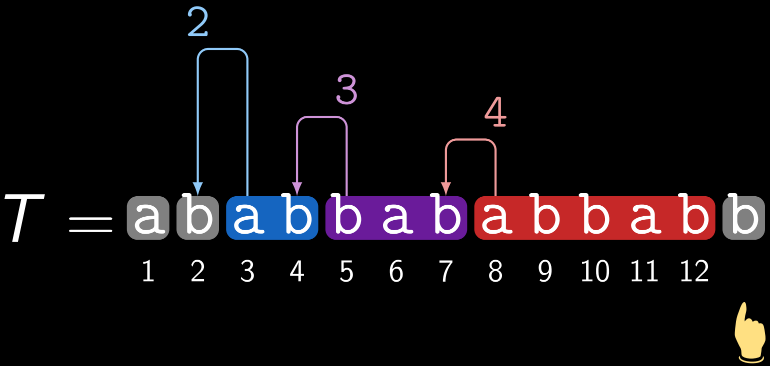
## example for LZMW

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12

coding: ab234

# example for LZMW




coding: ab234b

# LZD and LZMW computation

| time | space | reference | |
|---|---|---|---|
| $\mathcal{O}(n \lg \sigma)$ | $\mathcal{O}(n)$ | Goto+'15 | |
| $\Omega(n^{5/4})$ | $\mathcal{O}(z)$ | Goto+'15, Badkobeh+'17 | where |
| $\mathcal{O}(n + z \lg^2 n)$ expected | $\mathcal{O}(z)$ | Badkobeh+'17 | |
| $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | this talk | |

- Goto+'15 only computes LZD
- $\sigma = n^{\mathcal{O}(1)}$ means that integer alphabets are supported

# our contributions

- for the whole text, we can compute LZD and LZMW in $\mathcal{O}(n)$ time and space
- compute the substring compression of LZD and LZMW with
  - $\mathcal{O}(n)$ index time for preprocessing
  - $\mathcal{O}(z)$ query time
- setting
  - $n$ : length of the input
  - integer alphabet
  - word RAM

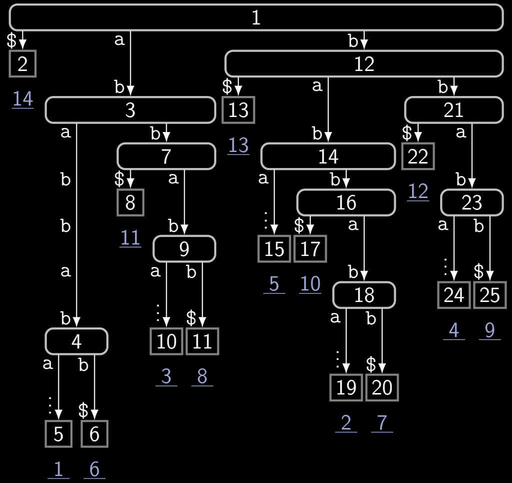
## tools

for computation, we leverage the following toolbox

- suffix tree ST Weiner'73
  - linear-time construction of ST Farach-Colton'00
- weighted ancestor query data structure Gawrychowski'14
  - find an ancestor with string depth $d$ of any ST node and any $d$ in $\mathcal{O}(1)$ time
  - constructable in linear time Belazzougui'21
- lowest marked ancestor data structure Cole+'05
  - can mark any ST node in $\mathcal{O}(1)$ time
  - can find the lowest marked ancestor of any ST node in $\mathcal{O}(1)$ time

sum of needed space and time amounts to $\mathcal{O}(n)$ each

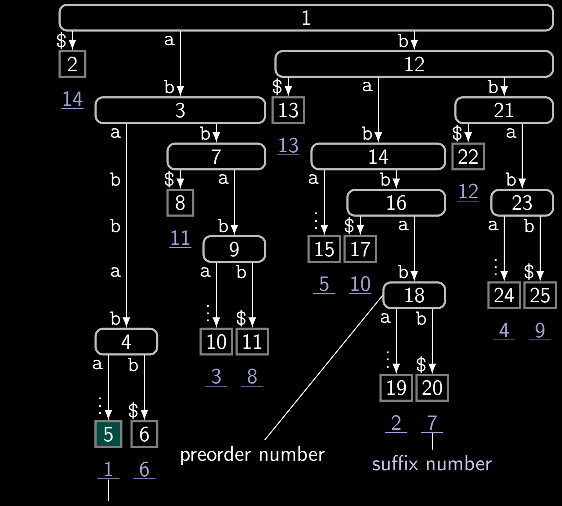
how used for LZD computation?



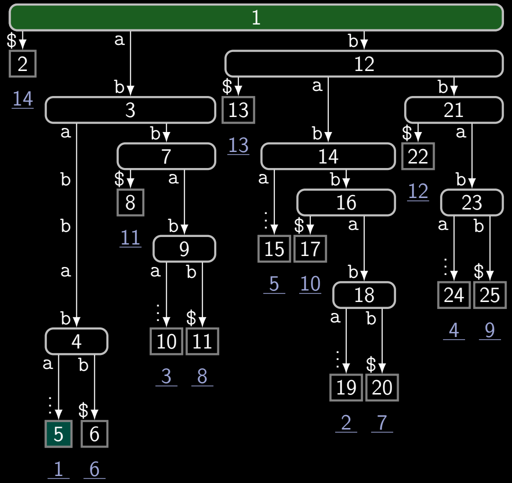

suffix tree of $T\$$ = ababbababbabb

$T$ = ababbababbabb

suffix tree of $T\$$ = ababbababbabb



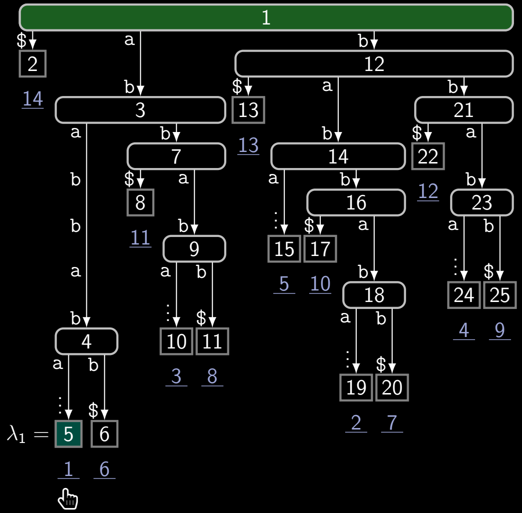

$T$ = ababbababbabb




suffix tree of $T\$ =$ ababbababbabb

- ST root represents empty factor

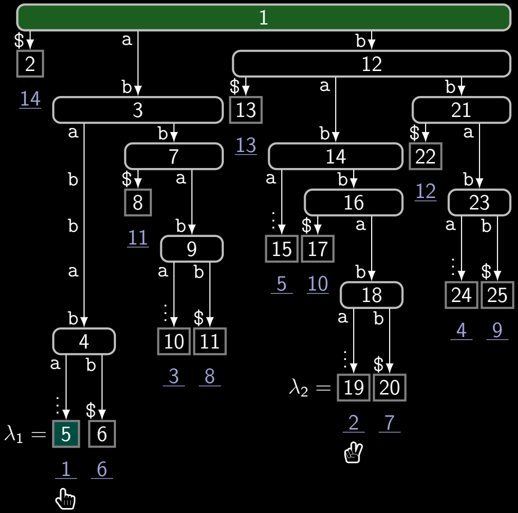
$T =$ ababbababbabb




suffix tree of $T\$ = $ ababbababbabb

- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of $\lambda_1$ is $\mathrm{dst}_1 = 1$
- lowest marked ancestor of $\lambda_1$ is ST root, so $e_L = T[1] = $ a

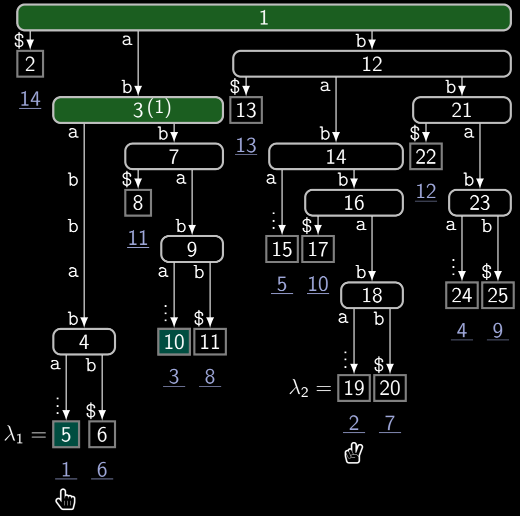
$T = $ ababbababbabb




$T = $ ababbababbabb

suffix tree of $T\$ = $ ababbababbabb

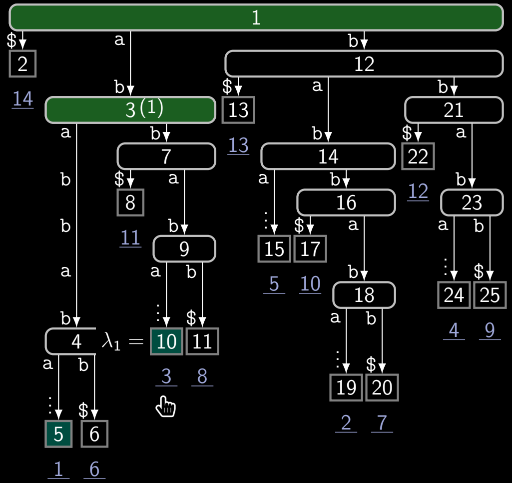
- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of $\lambda_1$ is $\mathrm{dst}_1 = 1$
- lowest marked ancestor of $\lambda_1$ is ST root, so $e_L = T[1] = $ a
- $\lambda_2$ is leaf with suffix number 2




$T = \text{ab}|\text{abbababbabb}$

suffix tree of $T\$ = \text{ababbababbabb}$

- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of $\lambda_1$ is $\text{dst}_1 = 1$
- lowest marked ancestor of $\lambda_1$ is ST root, so $e_L = T[1] = \text{a}$
- $\lambda_2$ is leaf with suffix number 2
- lowest marked ancestor of $\lambda_2$ is ST root, so $e_R = T[2] = \text{b}$
- mark ancestor of $\lambda_1$ with string depth 2 with 1

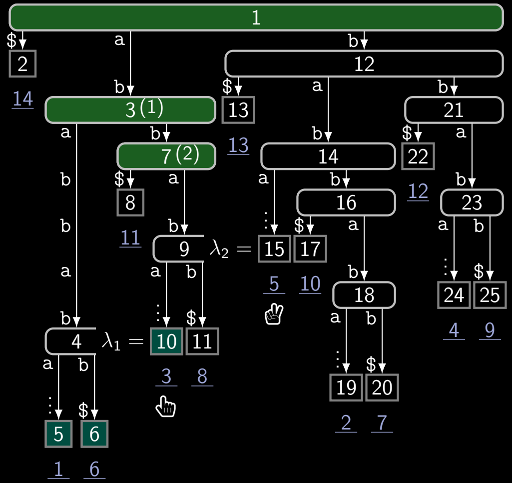



suffix tree of $T\$ = \text{ababbababbabb}$

process $F_2$

- suffix number of $\lambda_1$ is $\mathrm{dst}_2 = 3$
- lowest marked ancestor of $\lambda_1$ is 3, so $e_L = 1$ (mark of 3)

$T = \text{ab}|\text{abbababbabb}$

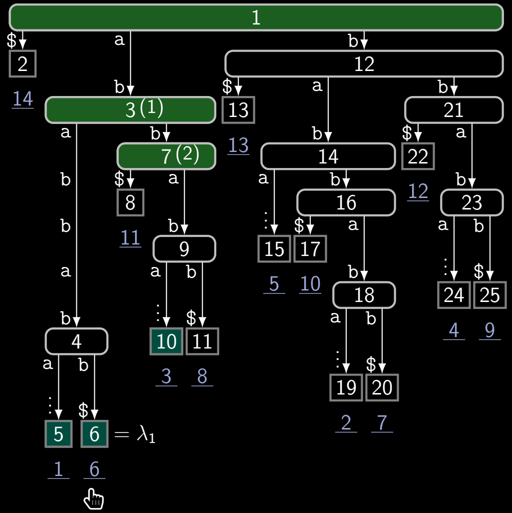



suffix tree of $T\$ = $ ababbabababbabb

process $F_2$

- suffix number of $\lambda_1$ is $\mathrm{dst}_2 = 3$
- lowest marked ancestor of $\lambda_1$ is 3, so $e_L = 1$ (mark of 3)
- like before, $e_R = T[2] = \mathrm{b}$
- mark ancestor of $\lambda_1$ with string depth $|F_2| = 3$ with 2

$T = \mathrm{ab}|\mathrm{abb}|\mathrm{ababbabb}$

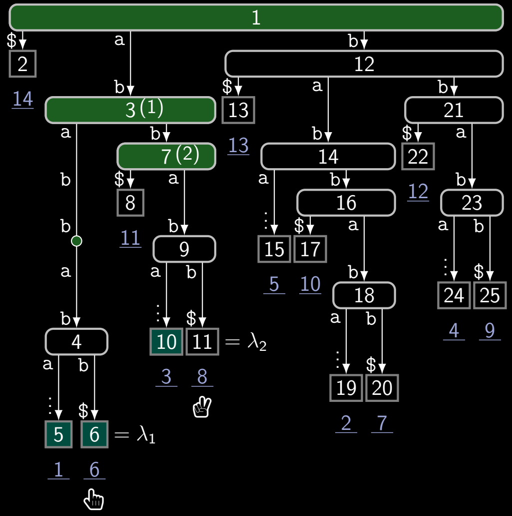



suffix tree of $T\$ = $ ababbababbabb

process $F_3$

- suffix number of $\lambda_1$ is $\mathrm{dst}_3 = 6$
- lowest marked ancestor of $\lambda_1$ is 3, so $e_L = 1$ (mark of 3)

$T = $ ab|abb|ababbabb

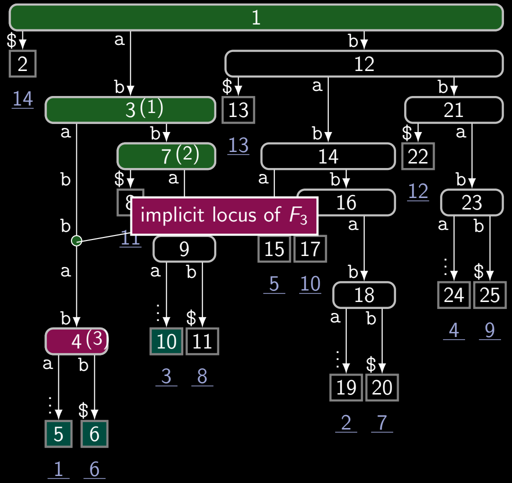



suffix tree of $T\$ = \mathtt{ababbababbabb}$

process $F_3$

- suffix number of $\lambda_1$ is $\mathrm{dst}_3 = 6$
- lowest marked ancestor of $\lambda_1$ is 3, so $e_L = 1$ (mark of 3)
- lowest marked ancestor of $\lambda_2$ is 7, so $e_L = 2$ (mark of 2)
- however: ancestor of $\lambda_1$ with string depth $|F_3| = 5$ does not exist!

$T = \mathtt{ab}|\mathtt{abb}|\mathtt{ababb}|\mathtt{abb}$




suffix tree of $T\$ = \mathtt{ababbababbabb}$

maintaining reference for $F_3$

- locus of $F_3$ can be witnesses by node 4
- let node 4 store length of $F_3$; mark node 4

$T = \mathtt{ab|abb|ababb|abb}$

# time complexity

for processing $F_x$

- take leaf $\lambda_1$ corresponding to the starting position $\mathrm{dst}_x$ of $F_x$
- compute the lowest marked ancestor $v_1$ of $\lambda_1$
- given $\ell_1$ is the string length of $v_1$, take leaf $\lambda_2$ having suffix number $\mathrm{dst}_x + \ell_1$
- compute the lowest marked ancestor $v_2$ of $\lambda_2$
- length of $F_x$ is $\ell_1 + \ell_2$, where $\ell_2$ is the string length of $v_2$
- if $v_1$ (or $v_2$) refers to an implicit node, use the stored length instead of $\ell_1$ (or $\ell_2$)

each step takes $\mathcal{O}(1)$ time, so we have $\mathcal{O}(z)$ total time, where $z$ is the number of processed factors

# LZMW

LZMW computation works similarly

- mark the locus of $F_{x-1}F_x$ instead of $F_x$
- need only one lowest marked ancestor query ($v_2$ not needed)

## summary

- can compute LZD and LZMW in $\mathcal{O}(n)$ time, in the computational model
  - $n$ : length of the input
  - alphabet can be integer
  - word RAM

for substring compression:

- $\mathcal{O}(n)$ index time
- $\mathcal{O}(z)$ query time, where $z$ is the number of factors to output

Thank you for listening. Any questions are welcome!