Exploring Regular Structures in Strings

17. July 2018

PhD Defense of Dominik Köppl

Jury Members

- Prof. Dr. Thomas Schwentick
- Prof. Dr. Johannes Fischer
- Prof. Dr. Shunsuke Inenaga
- Prof. Dr. Sven Rahmann

managing massive text data

various types:

documents	*the web*
versioned source code	*git*
biological data	*DNA*

range: gigabyte – terabyte

managing massive text data

various types:

documents	*the web*
versioned source code	*git*
biological data	*DNA*

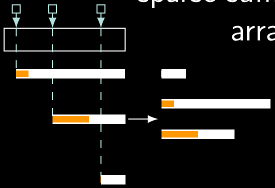
range: gigabyte – terabyte

problems

I text indexing

II text compression

III text analysis



sparse suffix array

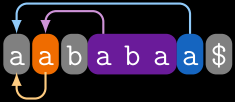
I. text data structures

sparse suffix array

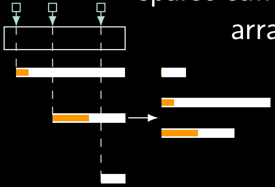
I. text data structures

compute

II. factorizations



Lempel-Ziv



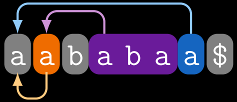
sparse suffix array

I. text data structures

compute

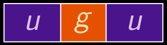
II. factorizations

find

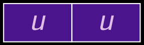


Lempel-Ziv

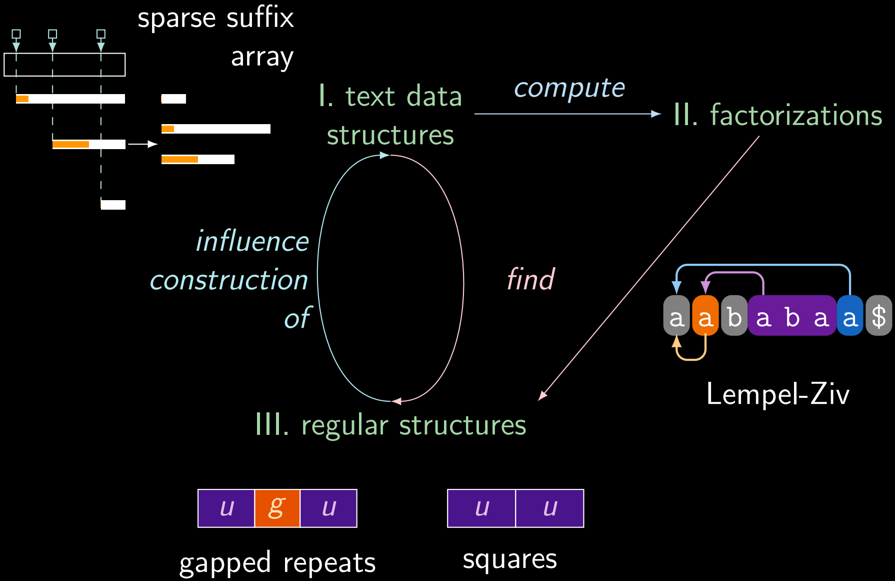
III. regular structures



gapped repeats



squares



sparse suffix array

I. text data structures

compute

II. factorizations

influence construction of

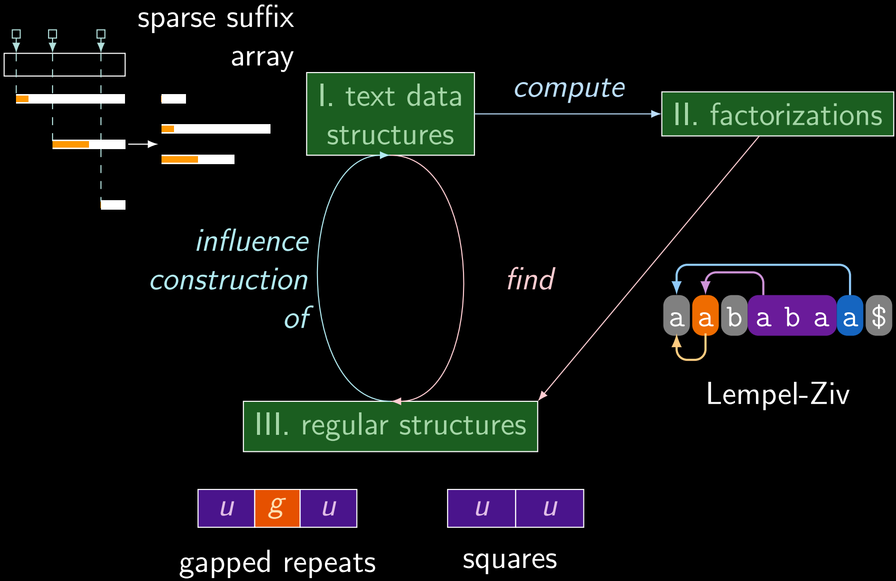
find

Lempel-Ziv

III. regular structures

gapped repeats

squares



sparse suffix array

I. text data structures

compute

II. factorizations

influence construction of

find

Lempel-Ziv

III. regular structures

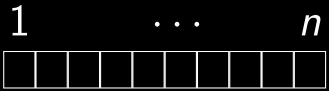
gapped repeats

squares

setting

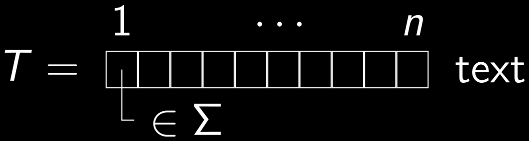
$T = \boxed{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$ text

setting

$T =$ 

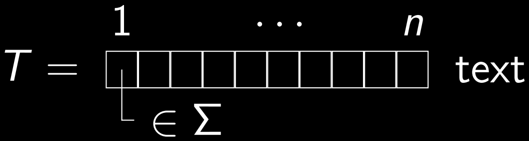
text

setting



$$\sigma := |\Sigma| = n^{\mathcal{O}(1)}$$

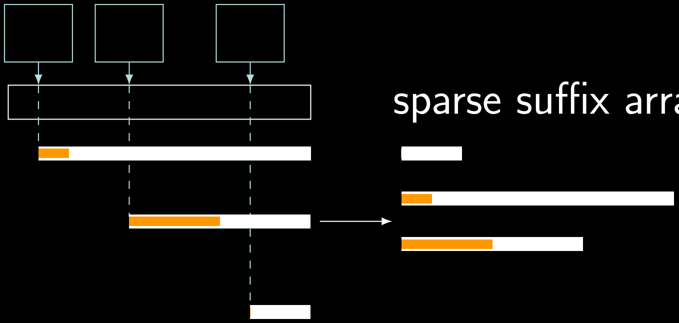
setting



$$\sigma := |\Sigma| = n^{\mathcal{O}(1)}$$

- word-RAM model
- T loaded into RAM (not measured)

I. sparse suffix sorting



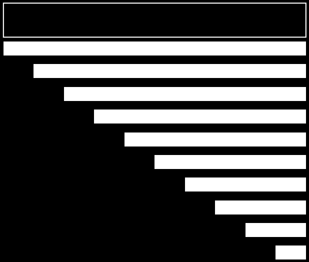
suffix sorting

$$T = \boxed{}$$

suffix sorting

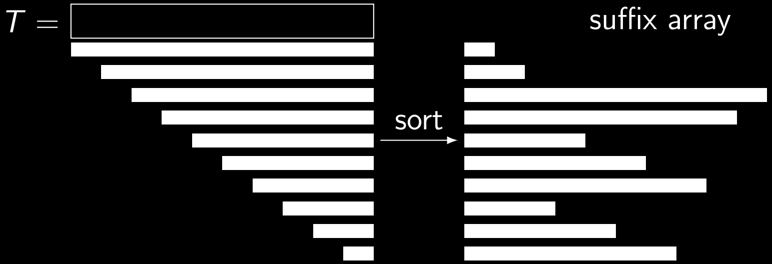
- sort *all* suffixes lexicographically

$T =$



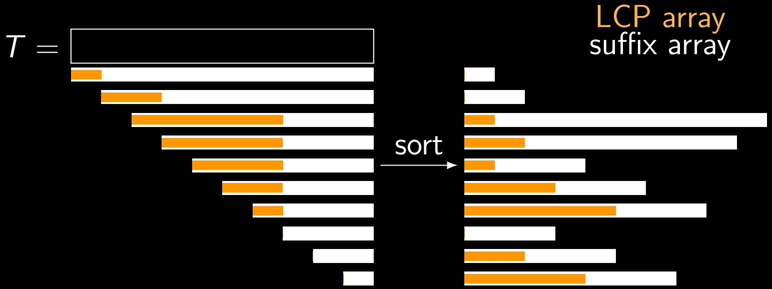
suffix sorting

- sort *all* suffixes lexicographically



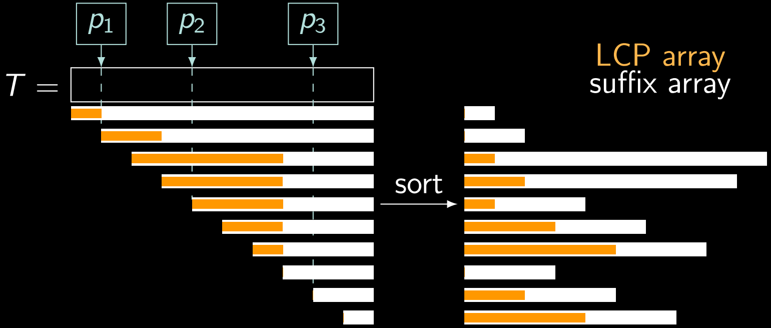
suffix sorting

- sort *all* suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $\mathcal{O}(n)$ time and words of space



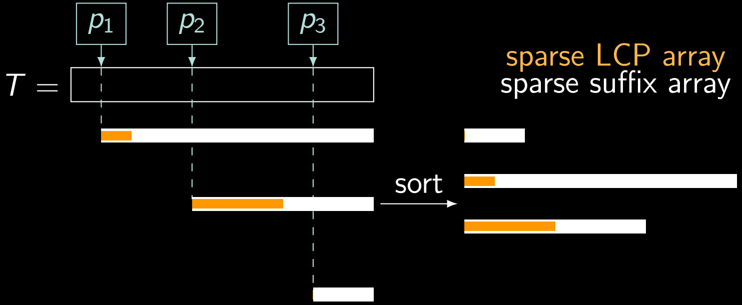
suffix sorting

- sort *all* suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $\mathcal{O}(n)$ time and words of space
- sometimes need only suffixes starting at $p_1, \ldots, p_m$



suffix sorting

- sort *all* suffixes lexicographically
- lengths of the longest common prefix (LCP) between adjacent suffixes.
- solved in $\mathcal{O}(n)$ time and words of space
- sometimes need only suffixes starting at $p_1, \ldots, p_m$

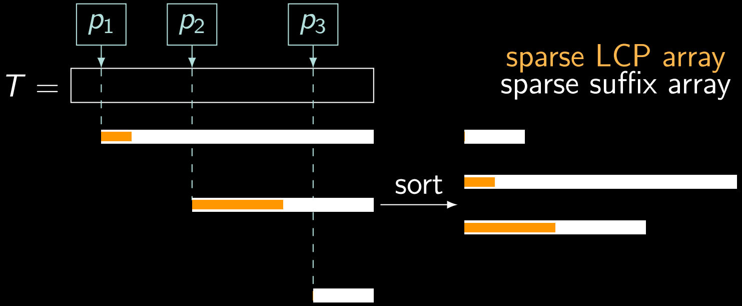


sparse suffix sorting

- set of m text positions $p_1, \ldots, p_m$
- sort suffixes starting at these positions

given text in RAM and $m = o(n)$, we want

- $o(n)$ time
- $\mathcal{O}(m) = o(n)$ space

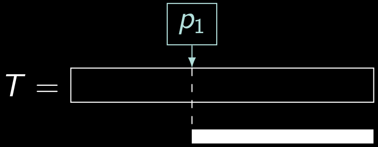


sparse suffix sorting

$T =$

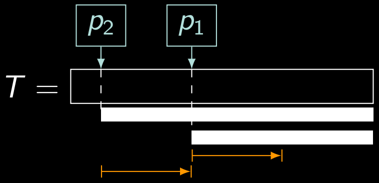
sparse suffix sorting

- $p_1, \dots, p_m$: online, arbitrary order



sparse suffix sorting

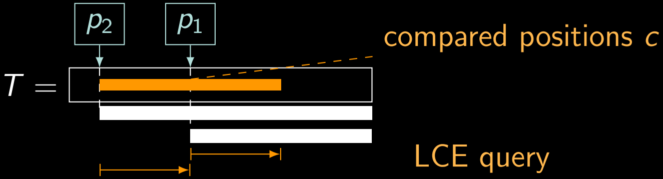
- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



LCE query

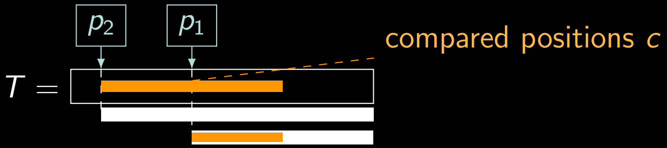
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



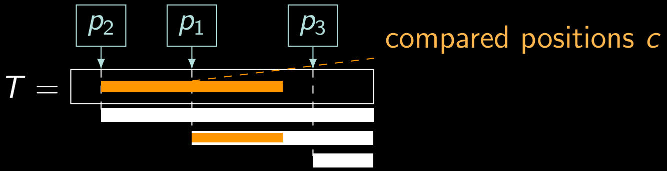
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



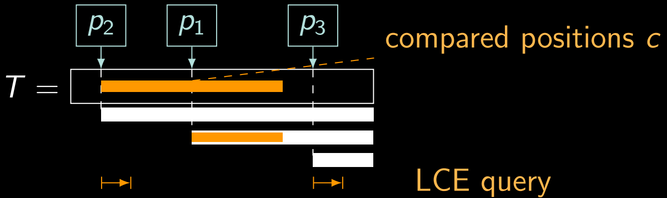
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



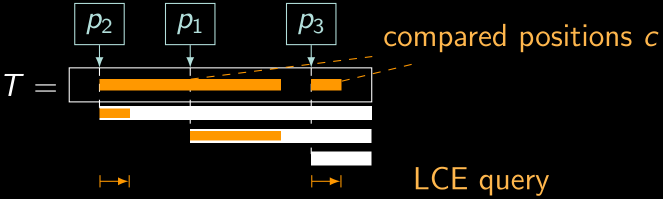
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



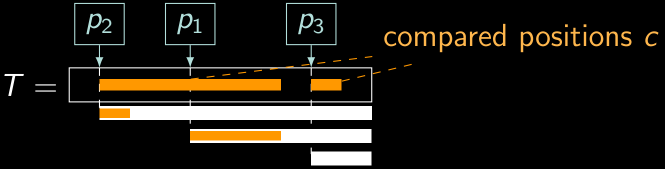
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query



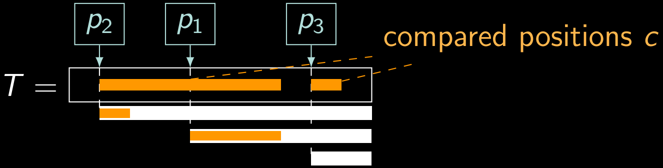
sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query
- result:
  - $\mathcal{O}(c(\sqrt{\lg \sigma} + \lg \lg n) + m \lg m \lg n \lg^* n)$ time
  - $\mathcal{O}(m)$ space
  - if text is *overwriteable*



sparse suffix sorting

- $p_1, \ldots, p_m$: online, arbitrary order
- compare two suffixes with LCE query
- result:
  - $\mathcal{O}(c(\sqrt{\lg \sigma} + \lg \lg n) + m \lg m \lg n \lg^* n)$ time
  - $\mathcal{O}(m)$ space
  - if text is *overwriteable*
  - $o(n)$ time if $c = o(n)$ and $m = o(n)$



sparse suffix sorting

time	**words**	**authors**
$\mathcal{O}(n)$	$n + \mathcal{O}(1)$	Goto'17, Li+'16
$\mathcal{O}(\frac{n^2}{m})$	$\mathcal{O}(m)$	Kärkkainen+'06
$\mathcal{O}(n)$ whp.	$\mathcal{O}(m)$	Prezza'18
$\mathcal{O}(n)$	$\mathcal{O}(m)$	open problem

our result

- $\mathcal{O}(c(\sqrt{\lg \sigma} + \lg \lg n) + m \lg m \lg n \lg^* n)$ time
- $\mathcal{O}(m)$ space

Fischer, I, Köppl
Deterministic sparse suffix sorting on rewritable texts.
In Proc. LATIN, **2016**

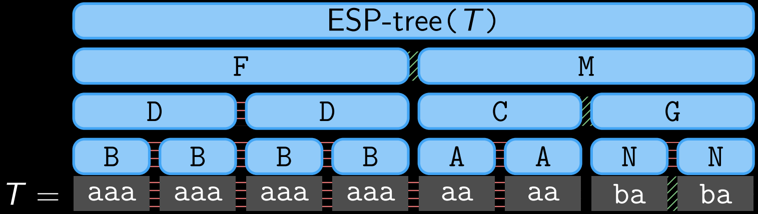
time bound

$$\mathcal{O}(\underbrace{c(\sqrt{\lg\sigma}+\lg\lg n)}_{\text{LCE construction}} + \overbrace{m\lg m}^{\text{search tree}}\ \underbrace{\lg n\lg^{*} n}_{\text{LCE queries}})$$

LCE data structure

- built on c
- is mergeable

Candidate: ESP-tree Cormode,Muthu.'07



ESP-tree(T)

F M

D D C G

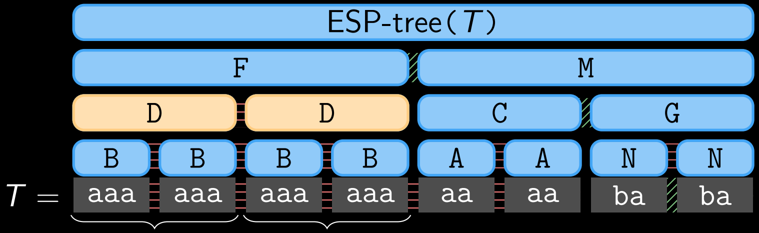
B B B B A A N N

$T =$ aaa aaa aaa aaa aa aa ba ba

Candidate: ESP-tree

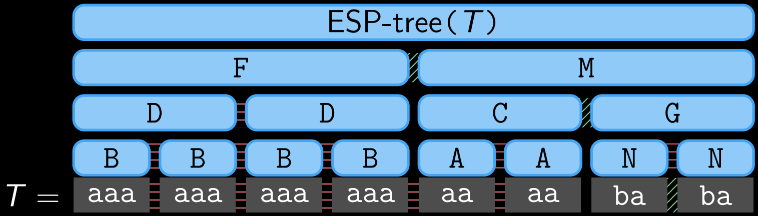
Cormode,Muthu.'07

- grammar tree



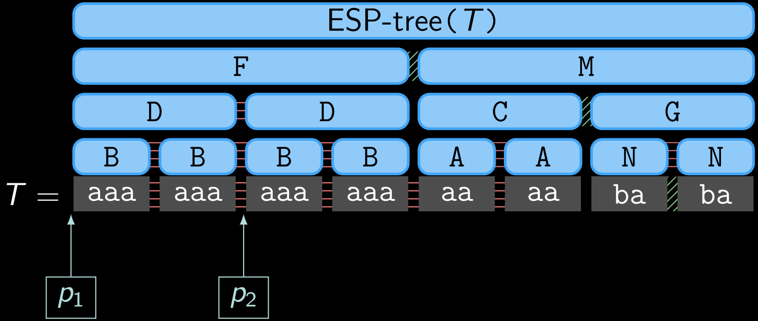
Candidate: ESP-tree Cormode,Muthu.'07

- grammar tree
- same substrings have mostly same parsing



Candidate: ESP-tree Cormode,Muthu.'07

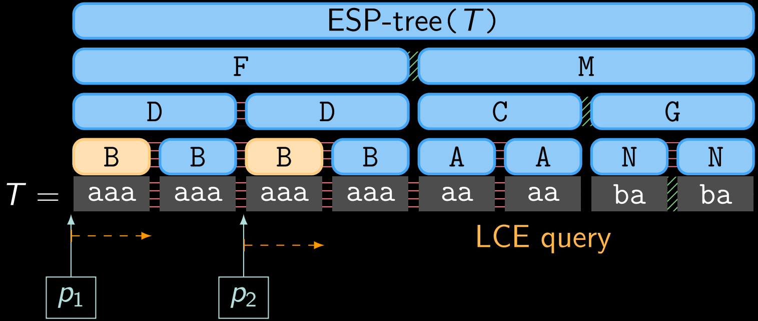
- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings



Candidate: ESP-tree

Cormode,Muthu.'07

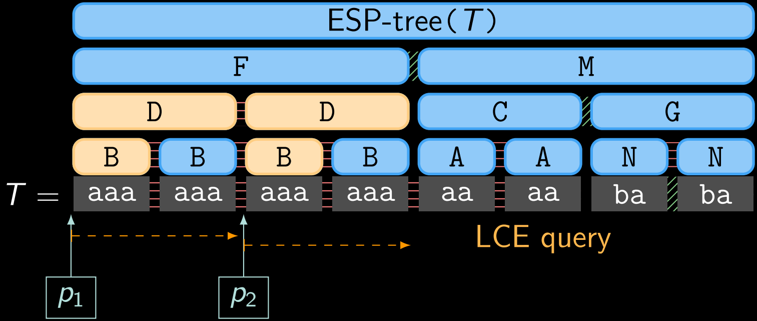
- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings



Candidate: ESP-tree

Cormode,Muthu.'07

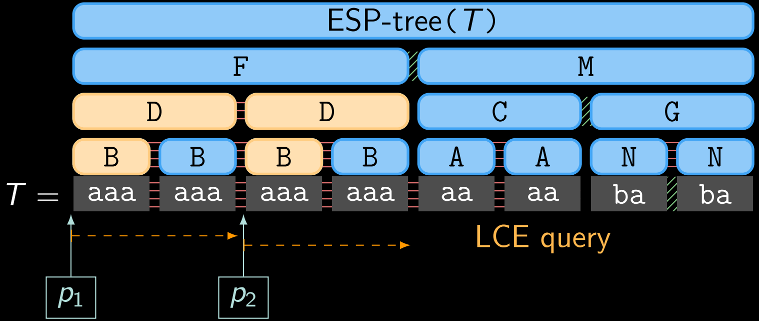
- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings



Candidate: ESP-tree

Cormode,Muthu.'07

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings



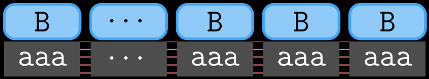
Candidate: ESP-tree Cormode,Muthu.'07

- grammar tree
- same substrings have mostly same parsing
- LCE query: compare nodes instead of strings

special case: repetitions!

greedy parsing

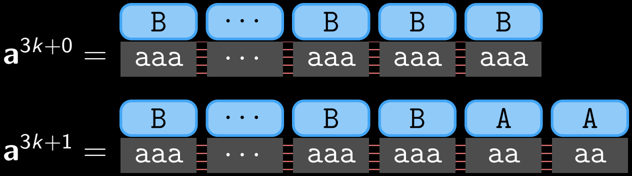
$\mathbf{a}^{3k+0} =$



B ⋯ B B B

aaa ⋯ aaa aaa aaa

greedy parsing



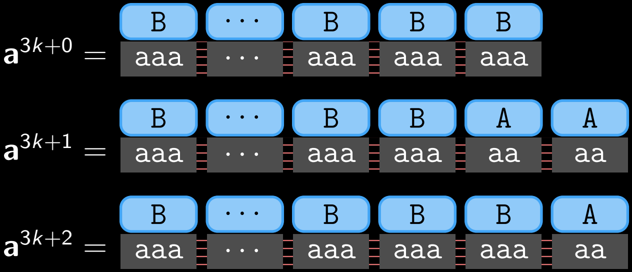
$\mathbf{a}^{3k+0} =$ aaa · · · aaa aaa aaa

B · · · B B B

$\mathbf{a}^{3k+1} =$ aaa · · · aaa aaa aa aa

B · · · B B A A

greedy parsing



$\mathbf{a}^{3k+0} =$

B ⋯ B B B

aaa ⋯ aaa aaa aaa

$\mathbf{a}^{3k+1} =$

B ⋯ B B A A

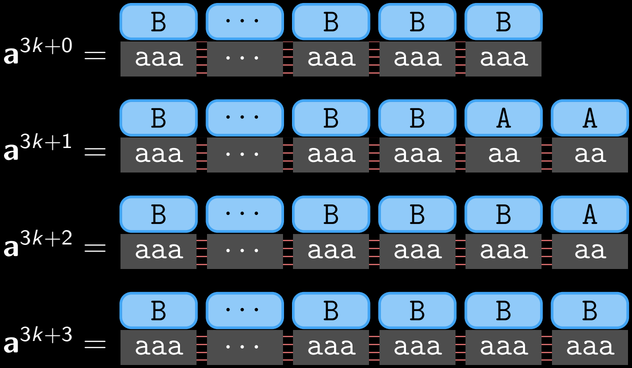
aaa ⋯ aaa aaa aa aa

$\mathbf{a}^{3k+2} =$

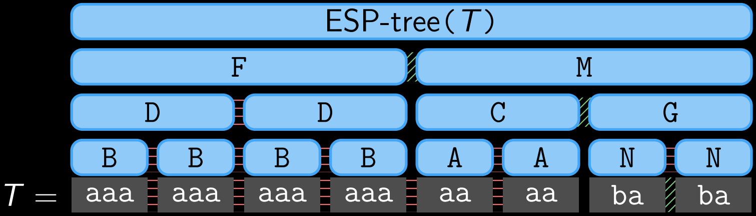
B ⋯ B B B A

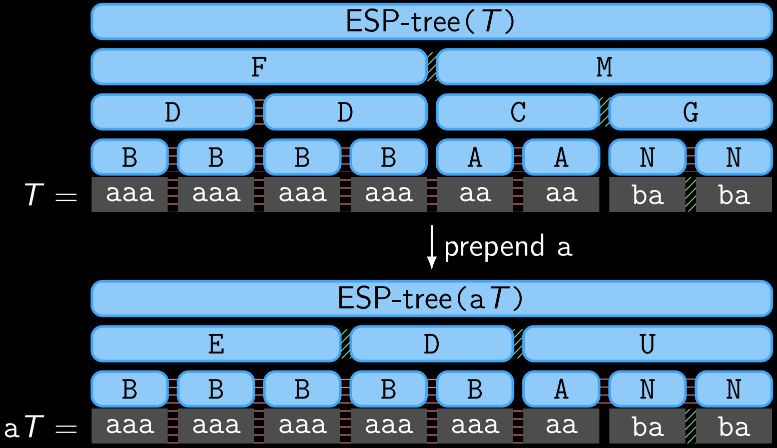
aaa ⋯ aaa aaa aaa aa

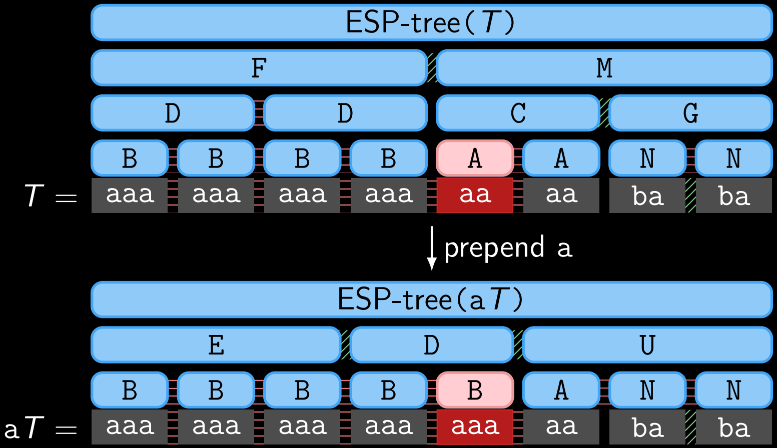
greedy parsing



	B	⋯	B	B	B	
$\mathbf{a}^{3k+0} =$	aaa	⋯	aaa	aaa	aaa	
	B	⋯	B	B	A	A
$\mathbf{a}^{3k+1} =$	aaa	⋯	aaa	aaa	aa	aa
	B	⋯	B	B	B	A
$\mathbf{a}^{3k+2} =$	aaa	⋯	aaa	aaa	aaa	aa
	B	⋯	B	B	B	B
$\mathbf{a}^{3k+3} =$	aaa	⋯	aaa	aaa	aaa	aaa


ESP-tree(T)
F
M
D
D
C
G
B
B
B
B
A
A
N
N
$T =$
aaa
aaa
aaa
aaa
aa
aa
ba
ba


ESP-tree(T)
F
M
D
D
C
G
B
B
B
B
A
A
N
N
$T =$
aaa
aaa
aaa
aaa
aa
aa
ba
ba
prepend a
ESP-tree(a T)
E
D
U
B
B
B
B
B
A
N
N
$\mathtt{a}T =$
aaa
aaa
aaa
aaa
aaa
aa
ba
ba



ESP-tree(T)

F M

D D C G

B B B B A A N N

T = aaa aaa aaa aaa aa aa ba ba

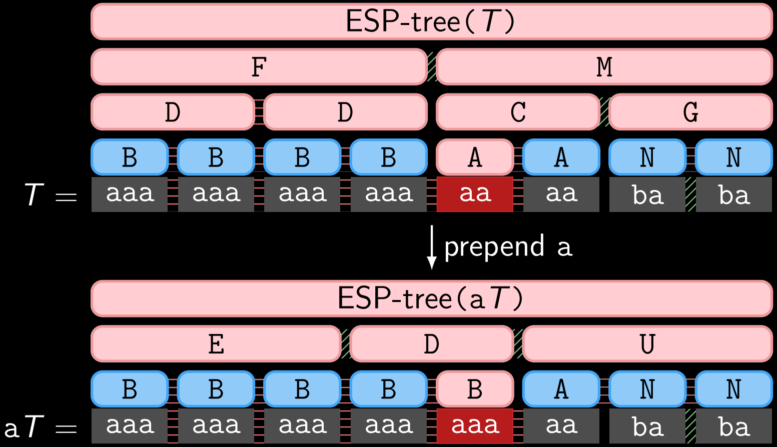
prepend a

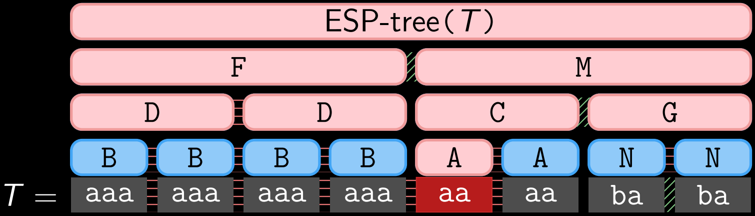
ESP-tree(aT)

E D U

B B B B B A N N

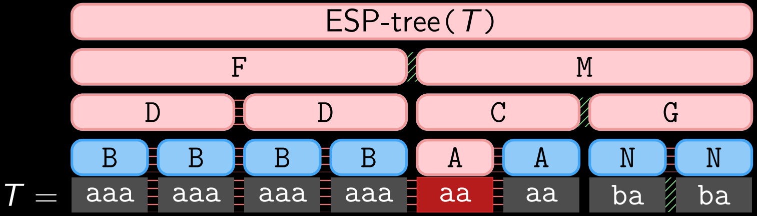
aT = aaa aaa aaa aaa aaa aa ba ba


ESP-tree(T)
F
M
D
D
C
G
B
B
B
B
A
A
N
N
T =
aaa
aaa
aaa
aaa
aa
aa
ba
ba
prepend a
ESP-tree(aT)
E
D
U
B
B
B
B
B
A
N
N
aT =
aaa
aaa
aaa
aaa
aaa
aa
ba
ba



Cormode,Muthu'07

Prepending a character changes $\mathcal{O}(\lg n \lg^* n)$ nodes.



Cormode,Muthu'07

Prepending a character changes $\cancel{\mathcal{O}(\lg n \lg^* n)}$ nodes.

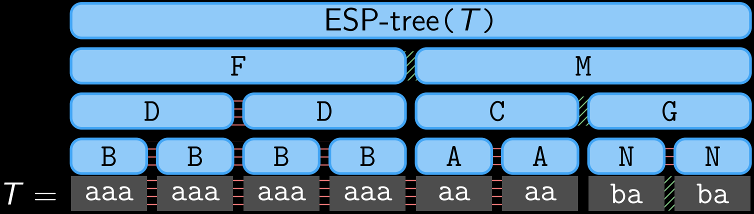
📄: can be $\Omega(\lg^2 n)$ but at most $\mathcal{O}(\lg^2 n \lg^* n)$

📄 Fischer, I, Köppl

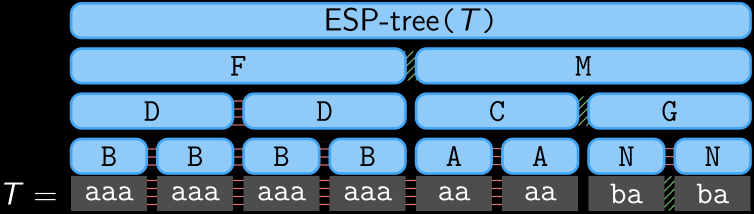
Deterministic sparse suffix sorting in the restore model.

Submitted to TALG, **2018**

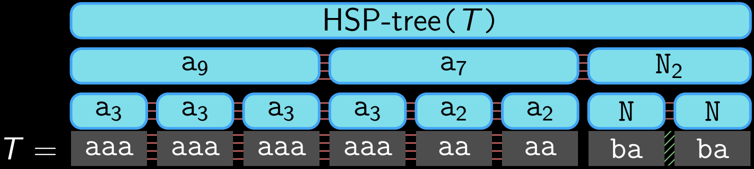
HSP trees



HSP trees

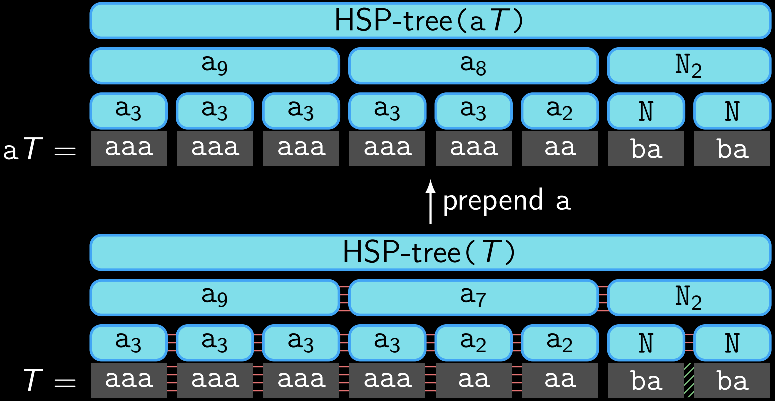


ESP-tree(T)
F
M
D
D
C
G
B
B
B
B
A
A
N
N
T =
aaa
aaa
aaa
aaa
aa
aa
ba
ba

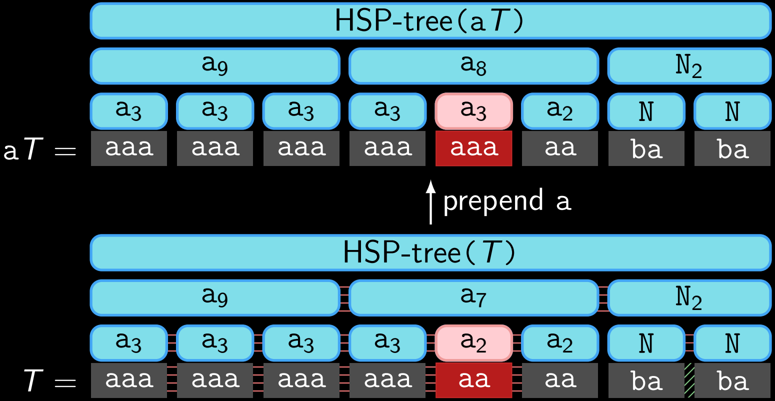


HSP-tree(T)
a_9
a_7
N_2
a_3
a_3
a_3
a_3
a_2
a_2
N
N
T =
aaa
aaa
aaa
aaa
aa
aa
ba
ba

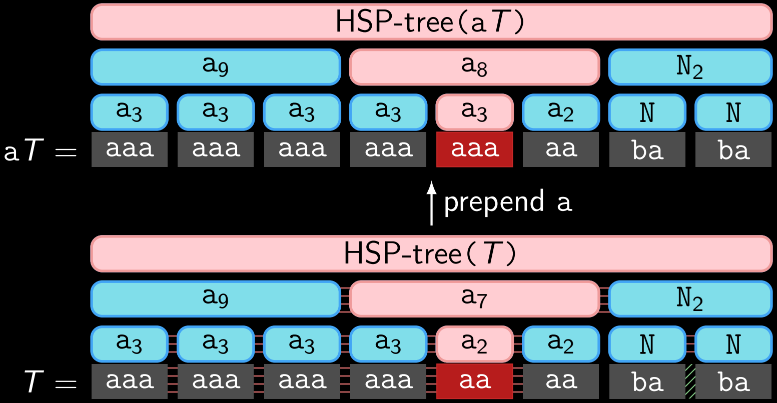
HSP trees


HSP-tree($\mathtt{a}T$)
$\mathtt{a}_9$
$\mathtt{a}_8$
$\mathtt{N}_2$
$\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_2$ $\mathtt{N}$ $\mathtt{N}$
$\mathtt{a}T =$ aaa aaa aaa aaa aaa aa ba ba
prepend a
HSP-tree(T)
$\mathtt{a}_9$
$\mathtt{a}_7$
$\mathtt{N}_2$
$\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_3$ $\mathtt{a}_2$ $\mathtt{a}_2$ $\mathtt{N}$ $\mathtt{N}$
$T =$ aaa aaa aaa aaa aa aa ba ba

HSP trees


HSP-tree(aT)
a$_9$
a$_8$
N$_2$
a$_3$
a$_3$
a$_3$
a$_3$
a$_3$
a$_2$
N
N
aT =
aaa
aaa
aaa
aaa
aaa
aa
ba
ba
prepend a
HSP-tree(T)
a$_9$
a$_7$
N$_2$
a$_3$
a$_3$
a$_3$
a$_3$
a$_2$
a$_2$
N
N
T =
aaa
aaa
aaa
aaa
aa
aa
ba
ba

HSP trees



HSP-tree(aT)

a_9 a_8 N_2

a_3 a_3 a_3 a_3 a_3 a_2 N N

aT = aaa aaa aaa aaa aaa aa ba ba

prepend a

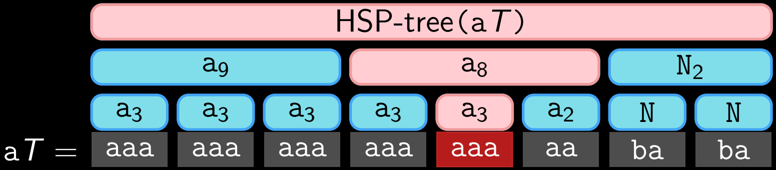
HSP-tree(T)

a_9 a_7 N_2

a_3 a_3 a_3 a_3 a_2 a_2 N N

T = aaa aaa aaa aaa aa aa ba ba

HSP trees



node changes

ESP-tree $\mathcal{O}(\lg^2 n \lg^* n)$ nodes → HSP-tree $\mathcal{O}(\lg n \lg^* n)$ nodes

time bound

$$\mathcal{O}\Big(\underbrace{c(\sqrt{\lg \sigma} + \lg\lg n)}_{\text{construction on } \mathcal{O}(c) \text{ characters}} + \overbrace{m \lg m}^{\text{search tree}} \underbrace{\lg n \lg^* n}_{\text{LCE queries}}\Big)$$

HSP-trees

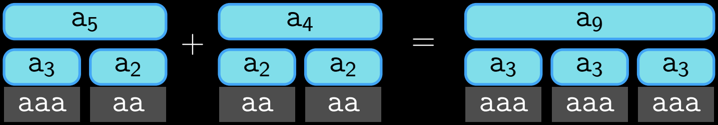
- $\mathcal{O}(\lg n \lg^* n)$ time per query

time bound

$$\mathcal{O}(\underbrace{c(\sqrt{\lg\sigma}+\lg\lg n)}_{\text{construction on } \mathcal{O}(c) \text{ characters}} + \overbrace{m\lg m}^{\text{search tree}}\ \underbrace{\lg n \lg^* n}_{\text{LCE queries}})$$

HSP-trees

- $\mathcal{O}(\lg n \lg^* n)$ time per query
- is mergeable in $\mathcal{O}(\lg n \lg^* n)$ time

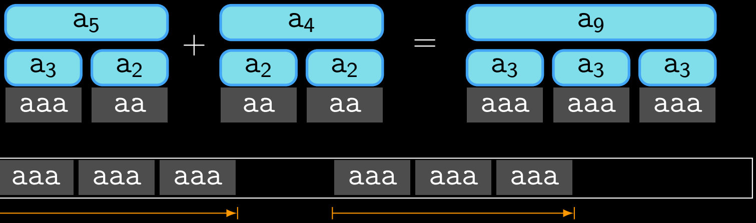


time bound

$$\mathcal{O}(\underbrace{c(\sqrt{\lg \sigma} + \lg \lg n)}_{\text{construction on } \mathcal{O}(c) \text{ characters}} + \overbrace{m \lg m}^{\text{search tree}} \underbrace{\lg n \lg^* n}_{\text{LCE queries}})$$

HSP-trees

- $\mathcal{O}(\lg n \lg^* n)$ time per query
- is mergeable in $\mathcal{O}(\lg n \lg^* n)$ time
- is storable in text space

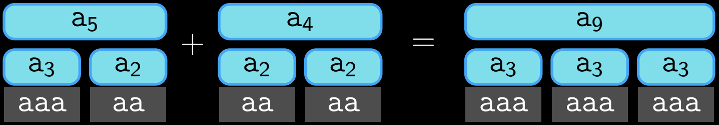


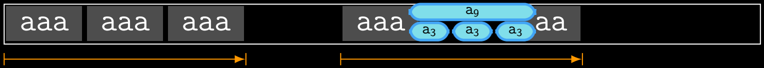
time bound

$$\mathcal{O}(\underbrace{c(\sqrt{\lg\sigma}+\lg\lg n)}_{\text{construction on }\mathcal{O}(c)\text{ characters}} + \overbrace{m\lg m}^{\text{search tree}}\ \underbrace{\lg n\lg^* n}_{\text{LCE queries}})$$

HSP-trees

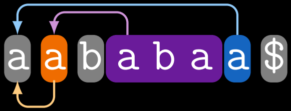
- $\mathcal{O}(\lg n\lg^* n)$ time per query
- is mergeable in $\mathcal{O}(\lg n\lg^* n)$ time
- is storable in text space





II. Lempel-Ziv factorization

1. LZ77
2. LZ78



LZ77

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

1 2 3 4 5 6 7 8

Coding:

LZ77

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

1 2 3 4 5 6 7 8

Coding: a

LZ77

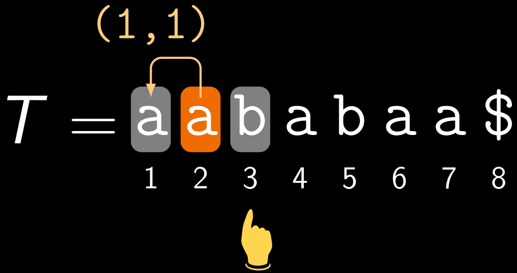
(1,1)

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

1 2 3 4 5 6 7 8

Coding: a(1,1)

LZ77



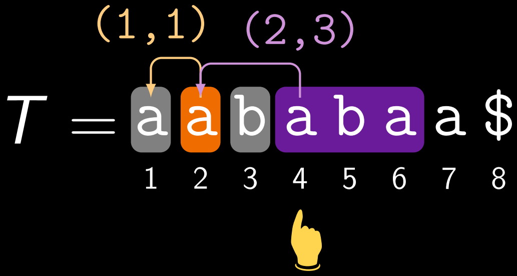
(1,1)

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

1 2 3 4 5 6 7 8

Coding: a(1,1)b

LZ77



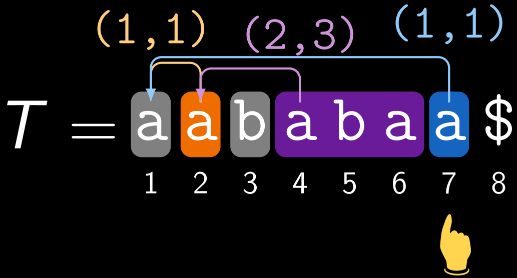
(1,1) (2,3)

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

1 2 3 4 5 6 7 8

Coding: a(1,1)b(2,3)

LZ77



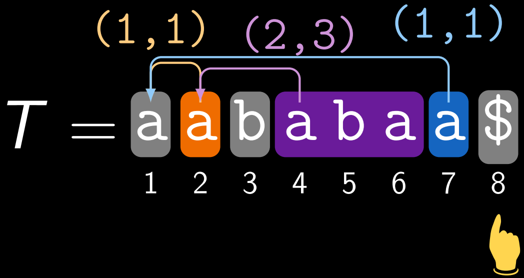
(1,1) (2,3) (1,1)

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

1 2 3 4 5 6 7 8

Coding: a(1,1)b(2,3)(1,1)

LZ77



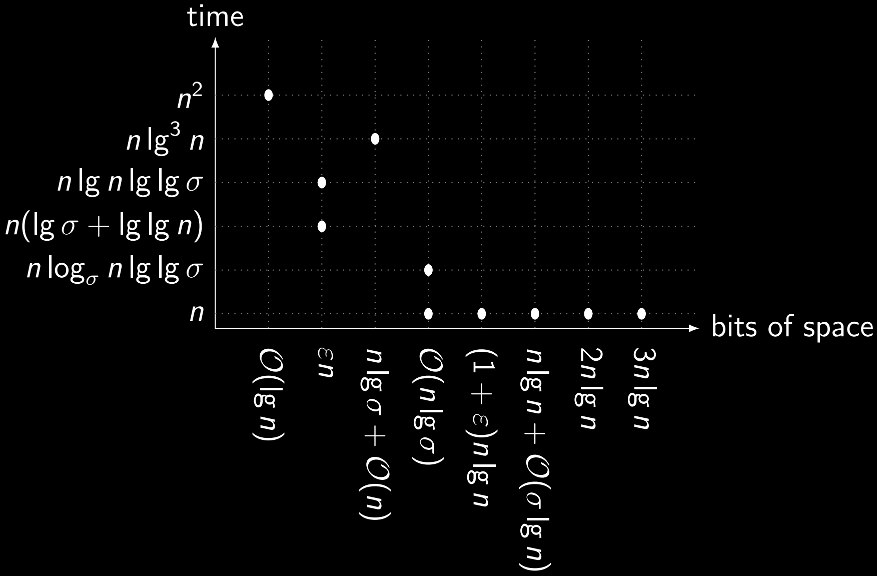
(1,1) (2,3) (1,1)

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

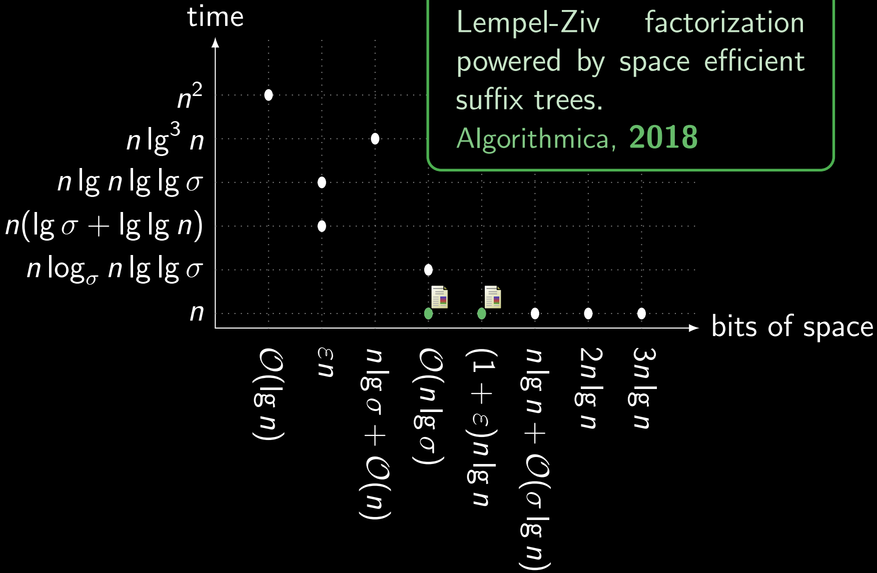
1 2 3 4 5 6 7 8

Coding: a(1,1)b(2,3)(1,1)$

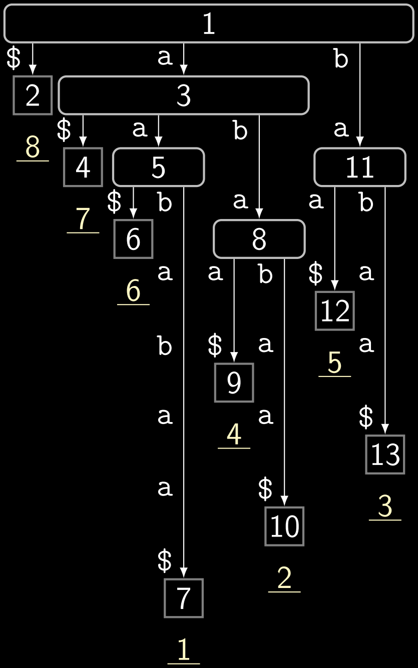
LZ77


time
n^2
$n \lg^3 n$
$n \lg n \lg \lg \sigma$
$n(\lg \sigma + \lg \lg n)$
$n \log_\sigma n \lg \lg \sigma$
n
bits of space
$\mathcal{O}(\lg n)$
εn
$n \lg \sigma + \mathcal{O}(n)$
$\mathcal{O}(n \lg \sigma)$
$(1+\varepsilon) n \lg n$
$n \lg n + \mathcal{O}(\sigma \lg n)$
$2n \lg n$
$3n \lg n$

LZ77

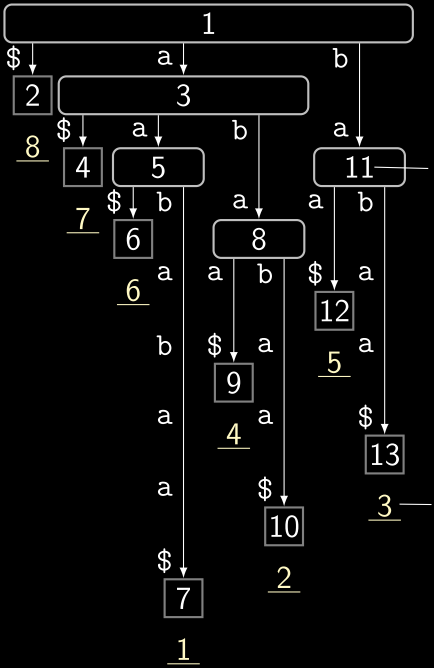


Fischer, I, Köppl, Sadakane
Lempel-Ziv factorization powered by space efficient suffix trees.
Algorithmica, **2018**



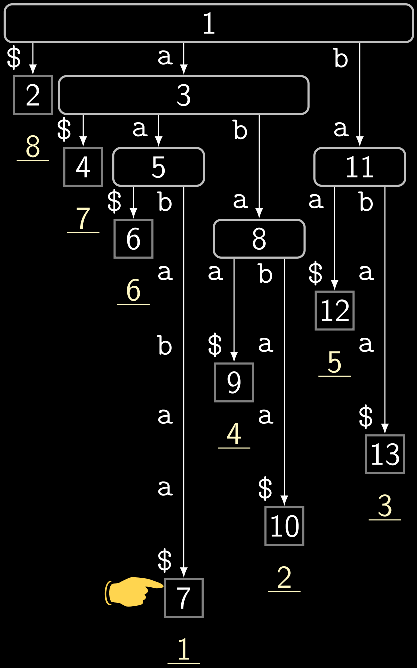
suffix tree

suffix tree



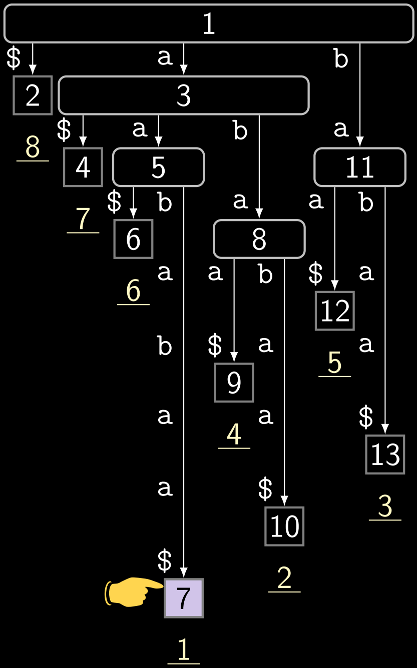
pre-order number

suffix number



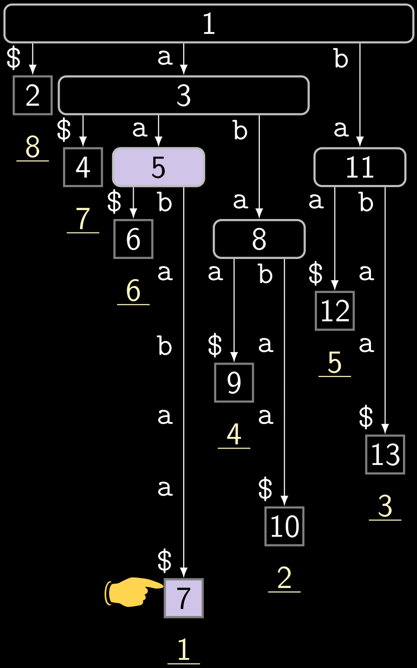
suffix tree

- walk from leaf to root



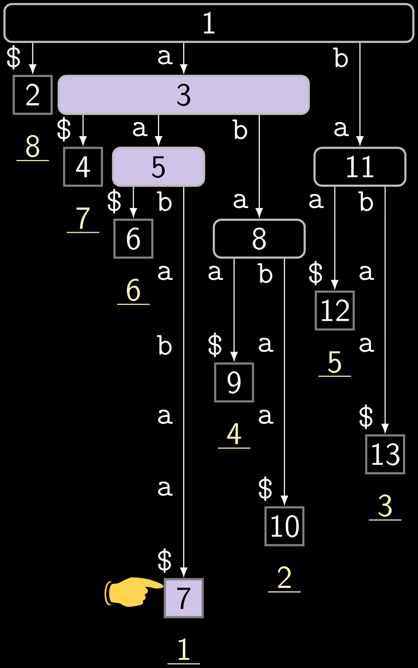
suffix tree

- walk from leaf to root
- mark visited nodes



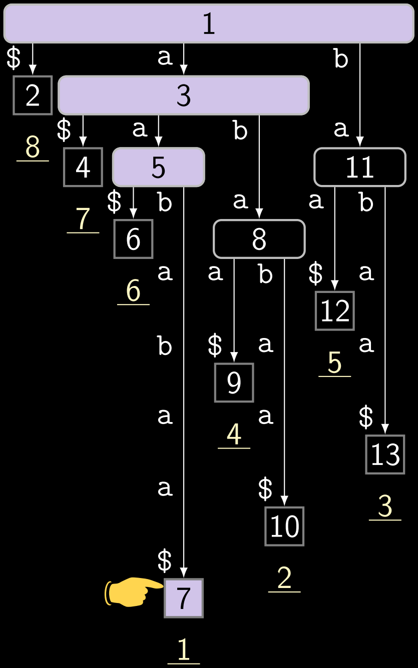
suffix tree

- walk from leaf to root
- mark visited nodes



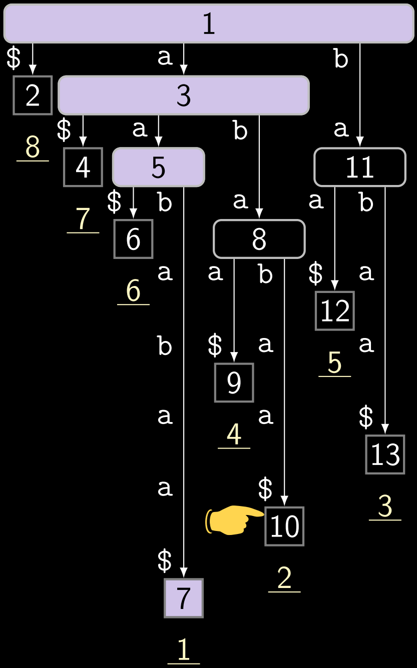
suffix tree

- walk from leaf to root
- mark visited nodes



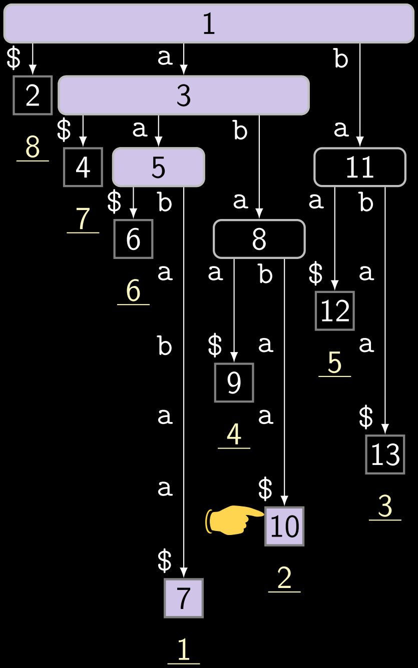
suffix tree

- walk from leaf to root
- mark visited nodes



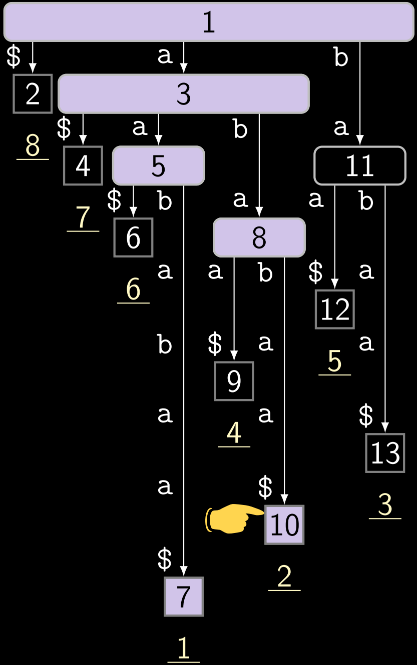
suffix tree

- walk from leaf to root
- mark visited nodes
- select leaves in text order



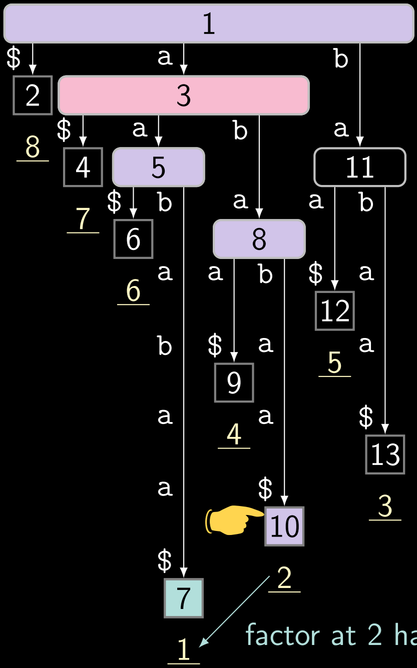
suffix tree

- walk from leaf to root
- mark visited nodes
- select leaves in text order



suffix tree

- walk from leaf to root
- mark visited nodes
- select leaves in text order



suffix tree

- walk from leaf to root
- mark visited nodes
- select leaves in text order
- visited nodes witnesses reference

factor at 2 has reference 1

suffix trees

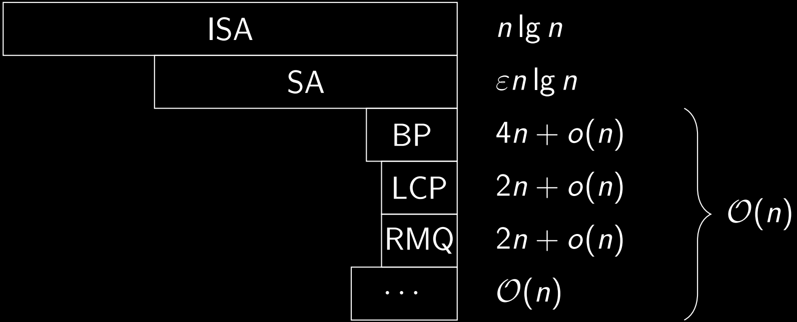
construction Farach-Colton+'00

- $\mathcal{O}(n)$ time
- $\mathcal{O}(n \lg n)$ bits working space

space-efficient $\mathcal{O}(n)$ time constructions:

- $(1+\varepsilon)n \lg n + \mathcal{O}(n)$ bits with $0 < \varepsilon \leq 1$ *succinct*
- $\mathcal{O}(n \lg \sigma)$ bits due to Munro+'17 *compressed*

succinct suffix tree



ISA $n \lg n$

SA $\varepsilon n \lg n$

BP $4n + o(n)$

LCP $2n + o(n)$

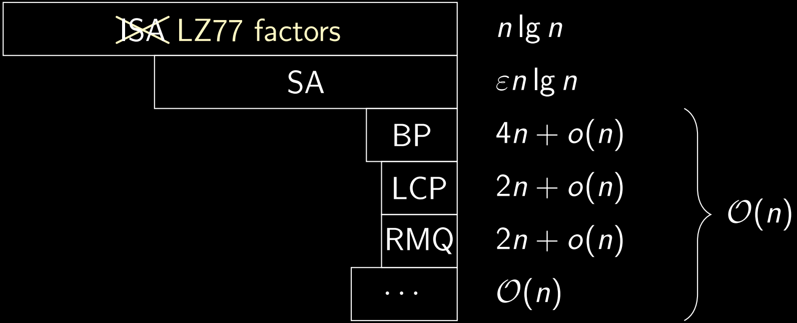
RMQ $2n + o(n)$

$\cdots$ $\mathcal{O}(n)$

$\mathcal{O}(n)$

total space: $(1+\varepsilon)n \lg n + \mathcal{O}(n)$ bits.

succinct suffix tree

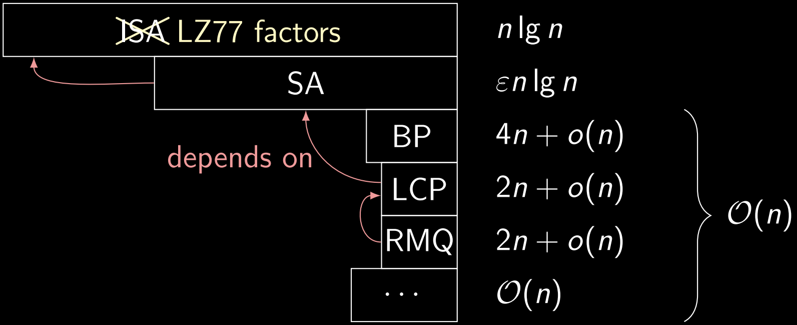


~~ISA~~ LZ77 factors	$n \lg n$	
SA	$\varepsilon n \lg n$	
BP	$4n + o(n)$	$\mathcal{O}(n)$
LCP	$2n + o(n)$	
RMQ	$2n + o(n)$	
$\cdots$	$\mathcal{O}(n)$	

total space: $(1 + \varepsilon) n \lg n + \mathcal{O}(n)$ bits.

goal: overwrite ISA with LZ77 factors.

succinct suffix tree



~~ISA~~ LZ77 factors — $n \lg n$

SA — $\varepsilon n \lg n$

BP — $4n + o(n)$

LCP — $2n + o(n)$

RMQ — $2n + o(n)$

$\cdots$ — $\mathcal{O}(n)$

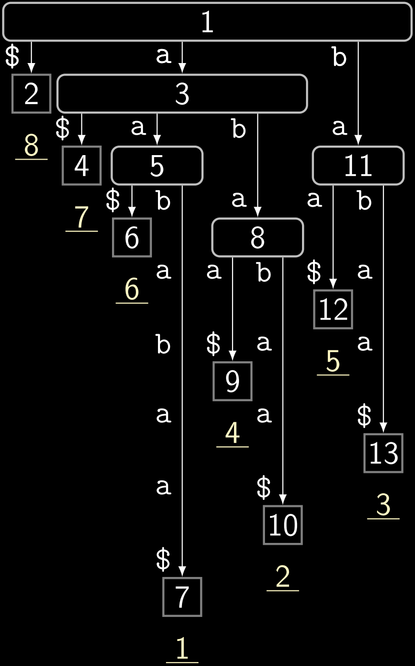
$\mathcal{O}(n)$

depends on

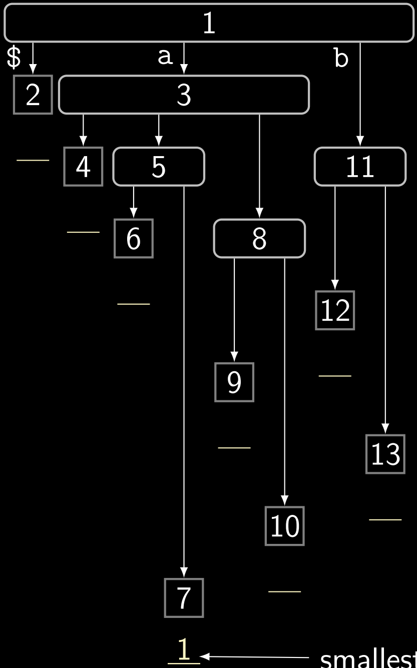
total space: $(1 + \varepsilon) n \lg n + \mathcal{O}(n)$ bits.

goal: overwrite ISA with LZ77 factors.

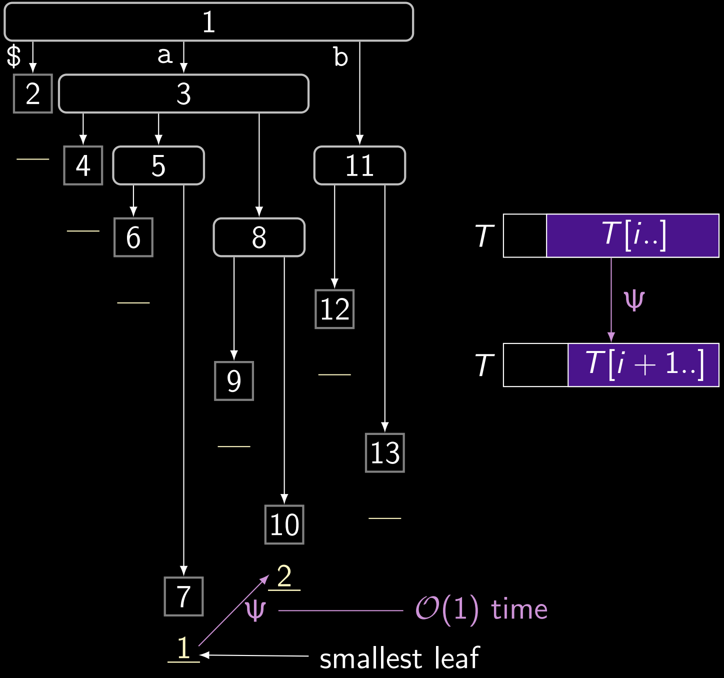
problem: RMQ, LCP, and SA depend on ISA.



1
$
a
b
2
3
$
a
b
a
8
4
5
11
$
b
a
a
b
7
6
8
a
a
b
$
a
6
12
b
$
a
a
9
5
a
a
$
4
13
a
$
3
10
$
2
7
1

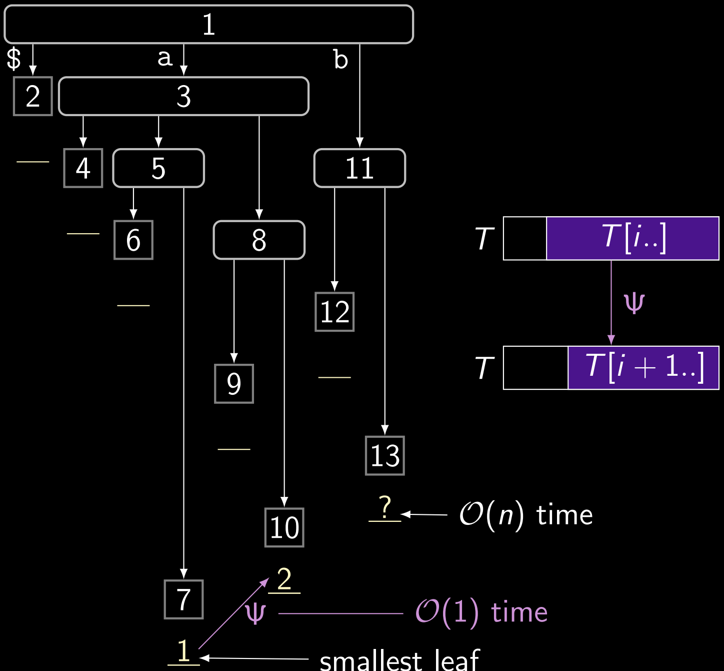


smallest leaf



$\mathcal{O}(1)$ time

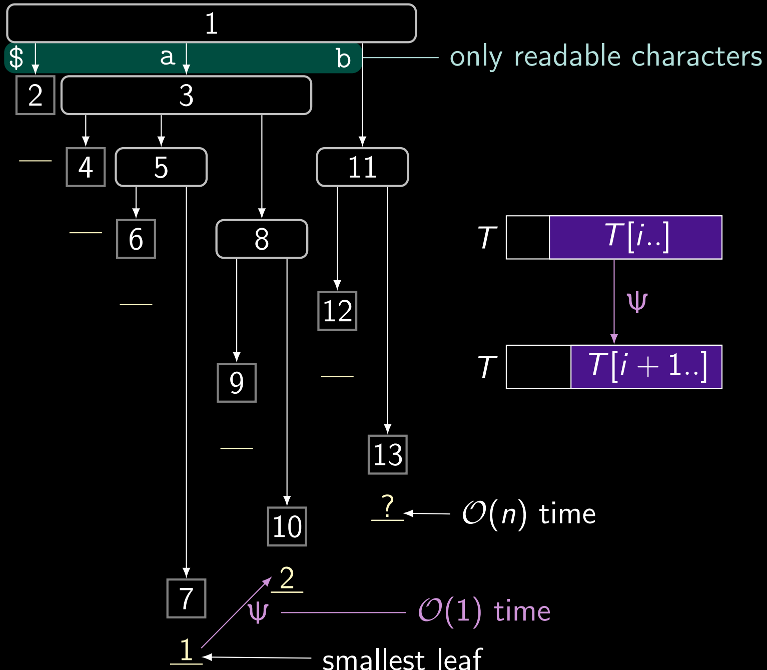
smallest leaf



$\mathcal{O}(n)$ time

$\mathcal{O}(1)$ time

smallest leaf

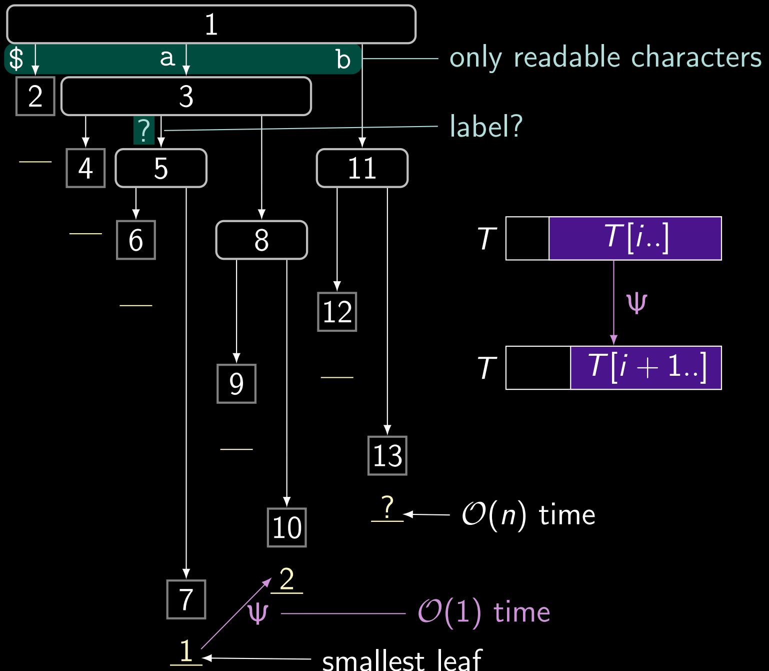


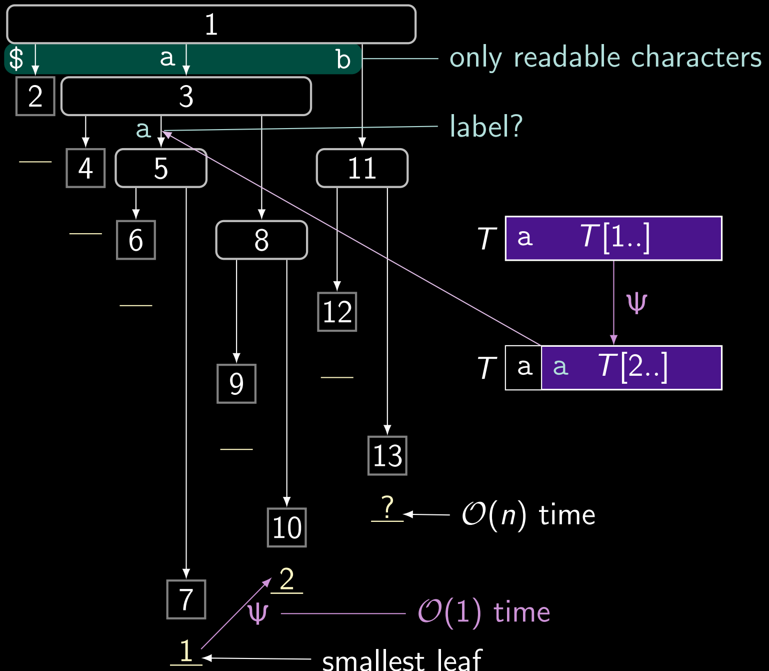
only readable characters

$\mathcal{O}(n)$ time

$\mathcal{O}(1)$ time

smallest leaf


1
$
a
b
only readable characters
2
3
?
label?
4
5
11
6
8
T
$T[i..]$
ψ
12
T
$T[i+1..]$
9
13
?
$\mathcal{O}(n)$ time
10
2
7
ψ
$\mathcal{O}(1)$ time
1
smallest leaf


1
$
a
b
only readable characters
2
3
a
label?
4
5
11
6
8
T
a
T[1..]
ψ
12
T
a
a
T[2..]
9
13
?
𝒪(n) time
10
2
ψ
𝒪(1) time
7
1
smallest leaf

LZ78

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

1	2	3	4	5	6	7	8

0 LZ trie

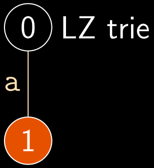
Coding:

LZ78

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

1 2 3 4 5 6 7 8

Coding: (0,a)



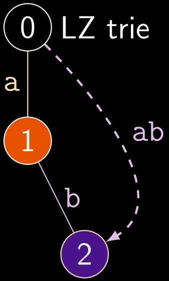
LZ78

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

1 2 3 4 5 6 7 8

👆

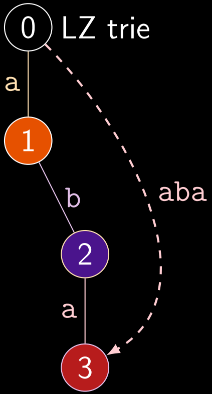
Coding: (0,a)(1,b)



LZ78

$T =$	a	a	b	a	b	a	a	$
	1	2	3	4	5	6	7	8

Coding: (0,a)(1,b)(2,a)

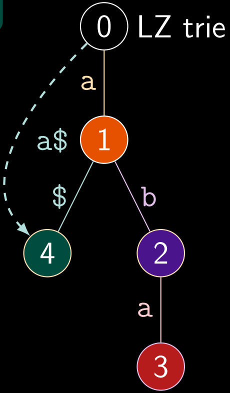


LZ78

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

1 2 3 4 5 6 7 8

Coding: (0,a)(1,b)(2,a)(1,$)



LZ78

two approaches

1. suffix tree based
2. LZ trie based

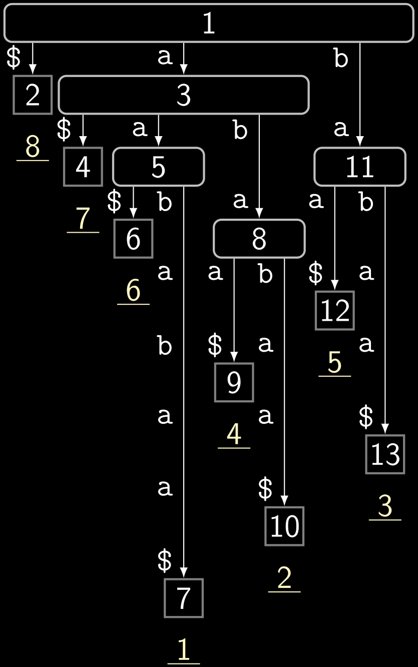
1. suffix tree based

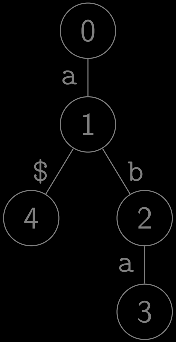
time	bits	authors
$\mathcal{O}(n)$	$\mathcal{O}(n \lg n)$	Nakashima+'15
$\mathcal{O}(n/\varepsilon)$	$(1+\varepsilon)n \lg n + \mathcal{O}(n)$	📄
$\mathcal{O}(n)$	$\mathcal{O}(n \lg \sigma)$	📄

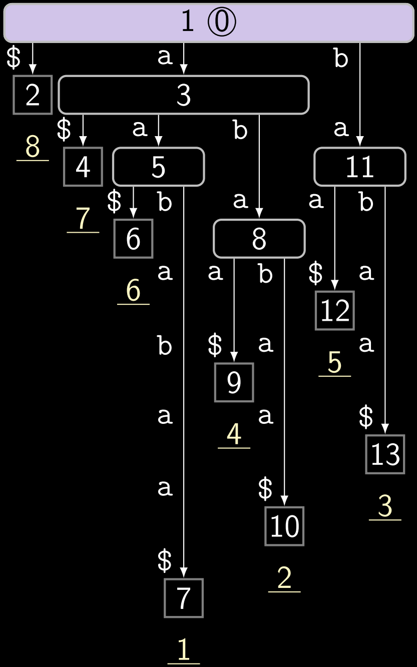
📄 Fischer, I, Köppl, Sadakane

Lempel-Ziv factorization powered by space efficient suffix trees.

Algorithmica, **2018**

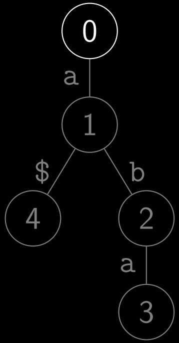

1
$
a
b
2
3
$
a
b
a
8
4
5
11
$
b
a
a
b
7
6
8
a
a
b
$
a
6
12
b
$
a
a
9
5
a
a
$
4
13
a
$
3
10
2
$
7
1


0
a
1
$
b
4
2
a
3

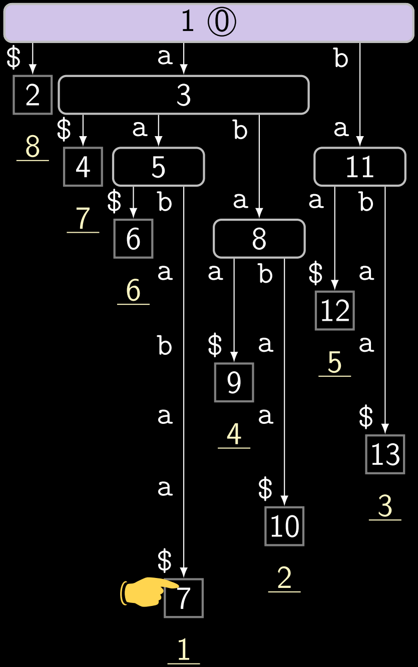


1 ⓪
$
a
b
2
3
8
$
a
b
a
4
5
11
7
$
b
a
a
b
6
8
6
a
a
b
$
a
12
b
$
a
a
9
5
a
4
a
$
13
a
3
$
10
2
$
7
1

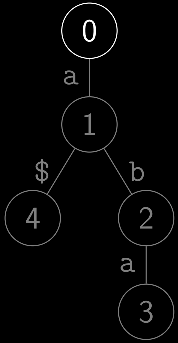
- mark root

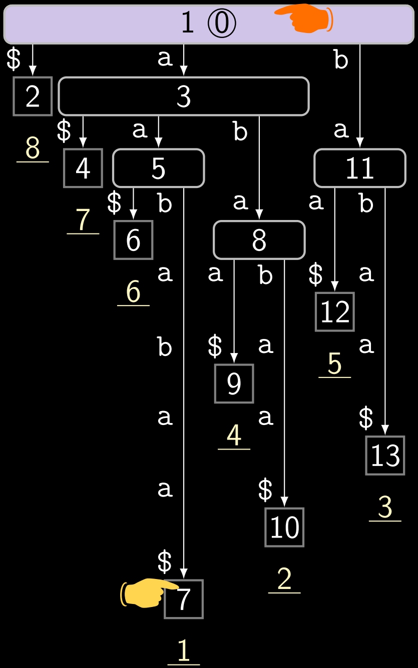


0
a
1
$
b
4
2
a
3

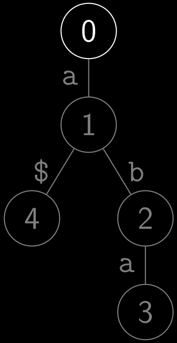


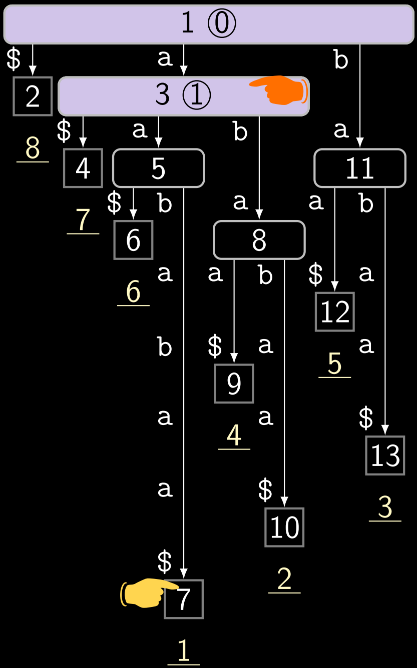
- mark root
- move to highest *unmarked* ancestor



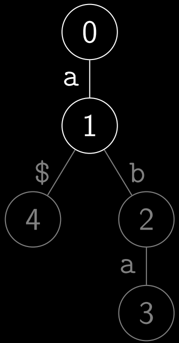


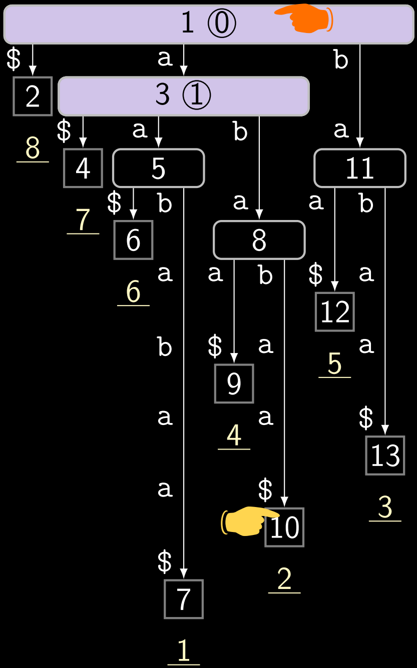
- mark root
- move to highest *unmarked* ancestor



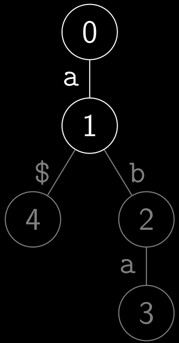


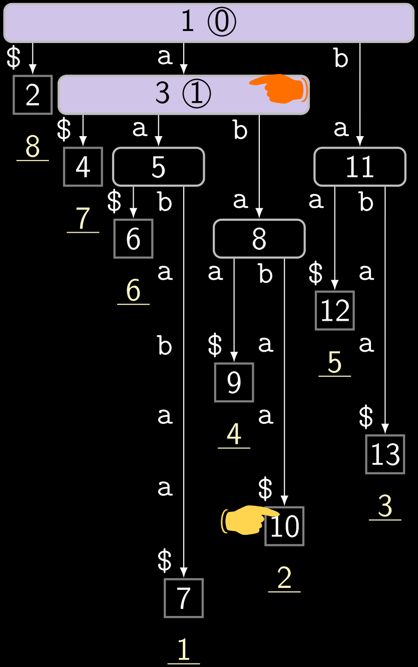
- mark root
- move to highest *unmarked* ancestor



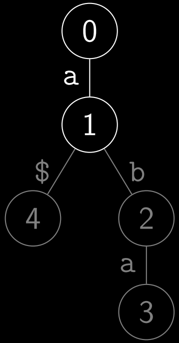


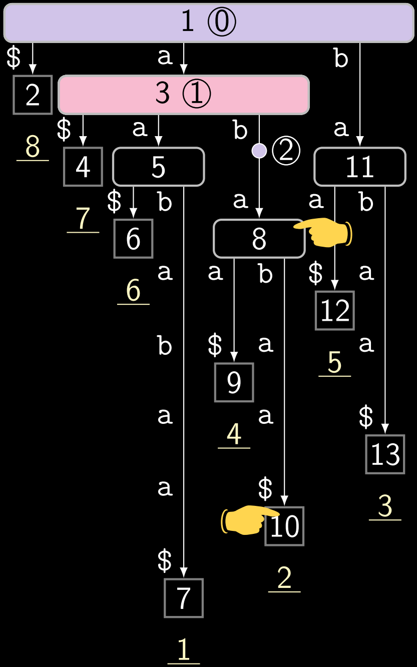
- mark root
- move to highest *unmarked* ancestor
- select leaves in text order



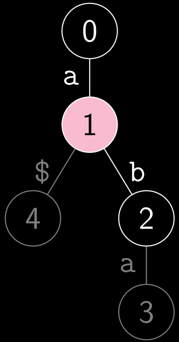


- mark root
- move to highest *unmarked* ancestor
- select leaves in text order





- mark root
- move to highest *unmarked* ancestor
- select leaves in text order
- last marked node is parent of new leaf



2. LZ trie based

time	**bits**	**authors**
$\mathcal{O}(n \lg \sigma)$	$\mathcal{O}(z \lg z)$	Lempel, Ziv'78
$\mathcal{O}\left(n + z\frac{\lg^2 \lg \sigma}{\lg \lg \lg \sigma}\right)$	$\mathcal{O}(z \lg z)$	Fischer, Gawrychowski'15
$\mathcal{O}(n)$ whp.	$\mathcal{O}(z \lg(\sigma z))$	📄

z : # factors

📄 Fischer, Köppl
Practical evaluation of Lempel-Ziv-78 and Lempel-Ziv-Welch tries.
In Proc. SPIRE, **2017**

LZ trie implementations

baseline:

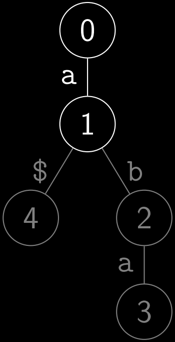
- binary first-child-next-sibling
- ternary Bentley, Sedgewick'97

new tries:

- hash: hash table representation
- compact hash: quotienting of hash
- rolling: store Karp-Rabin fingerprints in hash table

hash

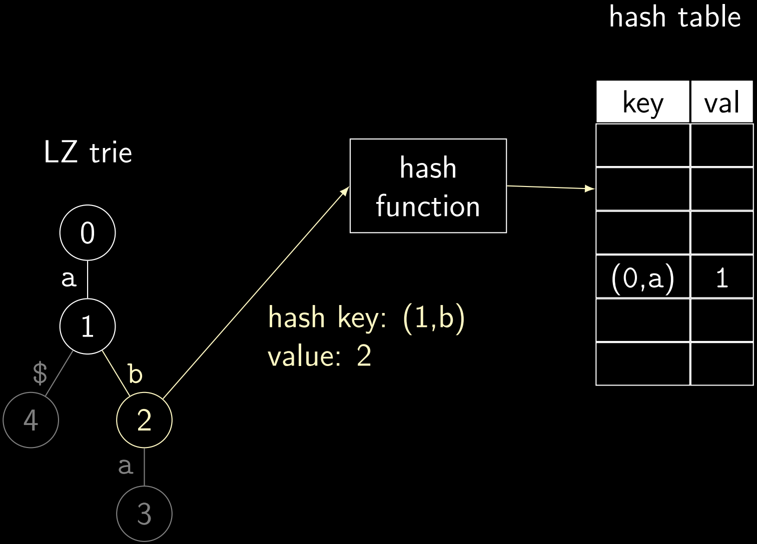
LZ trie



hash table

key	val
(0,a)	1

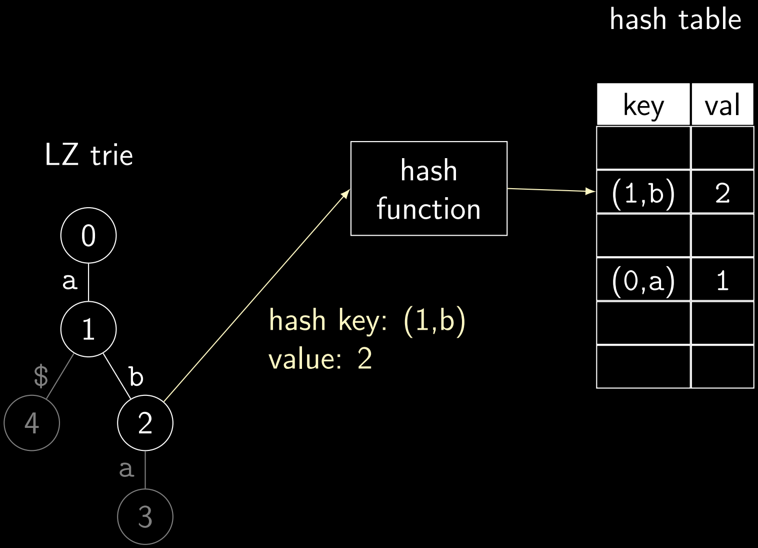
hash



hash key: (1,b)
value: 2

key	val
(0,a)	1

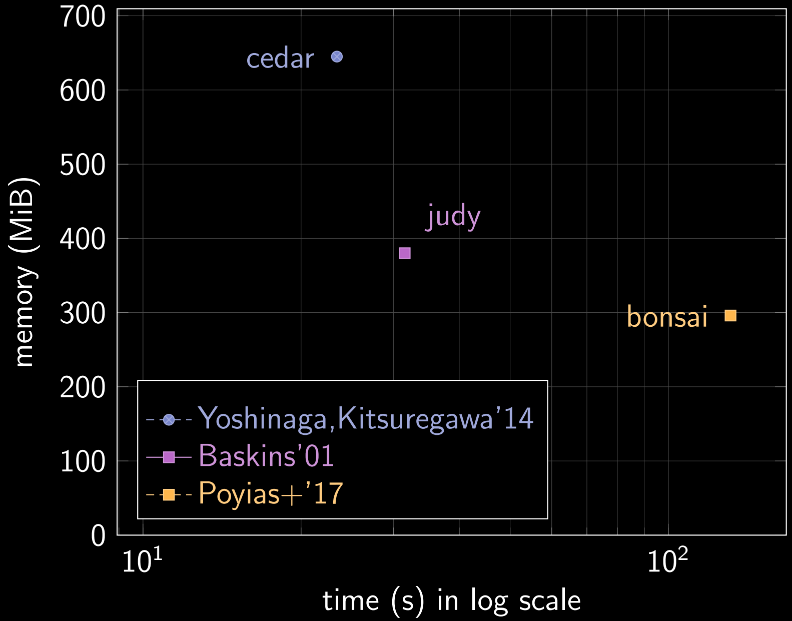
hash



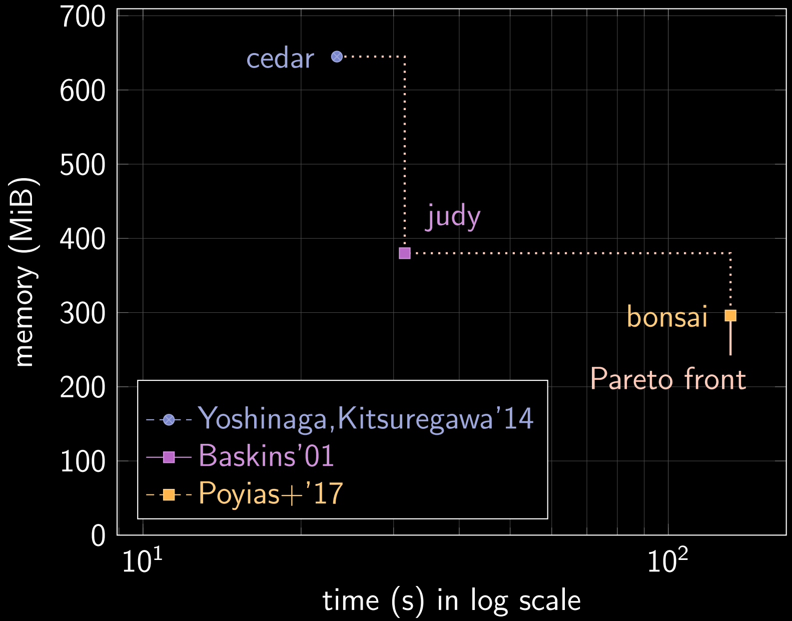
hash key: (1,b)
value: 2

key	val
(1,b)	2
(0,a)	1

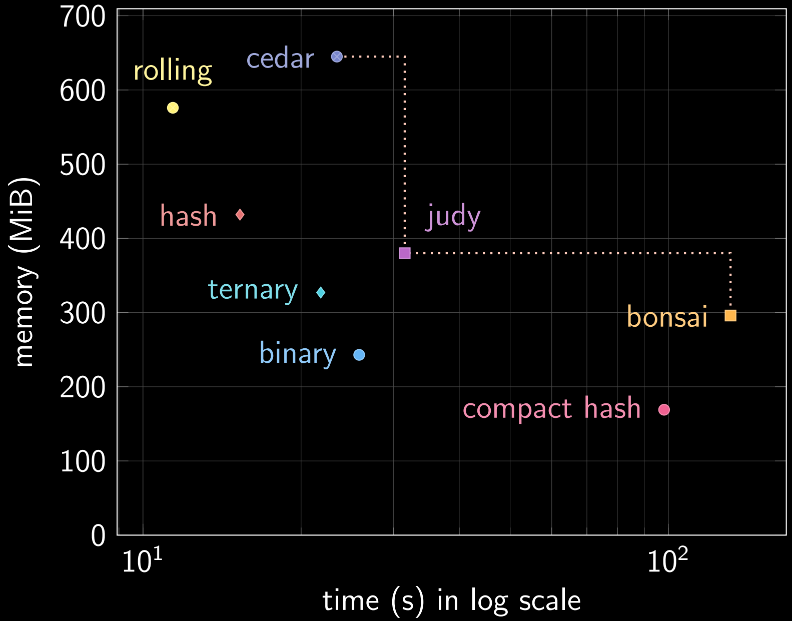
dataset PC-ENGLISH 200 MiB



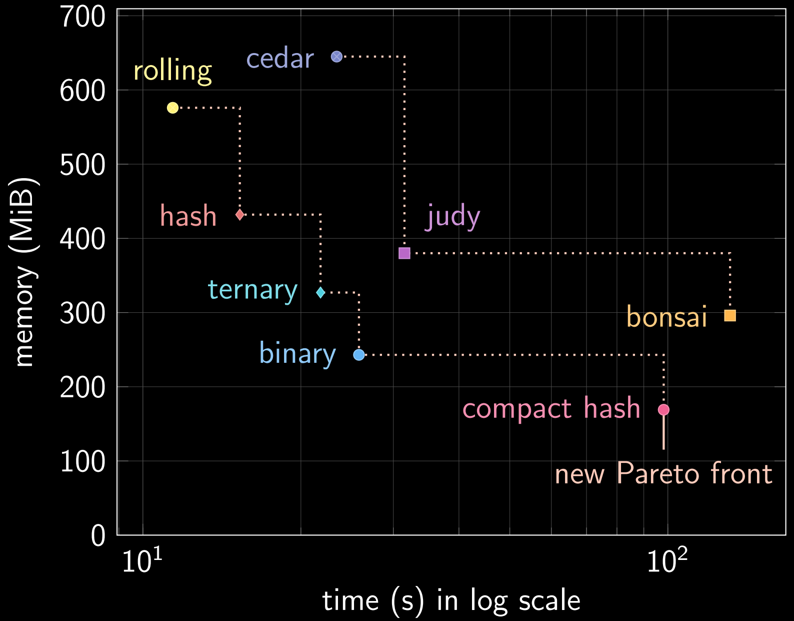
dataset PC-ENGLISH 200 MiB



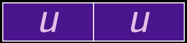
dataset PC-ENGLISH 200 MiB



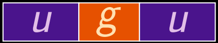
dataset PC-ENGLISH 200 MiB



III. regular structures


u u

squares

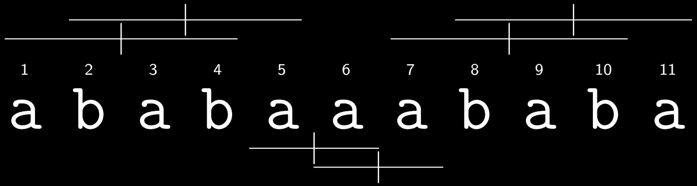

u g u

gapped repeats

distinct squares

1	2	3	4	5	6	7	8	9	10	11
a	b	a	b	a	a	a	b	a	b	a

distinct squares

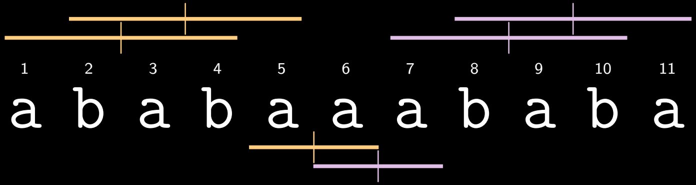


1	2	3	4	5	6	7	8	9	10	11
a	b	a	b	a	a	a	b	a	b	a

squares

- abab at 1
- baba at 2
- aa at 5
- aa at 6
- abab at 7
- baba at 8

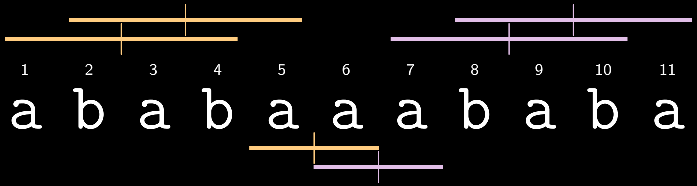
distinct squares



squares

- abab at 1
- baba at 2
- aa at 5
- aa at 6
- abab at 7
- baba at 8

distinct squares



leftmost squares

- abab at 1
- baba at 2
- aa at 5
- ~~aa at 6~~
- ~~abab at 7~~
- ~~baba at 8~~

finding distinct squares

algorithms

time	bits	authors
$\mathcal{O}(n)$	$\mathcal{O}(n \lg n)$	Crochemore+'14
$\mathcal{O}(n \lg^{\varepsilon} n)$	$\mathcal{O}(n \lg \sigma)$	📄
$\mathcal{O}\left(\frac{n}{\varepsilon}\right)$	$(2+\varepsilon) n \lg n + \mathcal{O}(n)$	📄
online:		
$\mathcal{O}\left(\frac{n \lg^2 \lg n}{\lg \lg \lg n}\right)$	$\mathcal{O}(n \lg n)$	📄

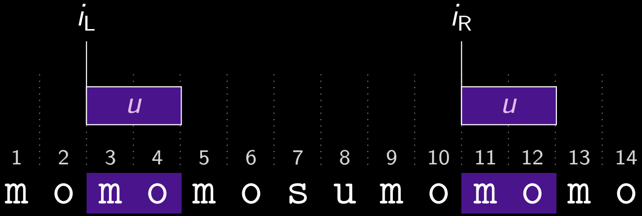
$0 < \varepsilon \leq 1$

📄 Bannai, Inenaga, Köppl
Computing all distinct squares in linear time for integer alphabets.
In Proc. CPM, **2017**

gapped repeats

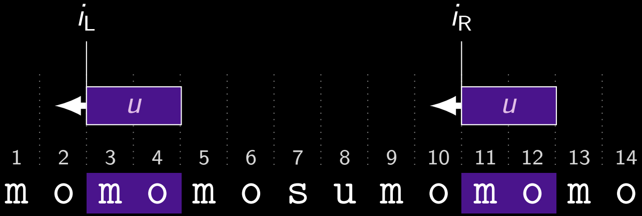
1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	o	m	o	m	o	s	u	m	o	m	o	m	o

gapped repeats



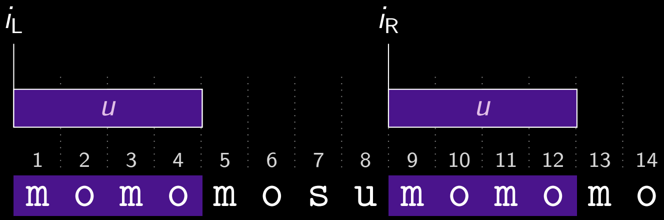
- gapped repeat (i_L, i_R, u)

gapped repeats



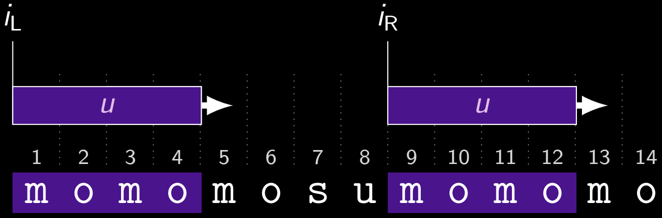
- gapped repeat (i_L, i_R, u)
- *maximal* if it cannot be extended
  - to the left nor

gapped repeats



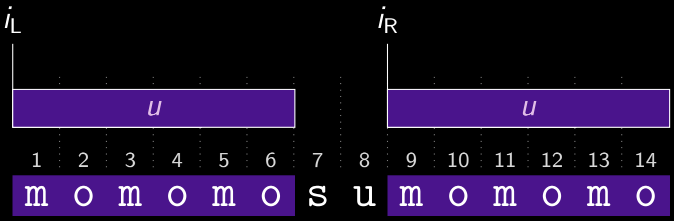
- gapped repeat (i_L, i_R, u)
- *maximal* if it cannot be extended
  - to the left nor

gapped repeats



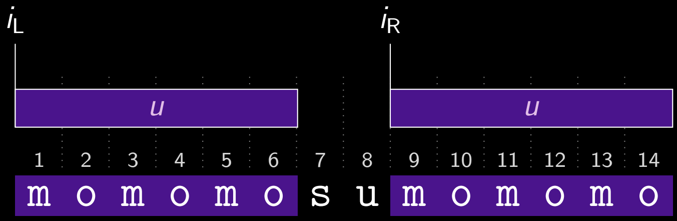
- gapped repeat (i_L, i_R, u)
- *maximal* if it cannot be extended
  - to the left nor
  - to the right.

gapped repeats



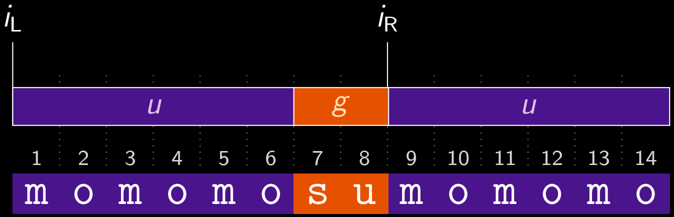
- gapped repeat (i_L, i_R, u)
- *maximal* if it cannot be extended
  - to the left nor
  - to the right.

α-gapped repeats



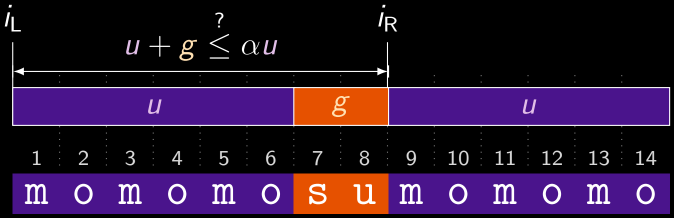
- maximal gapped repeat (i_L, i_R, u)

α-gapped repeats



- maximal gapped repeat (i_L, i_R, u)
- $g :=$ gap

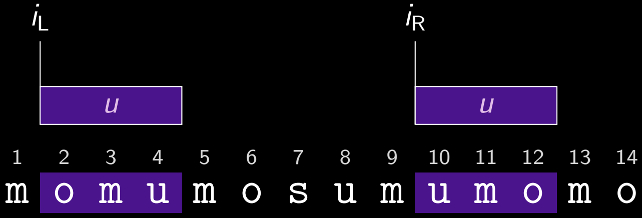
α-gapped repeats



i_L
$u + g \overset{?}{\leq} \alpha u$
i_R
u
g
u
1 2 3 4 5 6 7 8 9 10 11 12 13 14
m o m o m o s u m o m o m o

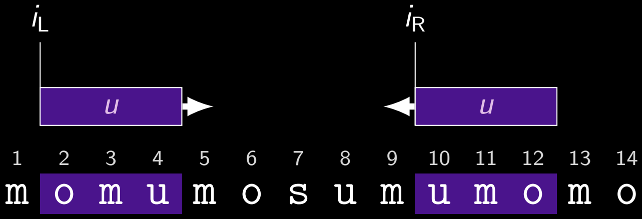
- maximal gapped repeat (i_L, i_R, u)
- $g :=$ gap

gapped palindromes



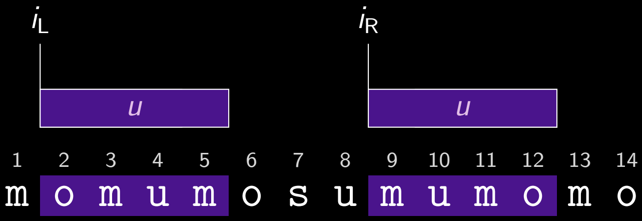
- (i_L, i_R, u) gapped palindrome

gapped palindromes



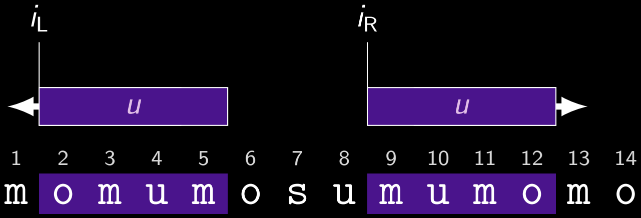
- (i_L, i_R, u) gapped palindrome
- is *maximal* if it cannot be extended
  - inwards nor

gapped palindromes



- (i_L, i_R, u) gapped palindrome
- is *maximal* if it cannot be extended
  - inwards nor

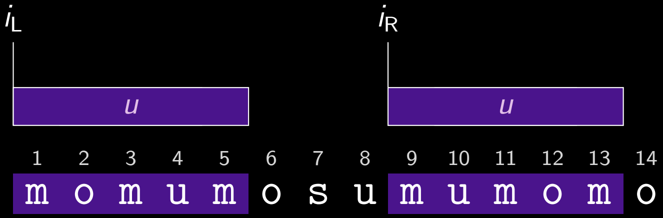
gapped palindromes



1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	o	m	u	m	o	s	u	m	u	m	o	m	o

- (i_L, i_R, u) gapped palindrome
- is *maximal* if it cannot be extended
  - inwards nor
  - outwards.

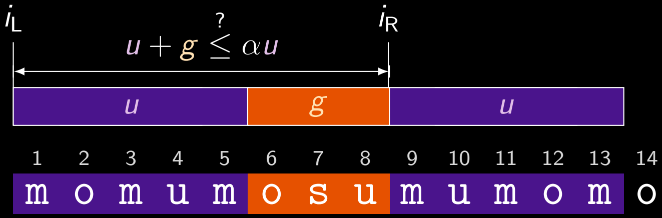
gapped palindromes



1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	o	m	u	m	o	s	u	m	u	m	o	m	o

- (i_L, i_R, u) gapped palindrome
- is *maximal* if it cannot be extended
    - inwards nor
    - outwards.

gapped palindromes



1	2	3	4	5	6	7	8	9	10	11	12	13	14
m	o	m	u	m	o	s	u	m	u	m	o	m	o

- (i_L, i_R, u) gapped palindrome
- is *maximal* if it cannot be extended
  - inwards nor
  - outwards.
- (i_L, i_R, u) is α*-gapped* if $g + u \leq \alpha u$

Definition

$$\left.\begin{matrix}\mathsf{occ}_{\mathcal{R}} \\ \mathsf{occ}_{\mathcal{P}}\end{matrix}\right\} := \#\ \text{occurrences of max. } \alpha\text{-gapped} \left\{\begin{matrix}\text{repeats} \\ \text{palindromes}\end{matrix}\right.$$

Problem

$$\mathsf{occ}_{\mathcal{R}}, \mathsf{occ}_{\mathcal{P}} \leq \,?$$

$\mathrm{occ}_{\mathcal{R}}$- repeats

$\mathcal{O}(\alpha^2 n)$	Kolpakov+'14
$\mathcal{O}(\alpha n)$	Crochemore+'15
$\leq 18\alpha n$	📄[1]
$\leq 13\alpha n$	📄[2]

📄[1] Gawrychowski, I, Inenaga, Köppl, Manea
Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes.
TOCS, **2018**

📄[2] I, Köppl
Improved upper bounds on all maximal α-gapped repeats and palindromes.
Accepted at TCS, **2018**

occ$_{\mathcal{P}}$- palindromes

$\leq 28\alpha n + 7n$ [1]

$\leq 16\alpha n - 3n$ [2]

[1] Gawrychowski, I, Inenaga, Köppl, Manea
Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes.
TOCS, **2018**

[2] I, Köppl
Improved upper bounds on all maximal α-gapped repeats and palindromes.
Accepted at TCS, **2018**

finding all maximal α-gapped repeats

previous results: Tanimura+'15,Crochemore+'15

$\mathcal{O}(\alpha n)$ time for constant σ.

finding all maximal α-gapped repeats

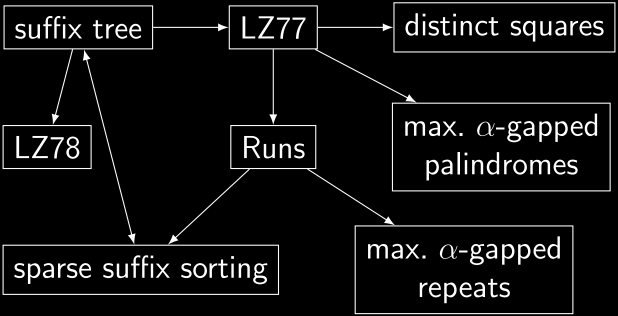
previous results: Tanimura+'15,Crochemore+'15

$\mathcal{O}(\alpha n)$ time for constant σ.

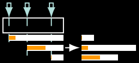
📄: same time with $\sigma = n^{\mathcal{O}(1)}$

📄 Gawrychowski, I, Inenaga, Köppl, Manea
Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes.
TOCS, **2018**

the big picture

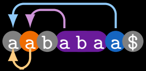

suffix tree
LZ77
distinct squares
LZ78
Runs
max. α-gapped palindromes
sparse suffix sorting
max. α-gapped repeats

summary



sparse suffix sorting of m suffixes

- $o(n)$ deterministic time if $c = o(n)$ and $m = o(n)$
- $\mathcal{O}(m)$ space



LZ 77/78 in $\mathcal{O}(n)$ time with

- $(1+\varepsilon)n \lg n + \mathcal{O}(n)$ bits
- $\mathcal{O}(n \lg \sigma)$ bits

LZ78 in $\mathcal{O}(n)$ time whp. and $\mathcal{O}(z \lg(z\sigma))$ bits + *practical!*

$\boxed{u}\boxed{u}$ finding all distinct squares

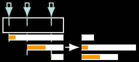
- $\mathcal{O}(n)$ time and $(2+\varepsilon)n \lg n$ bits
- online near-linear time

$\boxed{u}\boxed{g}\boxed{u}$ all maximal α-gapped repeats / palindromes

- find in $\mathcal{O}(\alpha n)$ time
- $\# = \mathcal{O}(\alpha n)$

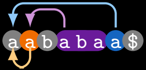
thank you for your attention!

summary



sparse suffix sorting of m suffixes

- $o(n)$ deterministic time if $c = o(n)$ and $m = o(n)$
- $\mathcal{O}(m)$ space



LZ 77/78 in $\mathcal{O}(n)$ time with

- $(1+\varepsilon)n \lg n + \mathcal{O}(n)$ bits
- $\mathcal{O}(n \lg \sigma)$ bits

LZ78 in $\mathcal{O}(n)$ time whp. and $\mathcal{O}(z \lg(z\sigma))$ bits + *practical!*

$\boxed{u}\boxed{u}$ finding all distinct squares

- $\mathcal{O}(n)$ time and $(2+\varepsilon)n \lg n$ bits
- online near-linear time

$\boxed{u}\boxed{g}\boxed{u}$ all maximal α-gapped repeats / palindromes

- find in $\mathcal{O}(\alpha n)$ time
- $\# = \mathcal{O}(\alpha n)$

string $T \in \Sigma^*$, $n := |T|$, $\sigma := |\Sigma| = n^{\mathcal{O}(1)}$, $0 < \varepsilon \leq 1$, z : # factors