Computing All Distinct Squares in Linear Time for Integer Alphabets

Hideo Bannai [1] Shunsuke Inenaga [1] *Dominik Köppl* [2]

[1]Department of Informatics, Kyushu University, Japan
[2]Department of Computer Science, TU Dortmund, Germany

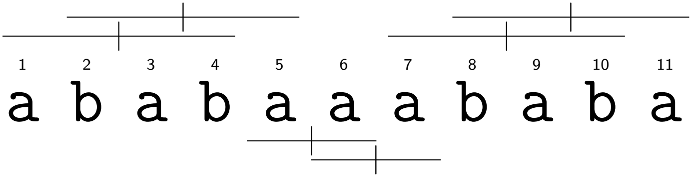
CPM'17

with python a one-liner: `[i**2 for i in range(1,n)]`

squares

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

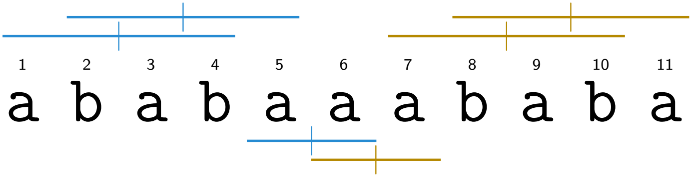
squares



squares

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

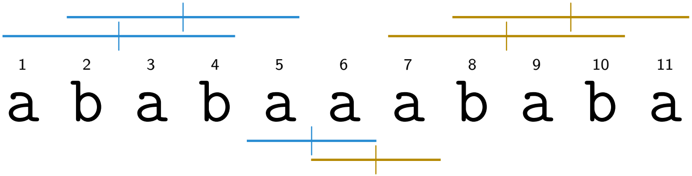
squares



leftmost squares

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

squares



leftmost squares

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

fact

leftmost squares $\equiv$ distinct squares

works on distinct squares

#distinct squares

$\leq 2n$	Fraenkel and Simpson'98
$\leq 2n - \Theta(\lg n)$	Ilie'07
$\leq \lfloor 11n/6 \rfloor$	Deza et al.'15
$\leq n$	(yet) unknown

algorithms computing all distinct squares

$\mathcal{O}(\sigma n)$ time	Gusfield and Stoye'04
$\mathcal{O}(n)$ time	Crochemore'14
$\mathcal{O}(n)$ time	this paper

where

- σ: alphabet size
- n: text length

setting

given

- text T
- $n := |T|$ text length
- alphabet of size $\sigma = n^{\mathcal{O}(1)}$

problem

find all distinct squares

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

naive solution

- create suffix array and LCP array
- iterate over each text position i
- iterate over all possible periods p
- compare $T[i..]$ with $T[i+p..]$ $\forall 1 \leq i, p \leq n$
- if found a square $\Rightarrow$ check whether already reported

$$\Rightarrow \mathcal{O}\left(\underbrace{n}_{\forall i} \cdot \underbrace{n}_{\forall p} \cdot t_\lambda\right)$$

- t_λ: time for look-up ($t_\lambda = n \lg \sigma$ for a simple trie)

better solutions

idea to get faster

- check only at certain text position
- check only periods up to a threshold

sufficient: all borders of Lempel-Ziv factors

idea from

[Gusfield and Stoye'04]

computing all distinct squares in $\mathcal{O}(\sigma n)$ time

Lempel-Ziv parsing

a b a b a a a b a b a

Lempel-Ziv parsing

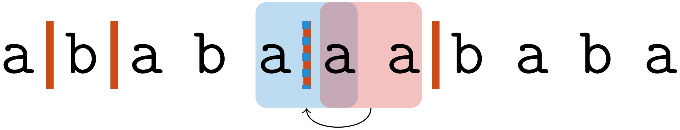
a b a b a a a b a b a

Lempel-Ziv parsing

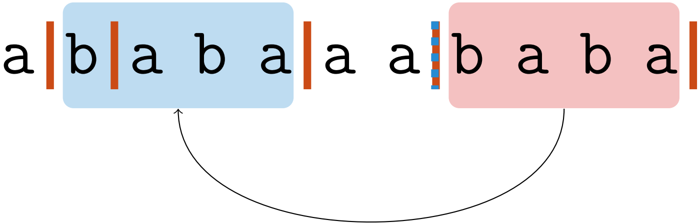
a | b | a b a a a b a b a

Lempel-Ziv parsing

Lempel-Ziv parsing



Lempel-Ziv parsing



Lempel-Ziv parsing

$$\overbrace{\mathtt{a}}^{F_1} \mid \overbrace{\mathtt{b}}^{F_2} \mid \overbrace{\mathtt{a\ b\ a}}^{F_3} \mid \overbrace{\mathtt{a\ a}}^{F_4} \mid \overbrace{\mathtt{b\ a\ b\ a}}^{F_5} \mid$$

computing squares

a b a b a a a b a b a

reported squares:

computing squares

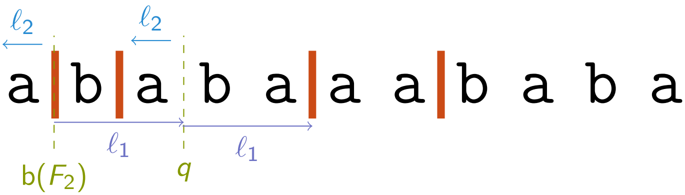
$$\overbrace{\texttt{a}}^{F_1} \,|\, \overbrace{\texttt{b}}^{F_2} \,|\, \overbrace{\texttt{a b a}}^{F_3} \,|\, \overbrace{\texttt{a a}}^{F_4} \,|\, \overbrace{\texttt{b a b a}}^{F_5}$$

reported squares:

computing squares

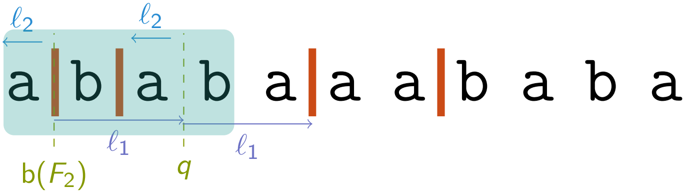
reported squares:

computing squares



reported squares:

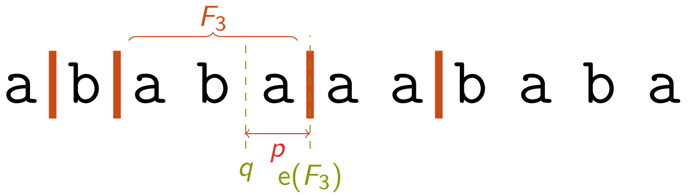
computing squares



reported squares:

- `abab`

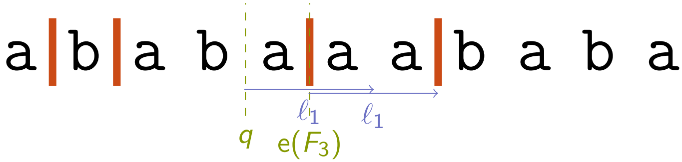
computing squares



reported squares:

- abab

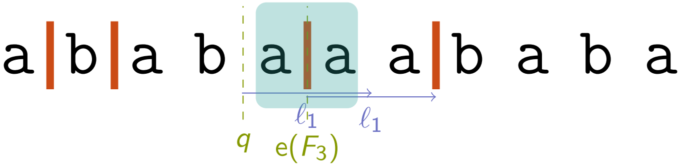
computing squares



reported squares:

- `abab`

computing squares



reported squares:

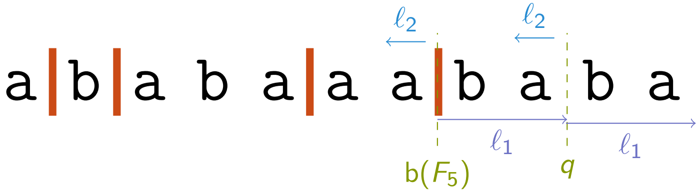
- abab
- aa

computing squares

reported squares:

- abab
- aa

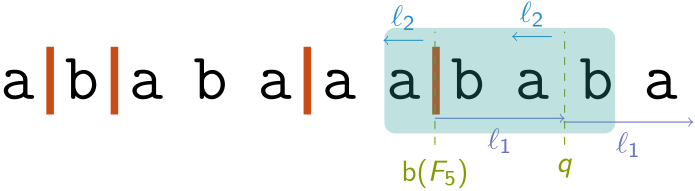
computing squares



reported squares:

- abab
- aa

computing squares



reported squares:

- abab
- aa
- abab

computing squares

a | b | a b a | a a | b a b a

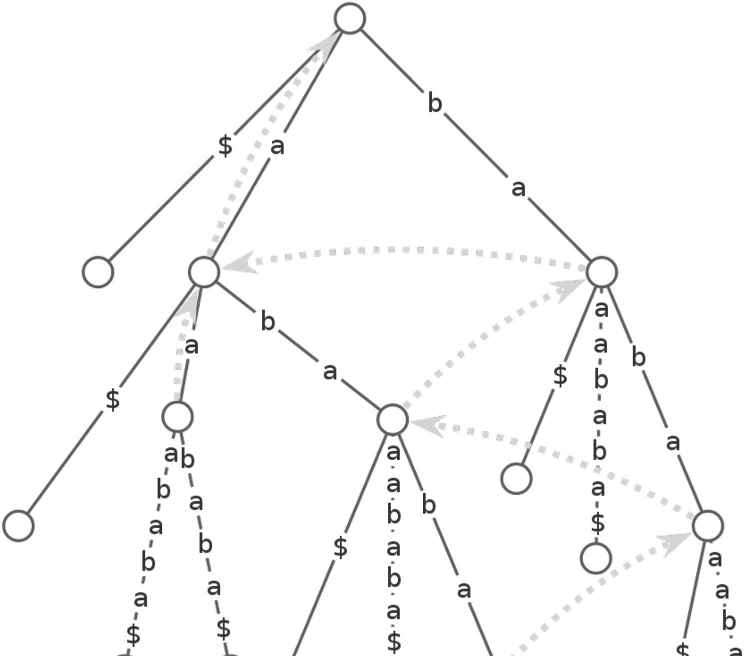
reported squares:

- abab
- aa
- abab

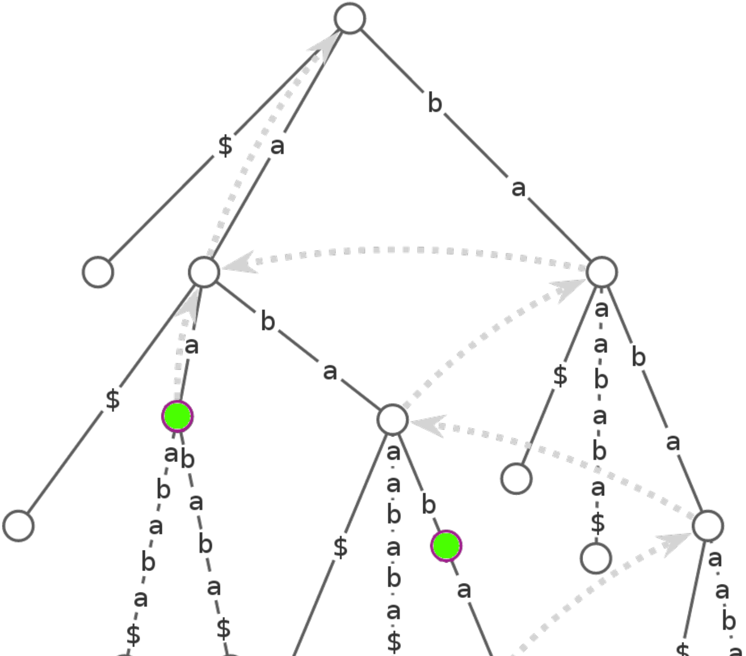
problems:

- reporting duplicates
- baba not found

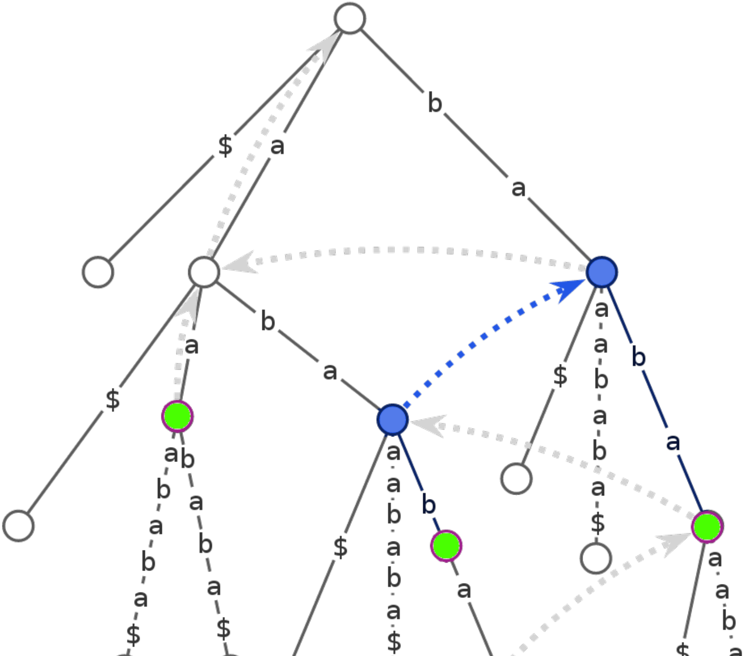
suffix link walk: construct suffix tree



suffix link walk: decorate squares



suffix link walk: rotate



time bounds

- suffix link walk traversals $\mathcal{O}(n)$ nodes
- number of suffix links to a particular node $\leq \sigma$
- $\Rightarrow$ a long traversal can happen $\mathcal{O}(\sigma)$ times

hence $\mathcal{O}(n\sigma)$ time [Gusfield and Stoye'04]

$\mathcal{O}(n)$ time goal

problem

- $\nexists$ dictionary with $\mathcal{O}(1)$ access/update time
- store lists, be careful about uniqueness!

solution

use longest previous factor table (LPF)!

longest previous factor table

a b a b a a a b a b a

longest previous factor table

0

a b a b a a a b a b a

longest previous factor table

0 0

a b a b a a a b a b a

longest previous factor table

0 0 3

a b a b a a a b a b a

longest previous factor table

0	0	3	2							
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0	0	3	2	1						
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0	0	3	2	1	2					
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0	0	3	2	1	2	5				
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0 0 3 2 1 2 5 4

a b a b a a a b a b a

longest previous factor table

0	0	3	2	1	2	5	4	3		
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0	0	3	2	1	2	5	4	3	2	
a	b	a	b	a	a	a	b	a	b	a

longest previous factor table

0	0	3	2	1	2	5	4	3	2	1
a	b	a	b	a	a	a	b	a	b	a

computing squares

0	0	3	2	1	2	5	4	3	2	1
a	b	a	b	a	a	a	b	a	b	a

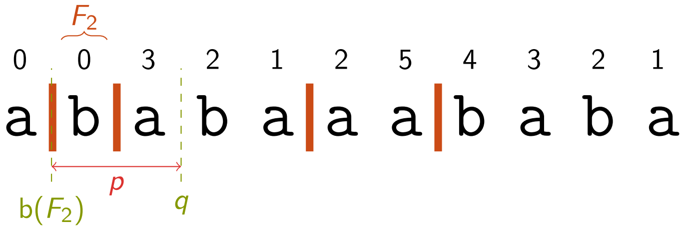
reported squares:

computing squares

$$\overbrace{\overset{0}{\texttt{a}}}^{F_1} \mid \overbrace{\overset{0}{\texttt{b}}}^{F_2} \mid \overbrace{\overset{3}{\texttt{a}}\ \overset{2}{\texttt{b}}\ \overset{1}{\texttt{a}}}^{F_3} \mid \overbrace{\overset{2}{\texttt{a}}\ \overset{5}{\texttt{a}}}^{F_4} \mid \overbrace{\overset{4}{\texttt{b}}\ \overset{3}{\texttt{a}}\ \overset{2}{\texttt{b}}\ \overset{1}{\texttt{a}}}^{F_5}$$

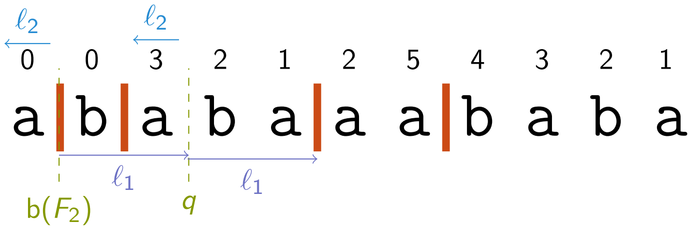
reported squares:

computing squares



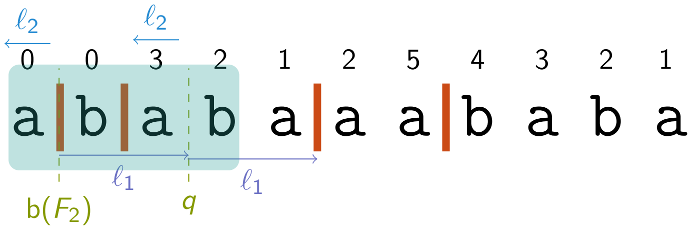
reported squares:

computing squares



reported squares:

computing squares



reported squares:

- abab

computing squares

0	0	3	2	1	2	5	4	3	2	1
a	b	a	b	a	a	a	b	a	b	a

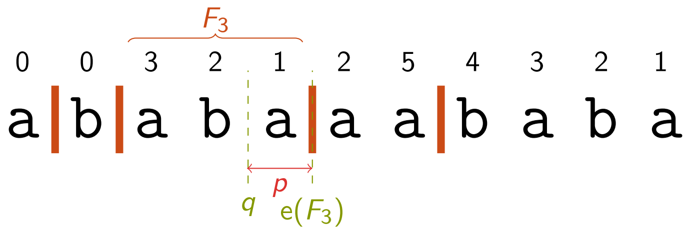
reported squares:

- `abab`
- `baba`

new techniques:

- right rotate found squares

computing squares



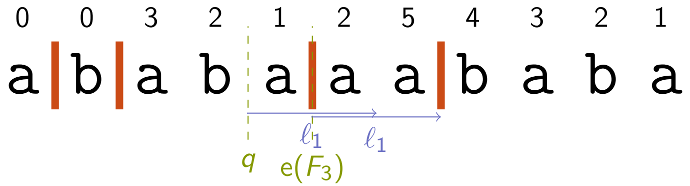
reported squares:

- abab
- baba

new techniques:

- right rotate found squares

computing squares



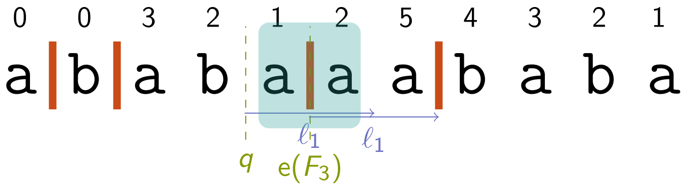
reported squares:

- abab
- baba

new techniques:

- right rotate found squares

computing squares



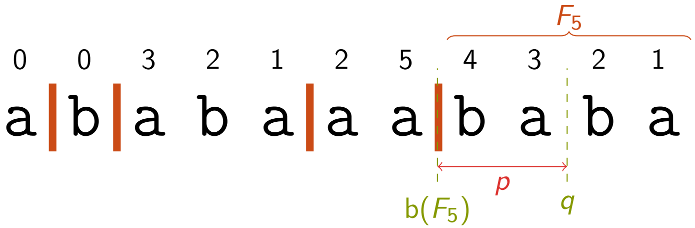
reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

computing squares



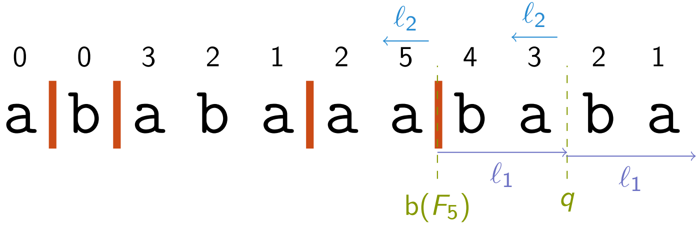
reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

computing squares



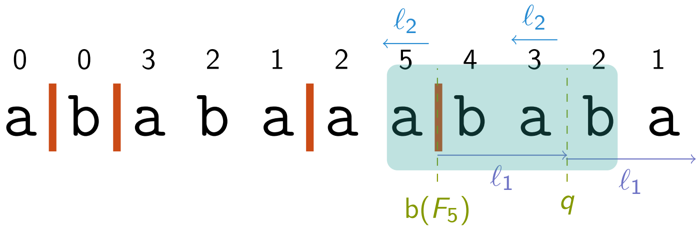
reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

computing squares



reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares
- skip if LPF[i] $\geq 2p$

RMQs on LPF

naive right rotation of square S takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

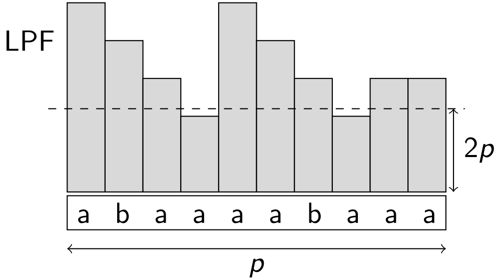
right rotations by RMQ on LPF

RMQs on LPF

naive right rotation of square S takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF

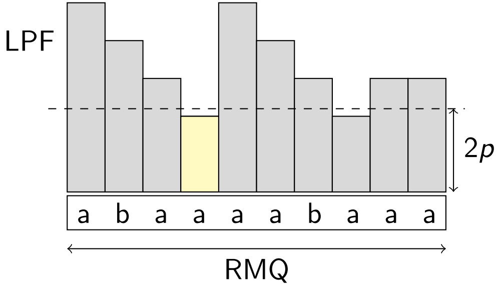


RMQs on LPF

naive right rotation of square S takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF

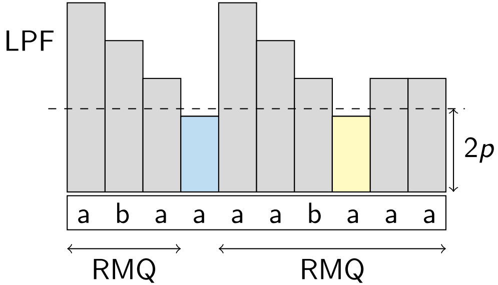


RMQs on LPF

naive right rotation of square S takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF

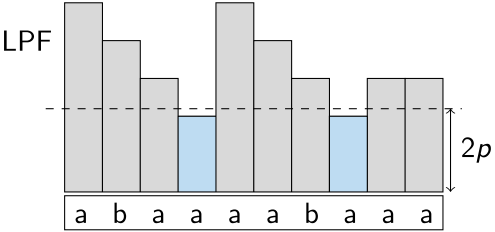


RMQs on LPF

naive right rotation of square S takes $\mathcal{O}(|S|)$ time
$\Rightarrow \mathcal{O}(n^2)$ total time!

our approach

right rotations by RMQ on LPF



$\#$RMQs $\leq 3\times$ number of newly found squares

time analysis

known:

- number of distinct squares $occ = \mathcal{O}(n)$

algorithm:	time
- build data structures (suffix array, etc.)	$\mathcal{O}(n)$
- query on $\mathcal{O}(n)$ positions:	
- for a square with LCP	$\mathcal{O}(1)$
- find right rotated squares with RMQ	$\mathcal{O}(occ)$ total

$\mathcal{O}(n + occ) = \mathcal{O}(n)$ total time

experiments

collection	σ	z	$\max_x \lvert F_x \rvert$	$\lvert \text{occ} \rvert$	time
`dblp.xml`	97	7,035,342	1060	7412	70
`proteins`	26	20,875,097	45,703	3,108,339	245
`dna`	17	13,970,040	97,966	132,594	310
`english`	226	13,971,134	987,766	13,408	2639
`einstein`	125	49,575	906,995	18,192,737	3953

- 200 MiB collections from Pizza&Chili corpus
- σ: alphabet size
- z: # Lempel-Ziv factors
- time in seconds

time bottleneck: RMQs

finding all distinct squares in linear time

techniques

- modification of [Gusfield and Stoye'04]
- using LPF array

open problems

- cope without RMQ
- create MAST in $\mathcal{O}(n)$ time

further results

- computing online in $\mathcal{O}\left(n \lg^2 \lg n / \lg \lg \lg n\right)$ time
- in linear time:
  - decorating suffix tree with information of all squares
  - building topology of the minimal augmented suffix tree (MAST)

finding all distinct squares in linear time

techniques

- modification of [Gusfield and Stoye'04]
- using LPF array

open problems

- cope without RMQ
- create MAST in $\mathcal{O}(n)$ time

further results

- computing online in $\mathcal{O}\left(n \lg^2 \lg n / \lg \lg \lg n\right)$ time
- in linear time:
  - decorating suffix tree with information of all squares
  - building topology of the minimal augmented suffix tree (MAST)

Thank you for listening. Any questions are welcome!