Computing All Distinct Squares in Linear Time for Integer Alphabets

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

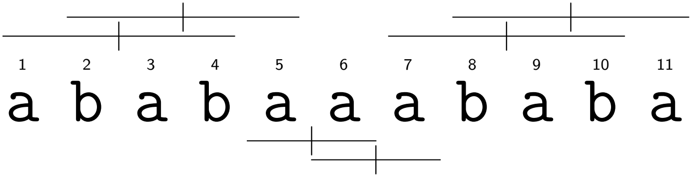
[1]Department of Informatics, Kyushu University, Japan
[2]Department of Computer Science, TU Dortmund, Germany

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

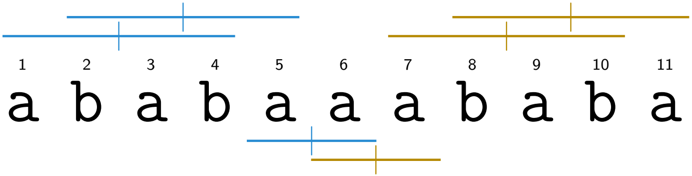
squares



squares

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

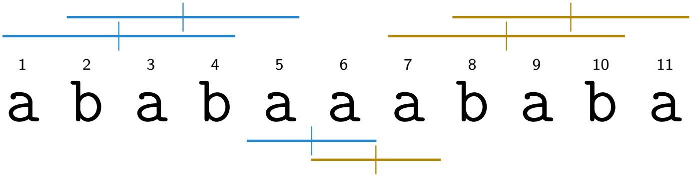
squares



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

squares = tandem repeats

GATCAATGAGGTGGTACATGCTAGTACACGCGAACACGCGA

tandem repeats

squares = tandem repeats

GATCAATGAGGTGGTACATGCTAGTACACGCGAACACGCGA

tandem repeats

- AA

squares = tandem repeats

GATCAATGAGGTGGTACATGCTAGTACACGCGAACACGCGA

tandem repeats

- AA
- GGT GGT

squares = tandem repeats

GATCAATGAGGTGGTACATGCTAGTACACGCGAACACGCGA

tandem repeats

- AA
- GGT GGT
- ACACGCGA ACACGCGA

squares = tandem repeats

GATCAATGAGGTGGTACATGCTAGTACACGCGAACACGCGA

tandem repeats

- AA
- GGT GGT
- ACACGCGA ACACGCGA

why?

- genetic fingerprint
- unterstand DNA better
- combinatorial (interesting)
- for compression?

setting

given

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

problem

find all distinct squares

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

naive solution

- iterate over each text position i
- iterate over all possible periods p
- compare $T[i+c] \stackrel{!}{=} T[i+p+c]\ \forall c = 0, \ldots, p-1$
- if found a square $\Rightarrow$ check whether already reported

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

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

naive solution

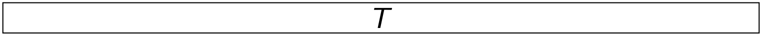
- iterate over each text position i
- iterate over all possible periods p
- compare $T[i+c] \stackrel{!}{=} T[i+p+c]\ \forall c = 0, \ldots, p-1$
- if found a square $\Rightarrow$ check whether already reported

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

- t_λ: time for look-up ($t_\lambda = n \lg \sigma$ for a simple trie)
- use LCP data structure to check characters in $\mathcal{O}(1)$ time

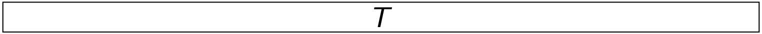
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time



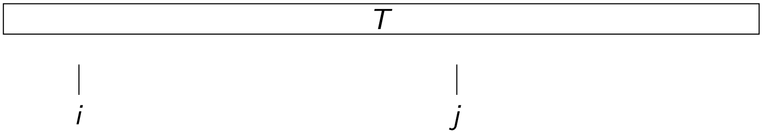
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$



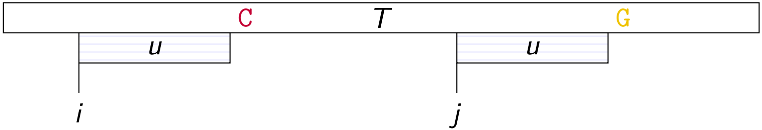
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\text{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$



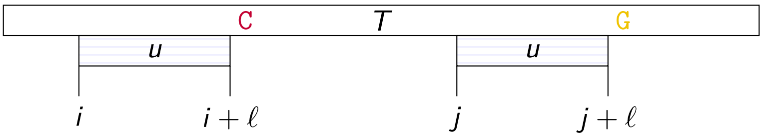
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathsf{lcp}(i, j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$



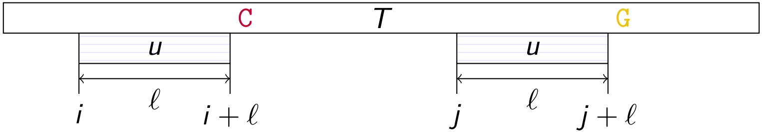
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$



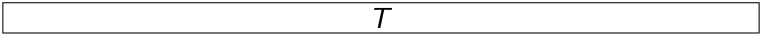
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathsf{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$



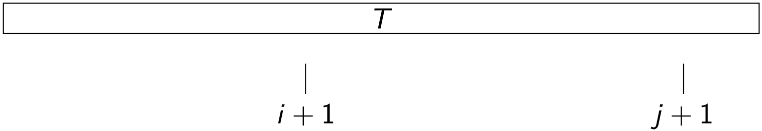
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$
  - ***longest common suffix***
    $\mathrm{lcs}(i,j) := \max\{\ell : T[i-\ell+1, i] = T[j-\ell+1, j]\}$



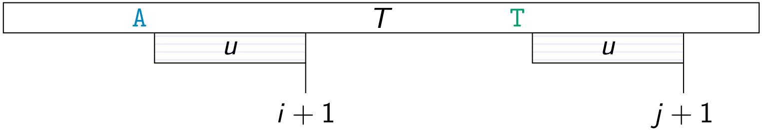
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$
  - ***longest common suffix***
    $\mathrm{lcs}(i,j) := \max\{\ell : T[i-\ell+1, i] = T[j-\ell+1, j]\}$



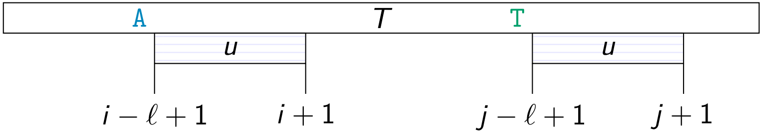
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$
  - ***longest common suffix***
    $\mathrm{lcs}(i,j) := \max\{\ell : T[i-\ell+1, i] = T[j-\ell+1, j]\}$



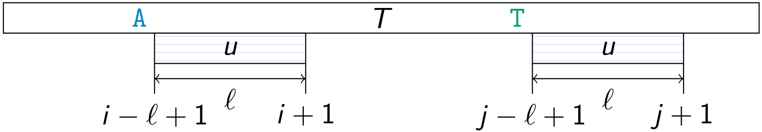
LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathrm{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$
  - ***longest common suffix***
    $\mathrm{lcs}(i,j) := \max\{\ell : T[i-\ell+1, i] = T[j-\ell+1, j]\}$



LCP data structure

- given string T
- construction in linear time
- answers in $\mathcal{O}(1)$ time
  - ***longest common prefix***
    $\mathsf{lcp}(i,j) := \max\{\ell : T[i, i+\ell-1] = T[j, j+\ell-1]\}$
  - ***longest common suffix***
    $\mathsf{lcs}(i,j) := \max\{\ell : T[i-\ell+1, i] = T[j-\ell+1, j]\}$



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

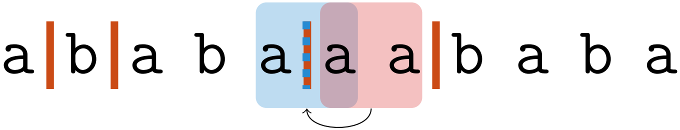
a | b a b a a a b a b a

Lempel-Ziv parsing

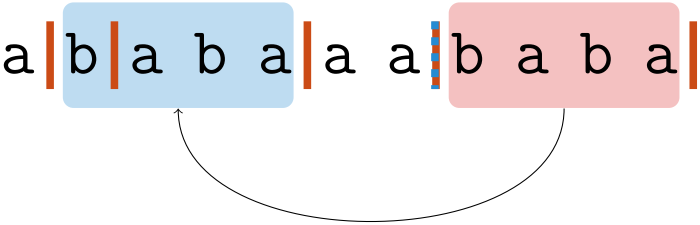
a | b | a b a a a b a b a

Lempel-Ziv parsing

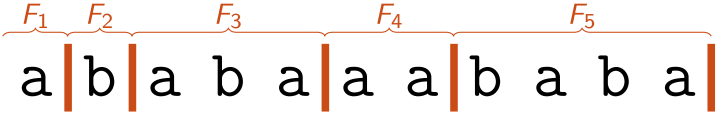
Lempel-Ziv parsing



Lempel-Ziv parsing



Lempel-Ziv parsing

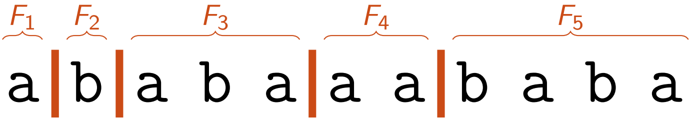


computing squares

a b a b a a a b a b a

reported squares:

computing squares

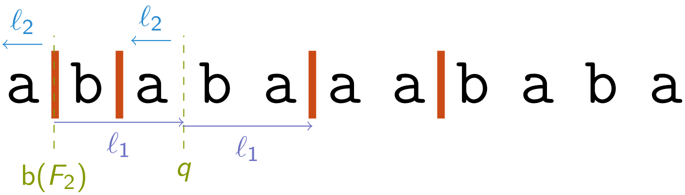


reported squares:

computing squares

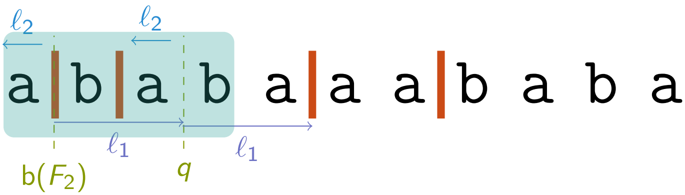
reported squares:

computing squares



reported squares:

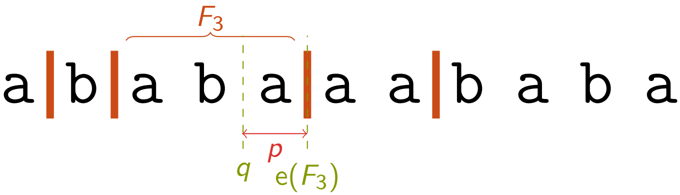
computing squares



reported squares:

- abab

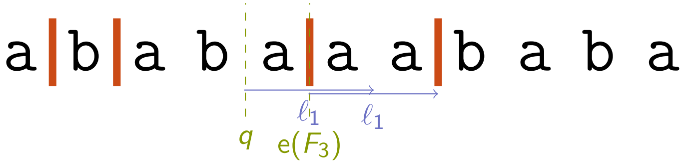
computing squares



reported squares:

- abab

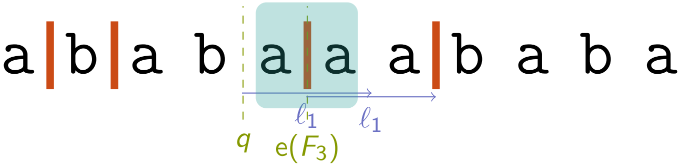
computing squares



reported squares:

- abab

computing squares



reported squares:

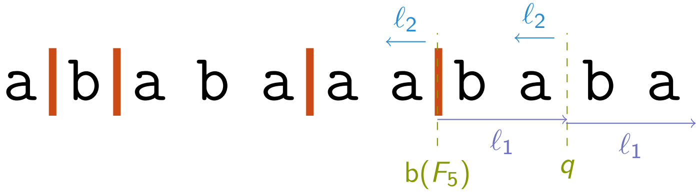
- abab
- aa

computing squares

reported squares:

- abab
- aa

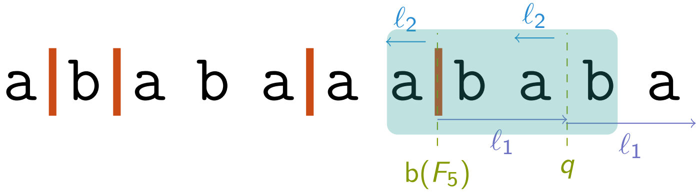
computing squares



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

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

problem

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

solution

use LPF array!

longest previous factor table

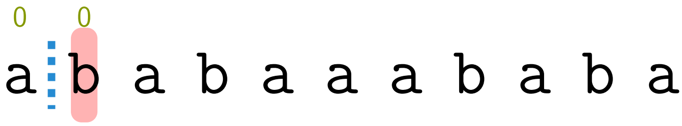
a b a b a a a b a b a

longest previous factor table

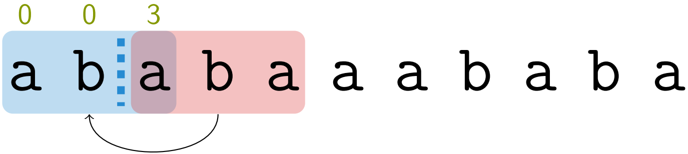
0

a b a b a a a b a b a

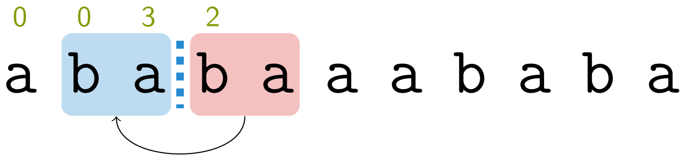
longest previous factor table



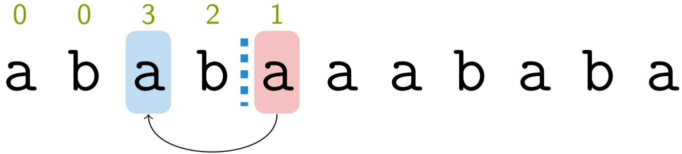
longest previous factor table



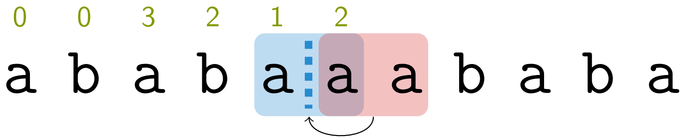
longest previous factor table



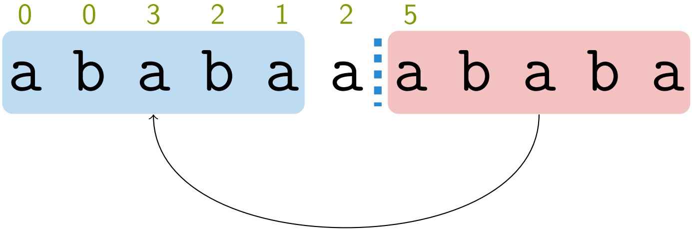
longest previous factor table



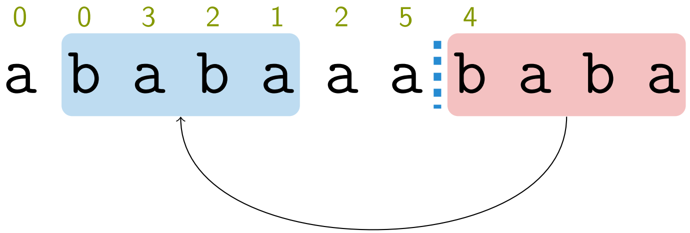
longest previous factor table



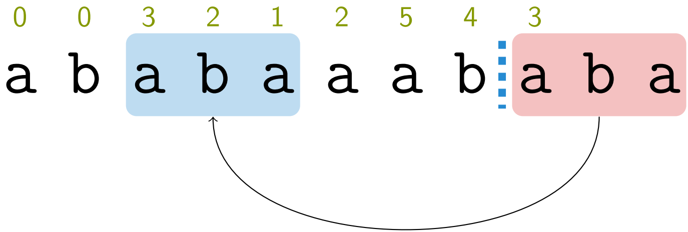
longest previous factor table



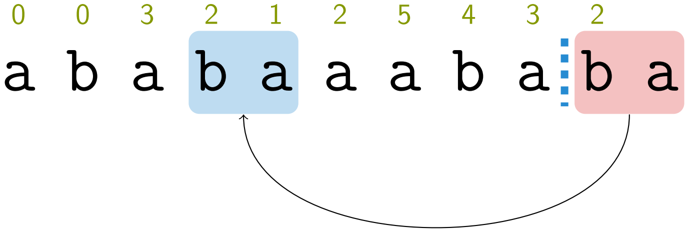
longest previous factor table



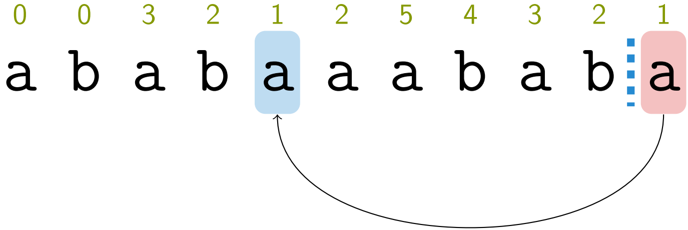
longest previous factor table



longest previous factor table



longest previous factor table

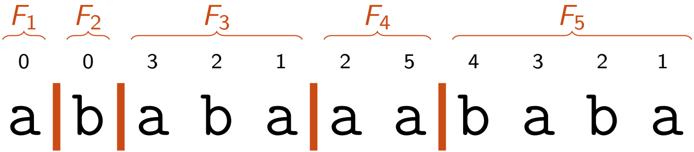


computing squares

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

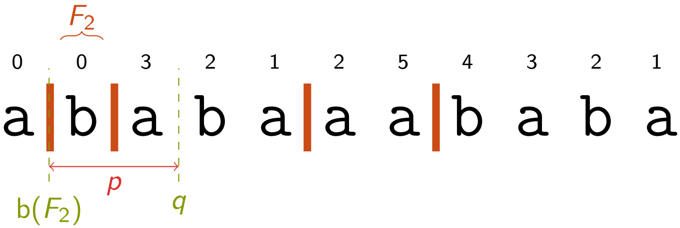
reported squares:

computing squares



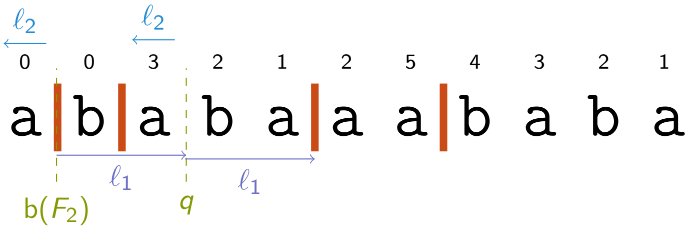
reported squares:

computing squares



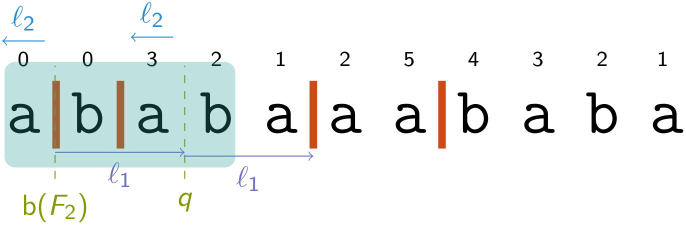
reported squares:

computing squares



reported squares:

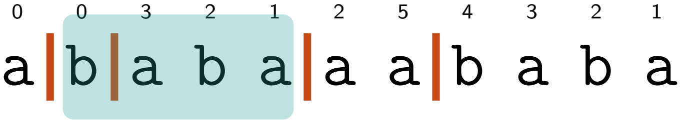
computing squares



reported squares:

- abab

computing squares



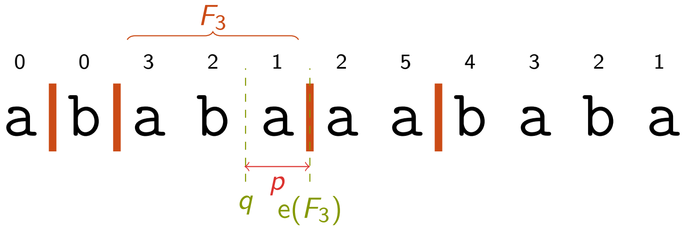
reported squares:

- abab
- baba

new techniques:

- right rotate found squares

computing squares



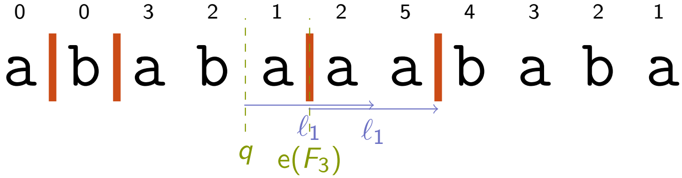
reported squares:

- abab
- baba

new techniques:

- right rotate found squares

computing squares



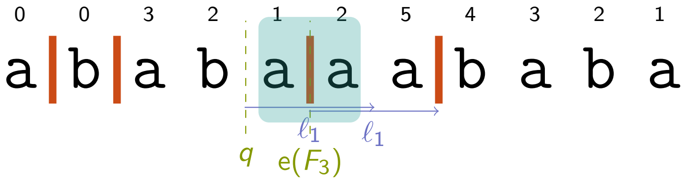
reported squares:

- abab
- baba

new techniques:

- right rotate found squares

computing squares



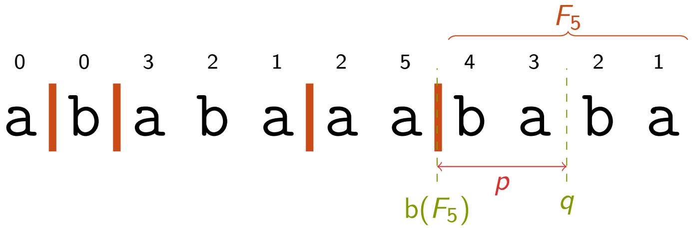
reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

computing squares



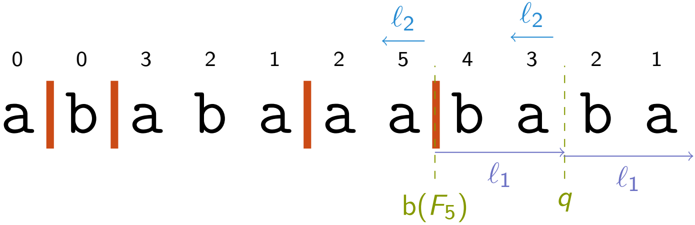
reported squares:

- abab
- baba
- aa

new techniques:

- right rotate found squares

computing squares



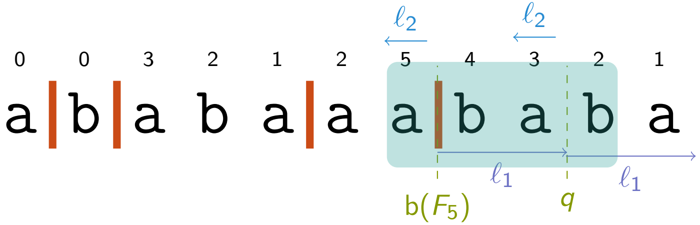
reported squares:

- 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] $> 2p$

experiments

collection	σ	z	$\max_x \lvert F_x \rvert$	$\lvert occ \rvert$	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

summary

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

techniques

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

further linear time results (read the paper!)

- decorating suffix tree with information of all squares
- building topology of the minimal augmented suffix tree (MAST)

open problems

- create MAST in $\mathcal{O}(n)$ time

summary

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

techniques

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

further linear time results (read the paper!)

- decorating suffix tree with information of all squares
- building topology of the minimal augmented suffix tree (MAST)

open problems

- create MAST in $\mathcal{O}(n)$ time

Thank you for listening. Any questions are welcome!