Online Detection of Repetitions with Backtracking

Dmitry Kosolobov

Ural Federal University Ekaterinburg, Russia

Introduction Catchers Repetition detectors

Repetitions

◆□ > ◆□ > ◆臣 > ◆臣 > ○

æ

Definitions

▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p

イロト イヨト イヨト イヨト

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w

э

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e

・ロト ・回ト ・ヨト ・ヨト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is e-repetition-free if w does not contain an e-repetition as a substring

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- ▶ w is e-repetition-free if w does not contain an e-repetition as a substring

Example text | minimal period | exponent

・ 同 ト ・ ヨ ト ・ ヨ ト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is e-repetition-free if w does not contain an e-repetition as a substring

Example

text	minimal period	exponent
abb · abb		

イロト イポト イヨト イヨト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is e-repetition-free if w does not contain an e-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	

(ロ) (同) (三) (三)

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is e-repetition-free if w does not contain an e-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2

イロト イポト イヨト イヨト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a		

イロト イポト イヨト イヨト

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	

(D) (A) (A) (A) (A)

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5

(D) (A) (A) (A) (A)

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab · ab · ab		

イロト イポト イヨト イヨト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab · ab · ab	2	

イロト イポト イヨト イヨト

Definitions

- ▶ p is a period of w if w[i] = w[i+p] for i = 0, 1, ..., |w|-1-p
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab∙ab∙ab	2	3

・ロト ・回ト ・ヨト ・ヨト

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab · ab · ab abbbb · abb	2	3

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab · ab · ab	2	3
abbbb · abb	5	

Definitions

- p is a period of w if w[i] = w[i+p] for $i = 0, 1, \dots, |w|-1-p$
- |w|/p is the exponent of w, where p is the minimal period of w
- w is an *e-repetition* if its exponent is greater than or equal to e
- w is *e-repetition-free* if w does not contain an *e*-repetition as a substring

Example

text	minimal period	exponent
abb · abb	3	2
ab · ab · a	2	2.5
ab · ab · ab	2	3
abbbb · abb	5	1.6

Introduction Catchers Repetition detectors

Problem and known solutions

イロト イヨト イヨト イヨト

æ

Introduction Catchers Repetition detectors

Problem and known solutions

Fix a rational e > 1

イロン イヨン イヨン イヨン

Fix a rational e > 1Maintain a string t (initially empty) under the following operations:

イロト イポト イヨト イヨト

```
Fix a rational e > 1
```

Maintain a string t (initially empty) under the following operations:

append(c): append a letter c to the right of t

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t

・ 同 ト ・ ヨ ト ・ ヨ ト

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

・ 同 ト ・ ヨ ト ・ ヨ ト

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

This structure is called *e-repetition detector*

< 同 > < 三 > < 三 >

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

This structure is called *e-repetition detector*

Results without backtracking (n is the number of operations)

A (1) > A (2) > A

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

This structure is called *e-repetition detector*

Results without backtracking (n is the number of operations)

Ω(n log n) unordered alphabet [Main, Lorentz 85]

・ 同 ト ・ ヨ ト ・ ヨ ト

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

This structure is called *e-repetition detector*

Results without backtracking (n is the number of operations)

- Ω(n log n) unordered alphabet [Main, Lorentz 85]
- ▶ $\Theta(n \log n) = 2$, unordered alphabet [Apostolico, Breslauer 96]

(D) (A) (A) (A) (A)

Fix a rational e > 1

Maintain a string t (initially empty) under the following operations:

- append(c): append a letter c to the right of t
- backtrack: remove the last letter of t
- repfree: check whether t is e-repetition-free

This structure is called *e-repetition detector*

Results without backtracking (n is the number of operations)

- Ω(n log n) unordered alphabet [Main, Lorentz 85]
- ▶ $\Theta(n \log n) = 2$, unordered alphabet [Apostolico, Breslauer 96]
- O(n log σ) ordered alphabet [Hong, Chen 08]
 (σ is the alphabet size)

Introduction Catchers Repetition detectors

Contribution

▲□▶ ▲圖▶ ▲≧▶ ▲≧▶

æ

Introduction Catchers Repetition detectors

Contribution

Fix a rational e > 1

・ロト ・ 日 ・ ・ 日 ・ ・ 日 ・ ・

æ

Contribution

Fix a rational e > 1

Theorem 1

For unordered alphabet, there is an *e*-repetition detector working in $O(n \log m)$ time and O(m) space, where *m* is the length of a longest string generated by a given sequence of *n* append and backtrack operations.

< 同 > < 三 > < 三 >

Contribution

Fix a rational e > 1

Theorem 1

For unordered alphabet, there is an *e*-repetition detector working in $O(n \log m)$ time and O(m) space, where *m* is the length of a longest string generated by a given sequence of *n* append and backtrack operations.

Theorem 2 (our method differs from [Hong, Chen 08])

For ordered alphabet, there is an *e*-repetition detector without backtracking working in $O(n \log \sigma)$ time and O(n) space, where *n* is the number of *append* operations and σ is the alphabet size.

(ロ) (同) (三) (三)

Introduction Catchers Repetition detectors

Catchers Backtracking in catchers

Catcher

Dmitry Kosolobov Online Detection of Repetitions with Backtracking 5 / 11

◆□▶ ◆圖▶ ◆理▶ ◆理▶ 「理」

Introduction Catchers Repetition detectors

Catchers Backtracking in catchers

Catcher

▶ fix a rational e > 1

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 臣 のへぐ

- ▶ fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*

イロン イヨン イヨン イヨン

- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$

イロト イポト イヨト イヨト

- ▶ fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* ($0 \le i \le j < |t|$)
- catcher searches the string t[i..j]

- ▶ fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* ($0 \le i \le j < |t|$)
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]

0

- \blacktriangleright fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers i and j ($0 \le i \le j < |t|$)
- catcher searches the string t[i..i]
- catcher finds e-repetitions using occurrences of t[i..j]

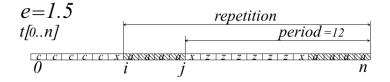
n

A (2) > (

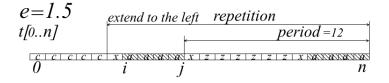
- ▶ fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds *e*-repetitions using occurrences of *t*[*i*..*j*]

$$e=1.5$$
 append(a)
 $t[0..n]$

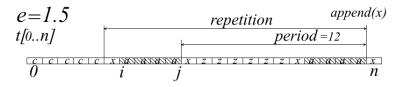
- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



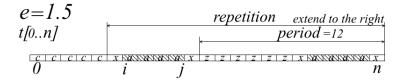
- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



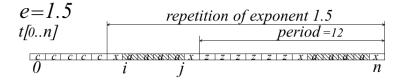
- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



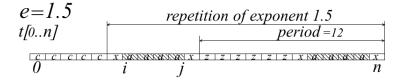
- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



- fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



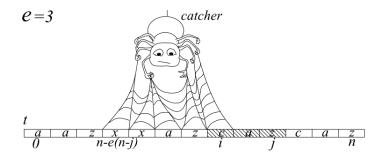
- ▶ fix a rational e > 1
- catcher "catches" *e*-repetitions in *t*
- catcher is defined by integers *i* and *j* $(0 \le i \le j < |t|)$
- catcher searches the string t[i..j]
- catcher finds e-repetitions using occurrences of t[i..j]



Lemma

Catchers Backtracking in catchers

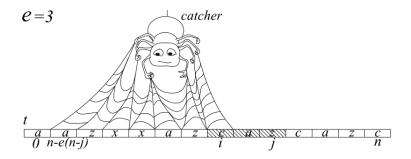
Catcher



Lemma

Catchers Backtracking in catchers

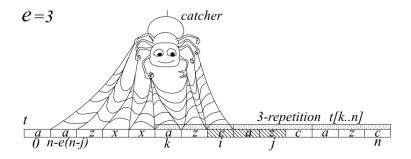
Catcher



Lemma

Catchers Backtracking in catchers

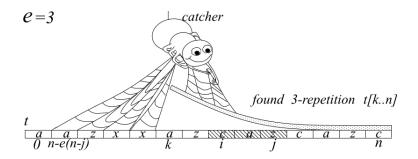
Catcher



Lemma

Catchers Backtracking in catchers

Catcher



Lemma

Catchers Backtracking in catchers

Backtracking

Dmitry Kosolobov Online Detection of Repetitions with Backtracking 6 / 11

・ロ・ ・聞・ ・ヨ・ ・ヨ・

æ

catcher uses constant space real-time string searching

イロン イヨン イヨン イヨン

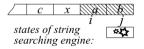
э

- catcher uses constant space real-time string searching
- catcher saves its states in an array

・ロト ・ 日 ・ ・ ヨ ・ ・ ヨ ・

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

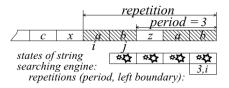
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

Catchers Backtracking in catchers

Backtracking

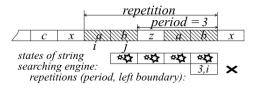
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

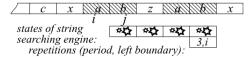


< 🗇 🕨 🖌 🚍 🕨

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

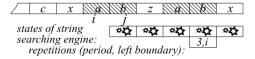


- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

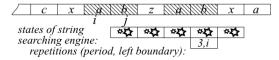


A (a) > (a) = (b) (a)

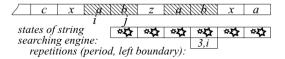
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



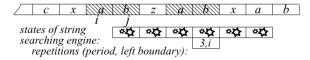
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



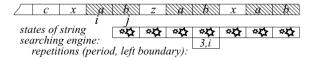
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

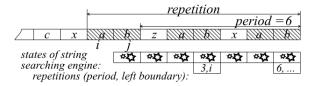


- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



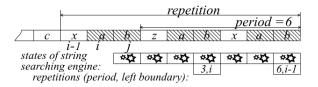
A (1) > A (2) > A

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

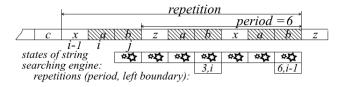


A (1) > (1) > (1)

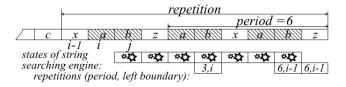
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

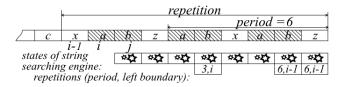


- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

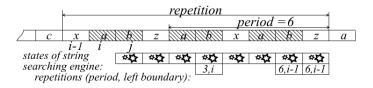


▲ □ ► < □ ► </p>

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

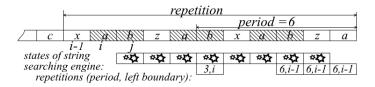


- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

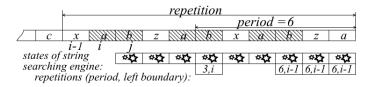


▲ □ ► < □ ► </p>

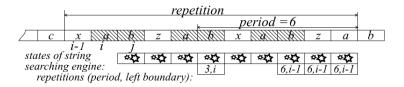
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



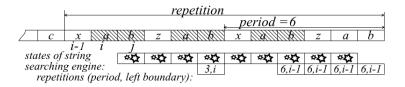
- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array

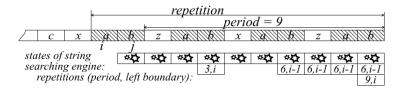


- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



▲ □ ► < □ ► </p>

- catcher uses constant space real-time string searching
- catcher saves its states in an array
- backtracking restores a previous state from that array



Catchers Backtracking in catchers

Time and space consumptions

イロン イヨン イヨン イヨン

臣

Lemma Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j - i + 1) \ge n - i$, then the states occupy O((c + 1)(n - i)) space.

Lemma Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j - i + 1) \ge n - i$, then the states occupy O((c + 1)(n - i)) space. *catcher occupies* O(j-i) *space if* n-j = O(j-i) O(j-i)*t i i i i n*

(日) (同) (三) (三)

Lemma Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j - i + 1) \ge n - i$, then the states occupy O((c + 1)(n - i)) space. *catcher occupies* O(j-i) *space if* n-j = O(j-i) O(j-i)*t n*

Lemma Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j - i + 1) \ge n - i$, then the states occupy O((c + 1)(n - i)) space. *catcher occupies* O(j-i) *space if* n-j = O(j-i) O(j-i)*t*

・ 同 ト ・ ヨ ト ・ ヨ ト

Lemma Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j - i + 1) \ge n - i$, then the states occupy O((c + 1)(n - i)) space. *catcher occupies O(j-i) space if n-j = O(j-i) O(j-i)* t 0 i j3-repitition n

・ロト ・ 同ト ・ ヨト ・ ヨト

Lemma

Fix c > 0. If t[0..|t| - 2] is *e*-repetition-free and $c(j-i+1) \ge n-i$, then the states occupy O((c+1)(n-i)) space.

catcher occupies
$$O(j-i)$$
 space if $n-j = O(j-i)$
t
 0 i j 3 -repitition n

Performing left extensions lazily, we can achieve O(c) time for each modification of the catcher (see details in the paper)

With backtracking Without backtracking

Idea of the detector with backtracking

・ロト ・ 日 ・ ・ ヨ ・ ・ ヨ ・

臣

Idea of the detector with backtracking Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Dmitry Kosolobov Online Detection of Repetitions with Backtracking 8 / 11

(D) (A) (A)

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

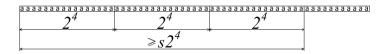
Fix an integer s depending on e (e.g. s = 3; details in the paper)



Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

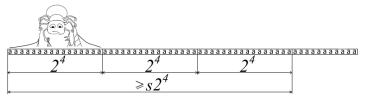


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)



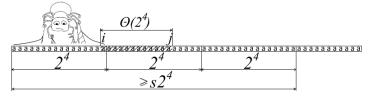
(本間) (本語) (本語)

Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)



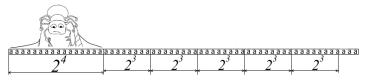
(D) (A) (A)

Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)



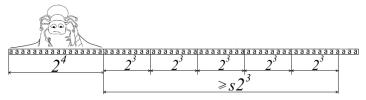
▲ □ ► < □ ► </p>

Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

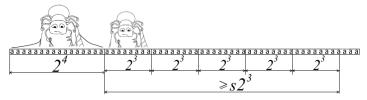


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

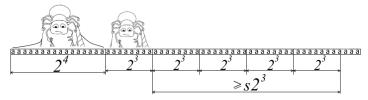


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)



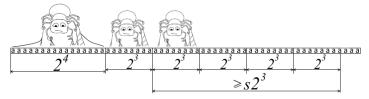
(D) (A) (A)

Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

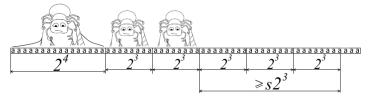


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

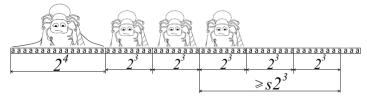


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

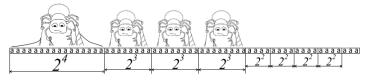


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

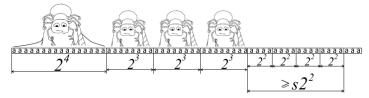


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

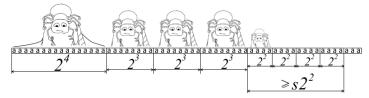


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

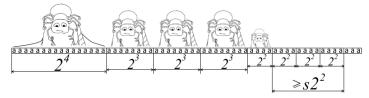


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

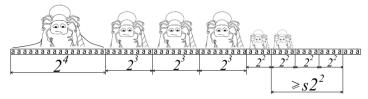


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

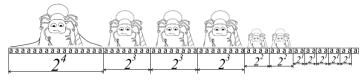


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

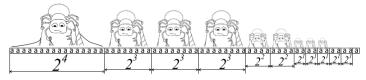


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

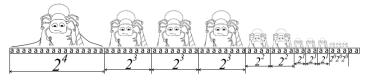


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)

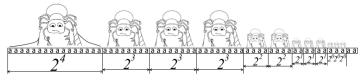


Idea of the detector with backtracking

Maintain $O(\log m)$ catchers "covering" the string t[0..m]

Example

Fix an integer s depending on e (e.g. s = 3; details in the paper)



- 1: check for *e*-repetitions of length $2, 3, \ldots, s-1$
- 2: for $(k \leftarrow 0; s2^k \le |t| \text{ and } |t| \mod 2^k = 0; k \leftarrow k + 1)$ do
- 3: **if** k > 0 **then**
- 4: remove two catchers "covering" $(|t|-s2^k..|t|-(s-1)2^k]$
- 5: create a catcher "covering" $(|t|-s2^k..|t|-(s-1)2^k]$

With backtracking Without backtracking

Time and space consumptions

Dmitry Kosolobov Online Detection of Repetitions with Backtracking 9/11

・ロト ・ 日 ・ ・ ヨ ・ ・ ヨ ・

臣

Space

A catcher "covering" a range of length 2^k occupies $O(2^k)$ space

(D) (A) (A)

Space

A catcher "covering" a range of length 2^k occupies $O(2^k)$ space All catchers occupy $O(\sum_{k=1}^{\log m} 2^k) = O(m)$ space

(D) (A) (A)

Space

A catcher "covering" a range of length 2^k occupies $O(2^k)$ space All catchers occupy $O(\sum_{k=1}^{\log m} 2^k) = O(m)$ space

Time

The modification of a catcher takes O(1) time

(D) (A) (A)

Space

A catcher "covering" a range of length 2^k occupies $O(2^k)$ space All catchers occupy $O(\sum_{k=1}^{\log m} 2^k) = O(m)$ space

Time

The modification of a catcher takes O(1) time The time for the creation of the catchers is amortized over the sequence of modifications (see details in the paper)

(ロ) (同) (三) (三)

Space

A catcher "covering" a range of length 2^k occupies $O(2^k)$ space All catchers occupy $O(\sum_{k=1}^{\log m} 2^k) = O(m)$ space

Time

The modification of a catcher takes O(1) time The time for the creation of the catchers is amortized over the sequence of modifications (see details in the paper) The overall time is $O(n \log m)$ (*m* is the length of a longest string generated by a given sequence of *n* appends and backtracks)

・ロト ・ 日 ・ ・ 日 ・ ・ 日 ・ ・

With backtracking Without backtracking

Idea of the detector without backtracking

・ロト ・ 日 ・ ・ ヨ ・ ・ ヨ ・

æ

Idea of the detector without backtracking

Lemma

Let *u* be the shortest unioccurrent suffix of *t* and *r* be an *e*-repetition of *t*. If t[0..|t|-2] is *e*-repetition-free, then $|u| \leq |r| < \frac{e}{e-1}|u|$.

・ 同 ト ・ ヨ ト ・ ヨ ト

Idea of the detector without backtracking

Lemma

Let *u* be the shortest unioccurrent suffix of *t* and *r* be an *e*-repetition of *t*. If t[0..|t|-2] is *e*-repetition-free, then $|u| \le |r| < \frac{e}{e-1}|u|$.

Unioccurrent suffixes can be found online (without backtracking)

A (2) > (

Idea of the detector without backtracking

Lemma

Let *u* be the shortest unioccurrent suffix of *t* and *r* be an *e*-repetition of *t*. If t[0..|t|-2] is *e*-repetition-free, then $|u| \le |r| < \frac{e}{e-1}|u|$.

Unioccurrent suffixes can be found online (without backtracking) Three catchers can efficiently cover the range described in Lemma (see details in the paper)

< 同 > < 三 > < 三 >

With backtracking Without backtracking

Thank you for your attention!

Dmitry Kosolobov Online Detection of Repetitions with Backtracking11 / 11