Online Detection of Repetitions with Backtracking

Dmitry Kosolobov

Ural Federal University
Ekaterinburg, Russia

Repetitions

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$

Repetitions

Definitions

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

Repetitions

Definitions

- p is a *period of w* if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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

Repetitions

Definitions

- p is a *period of w* if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$		

Repetitions

Definitions

- p is a *period of w* if $w[i] = w[i{+}p]$ for $i = 0, 1, \ldots, |w|{-}1{-}p$
- $|w|/p$ 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
$abb \cdot abb$	3	

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$		

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i{+}p]$ for $i = 0, 1, \ldots, |w|{-}1{-}p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$		

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$	2	

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i{+}p]$ for $i = 0, 1, \ldots, |w|{-}1{-}p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$	2	3

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i{+}p]$ for $i = 0, 1, \ldots, |w|{-}1{-}p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$	2	3
$abbbb \cdot abb$		

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$	2	3
$abbbb \cdot abb$	5	

Repetitions

Definitions

- p is a *period of* w if $w[i] = w[i+p]$ for $i = 0, 1, \ldots, |w|-1-p$
- $|w|/p$ 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
$abb \cdot abb$	3	2
$ab \cdot ab \cdot a$	2	2.5
$ab \cdot ab \cdot ab$	2	3
$abbbb \cdot abb$	5	1.6

Problem and known solutions

Problem and known solutions

Fix a rational $e > 1$

Problem and known solutions

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

Problem and known solutions

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

Problem and known solutions

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

Problem and known solutions

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

Problem and known solutions

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*

Problem and known solutions

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)

Problem and known solutions

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)

- $\Omega(n \log n)$ unordered alphabet [Main, Lorentz 85]

Problem and known solutions

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)

- $\Omega(n \log n)$ unordered alphabet [Main, Lorentz 85]
- $\Theta(n \log n)$ $e{=}2$, unordered alphabet [Apostolico, Breslauer 96]

Problem and known solutions

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)

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

Contribution

Contribution

Fix a rational $e > 1$

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.

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.

Catcher

Catcher

- fix a rational $e > 1$

Catcher

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

Catcher

- 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

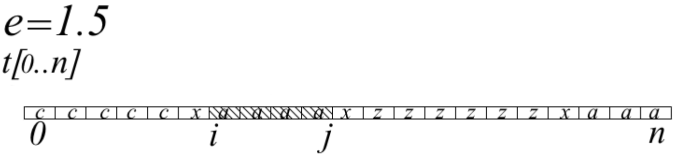
- fix a rational $e > 1$
- catcher "catches" e-repetitions in t
- catcher is defined by integers i and j ($0 \leq i \leq j < |t|$)
- catcher searches the string $t[i..j]$

Catcher

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

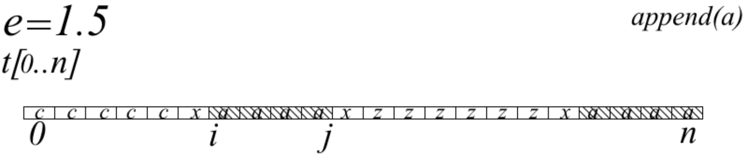
Catcher

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



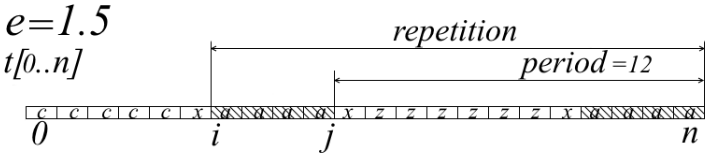
Catcher

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



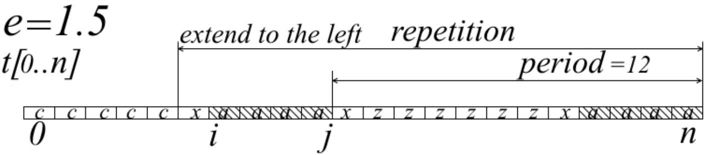
Catcher

- 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 $t[i..j]$



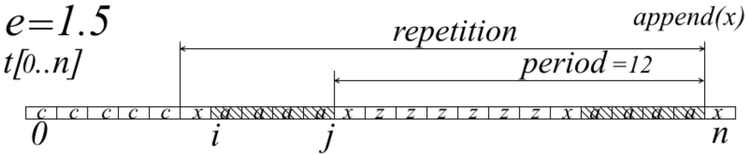
Catcher

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



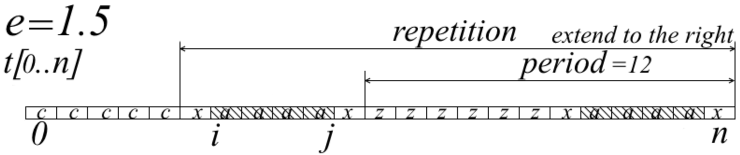
Catcher

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



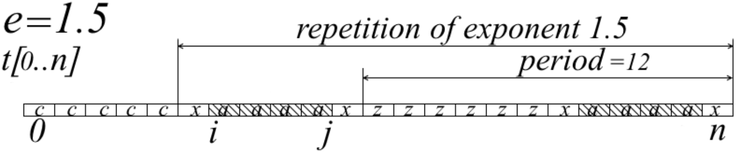
Catcher

- 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 $t[i..j]$



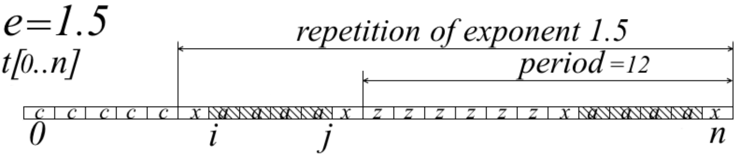
Catcher

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



Catcher

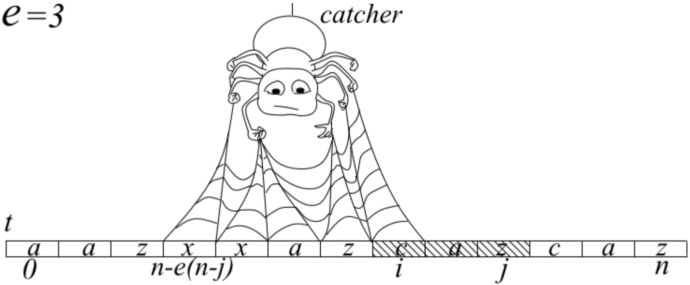
- fix a rational $e > 1$
- catcher "catches" e-repetitions in t
- catcher is defined by integers i and j ($0 \leq i \leq j < |t|$)
- catcher searches the string $t[i..j]$
- catcher finds e-repetitions using occurrences of $t[i..j]$



Lemma

If $t[k..n]$ is an e-repetition and $n - e(n-j) \leq k \leq i$, the catcher finds this repetition.

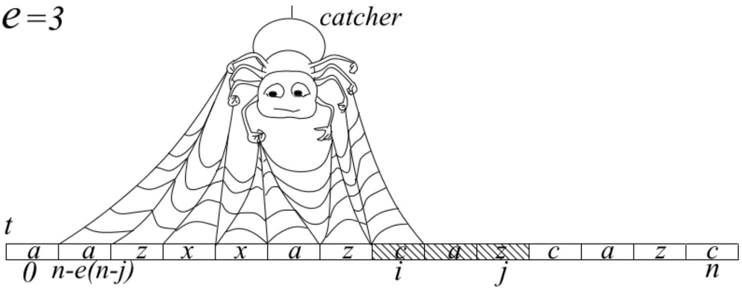
Catcher



Lemma

If $t[k..n]$ is an e-repetition and $n - e(n-j) \leq k \leq i$, the catcher finds this repetition.

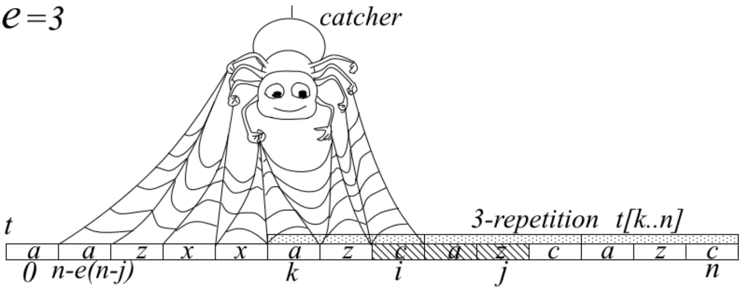
Catcher



Lemma

If $t[k..n]$ is an e-repetition and $n - e(n{-}j) \leq k \leq i$, the catcher finds this repetition.

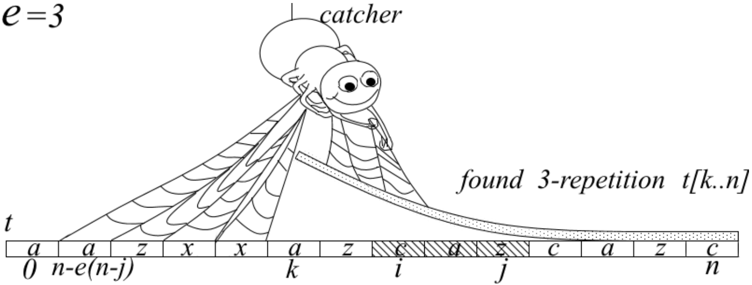
Catcher



Lemma

If $t[k..n]$ is an e-repetition and $n - e(n{-}j) \leq k \leq i$, the catcher finds this repetition.

Catcher



Lemma

If $t[k..n]$ is an e-repetition and $n - e(n{-}j) \leq k \leq i$, the catcher finds this repetition.

Backtracking

Backtracking

- catcher uses constant space real-time string searching

Backtracking

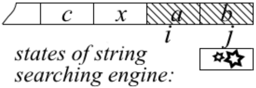
- catcher uses constant space real-time string searching
- catcher saves its states in an array

Backtracking

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

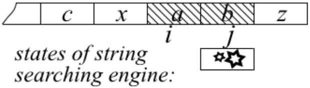
Backtracking

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



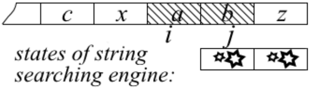
Backtracking

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



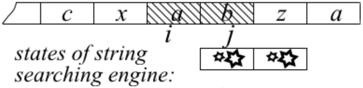
Backtracking

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



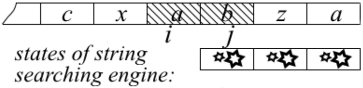
Backtracking

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



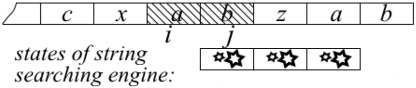
Backtracking

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



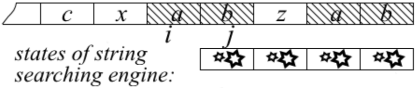
Backtracking

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



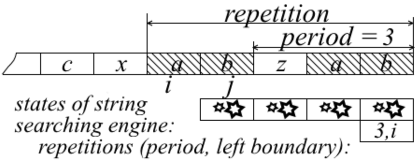
Backtracking

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



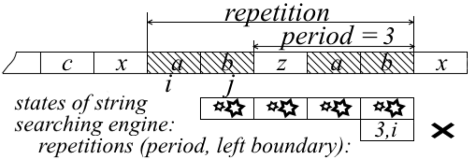
Backtracking

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



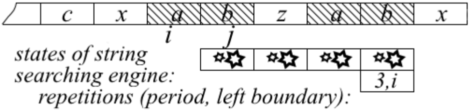
Backtracking

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



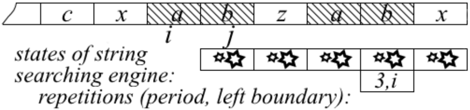
Backtracking

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



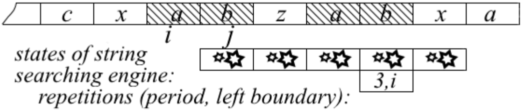
Backtracking

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



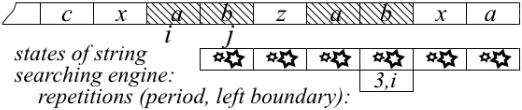
Backtracking

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



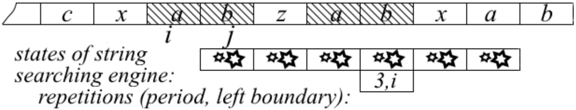
Backtracking

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



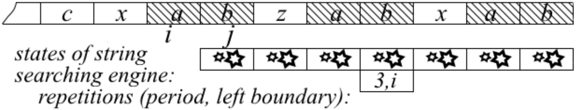
Backtracking

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



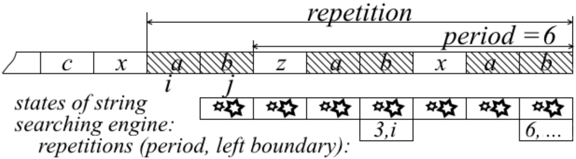
Backtracking

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



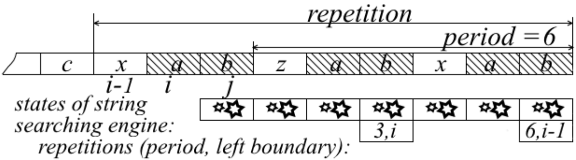
Backtracking

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



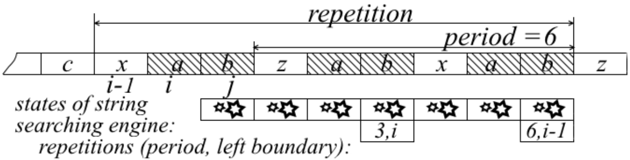
Backtracking

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



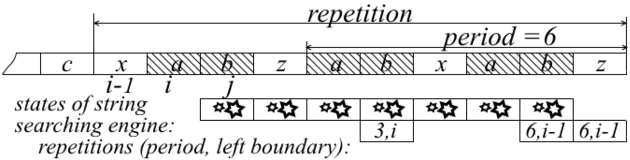
Backtracking

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



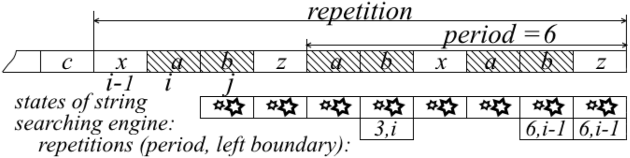
Backtracking

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



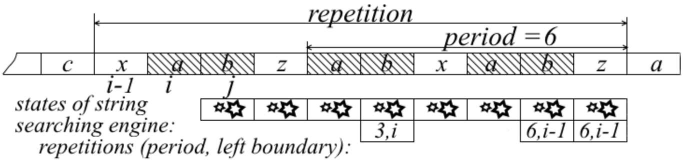
Backtracking

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



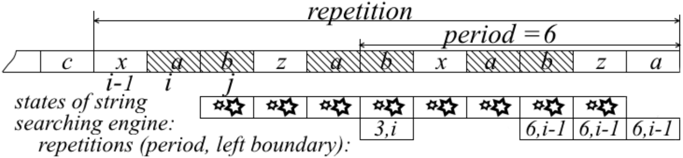
Backtracking

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



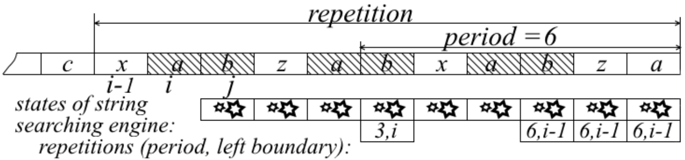
Backtracking

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



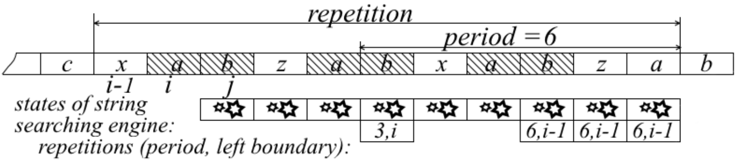
Backtracking

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



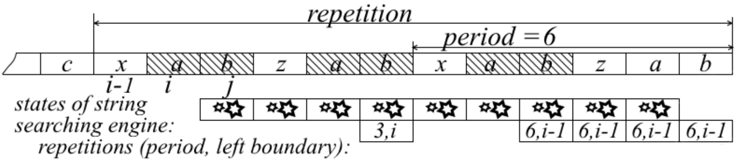
Backtracking

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



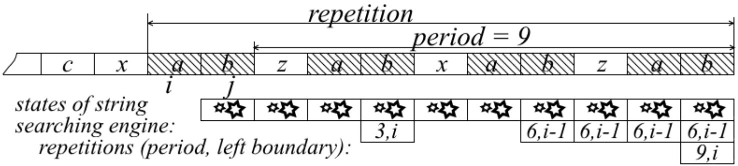
Backtracking

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



Backtracking

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



Time and space consumptions

Time and space consumptions

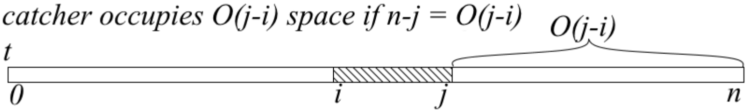
Lemma

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

Time and space consumptions

Lemma

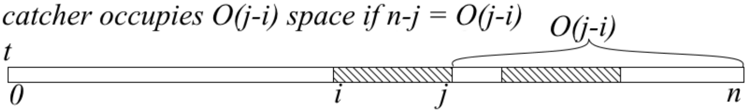
Fix $c > 0$. If $t[0..|t|-2]$ is e-repetition-free and $c(j-i+1) \geq n-i$, then the states occupy $O((c+1)(n-i))$ space.



Time and space consumptions

Lemma

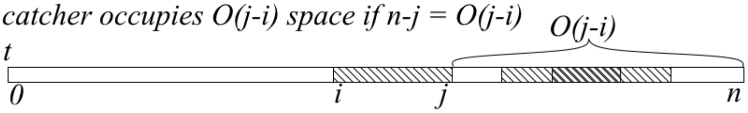
Fix $c > 0$. If $t[0..|t| - 2]$ is e-repetition-free and $c(j - i + 1) \geq n - i$, then the states occupy $O((c + 1)(n - i))$ space.



Time and space consumptions

Lemma

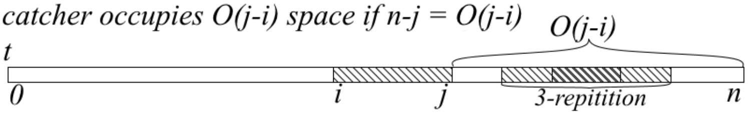
Fix $c > 0$. If $t[0..|t|-2]$ is e-repetition-free and $c(j-i+1) \geq n-i$, then the states occupy $O((c+1)(n-i))$ space.



Time and space consumptions

Lemma

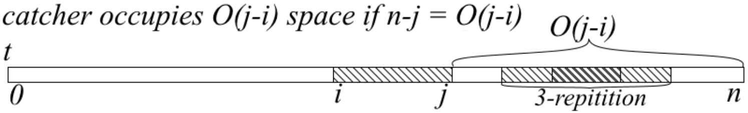
Fix $c > 0$. If $t[0..|t|-2]$ is e-repetition-free and $c(j-i+1) \geq n-i$, then the states occupy $O((c+1)(n-i))$ space.



Time and space consumptions

Lemma

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



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

Idea of the detector with backtracking

Idea of the detector with backtracking

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

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)

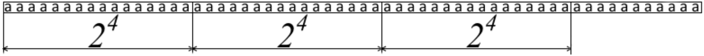
a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a

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)

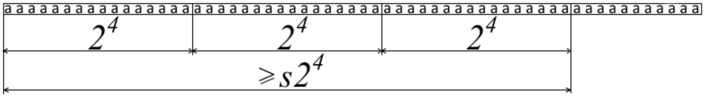


Idea of the detector with backtracking

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)

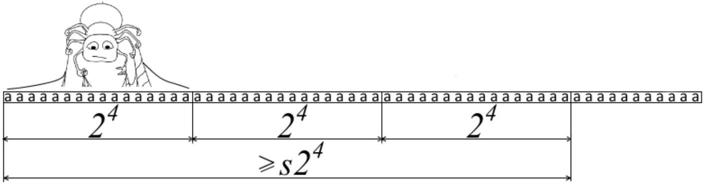


Idea of the detector with backtracking

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)

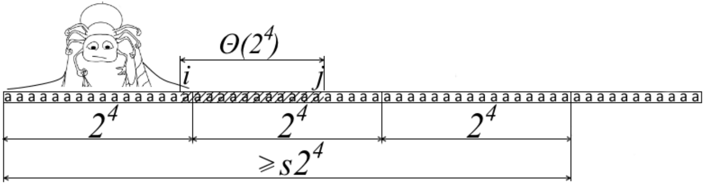


Idea of the detector with backtracking

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)

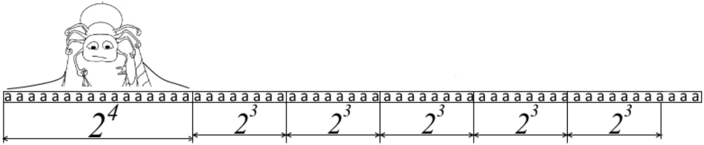


Idea of the detector with backtracking

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)

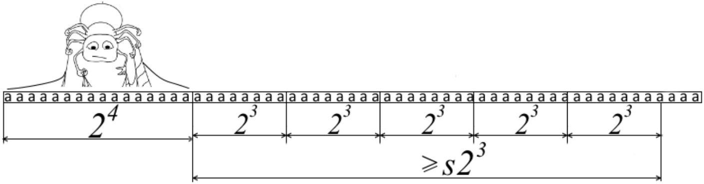


Idea of the detector with backtracking

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)

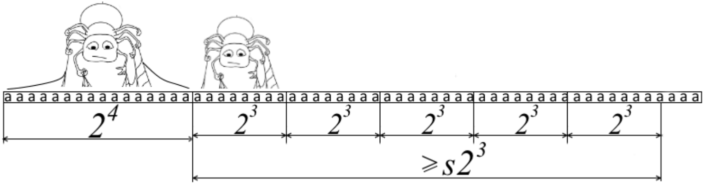


Idea of the detector with backtracking

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)

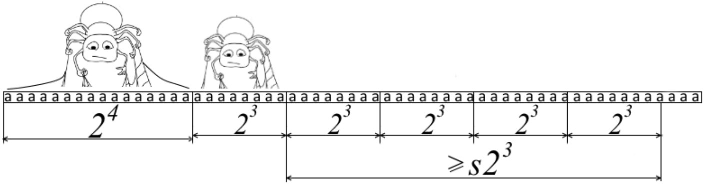


Idea of the detector with backtracking

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)

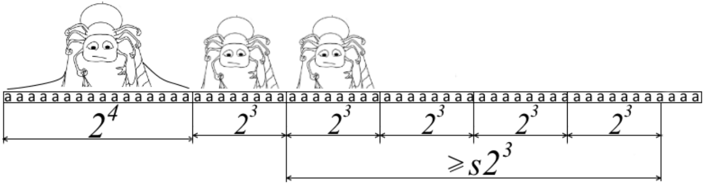


Idea of the detector with backtracking

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)

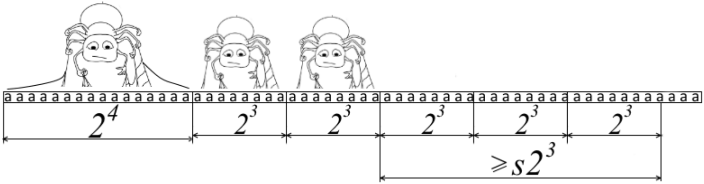


Idea of the detector with backtracking

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)

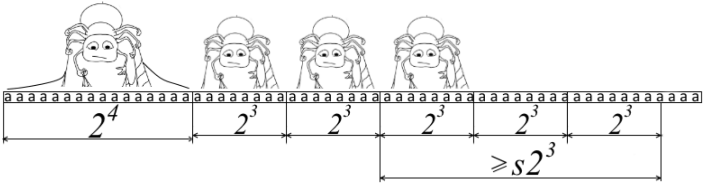


Idea of the detector with backtracking

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)

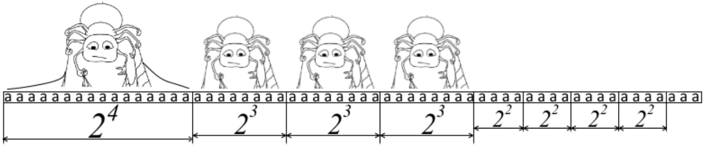


Idea of the detector with backtracking

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)

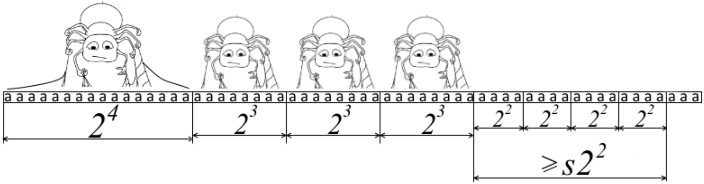


Idea of the detector with backtracking

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)

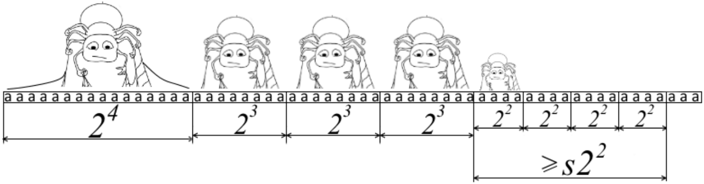


Idea of the detector with backtracking

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)

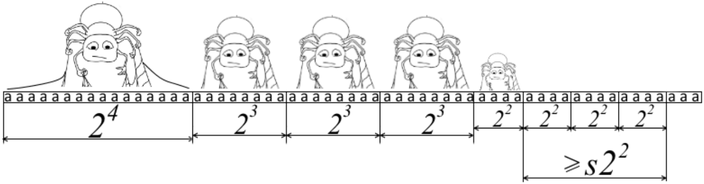


Idea of the detector with backtracking

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)

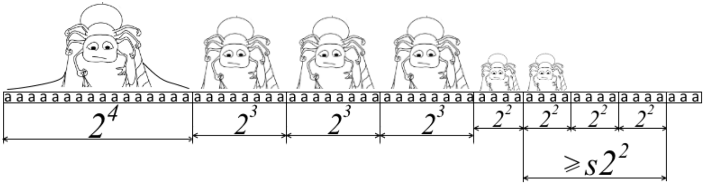


Idea of the detector with backtracking

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)

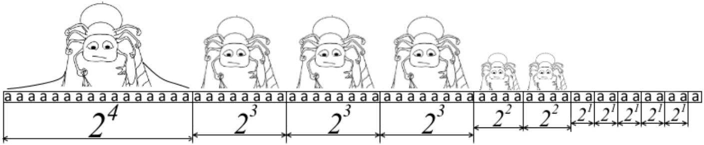


Idea of the detector with backtracking

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)

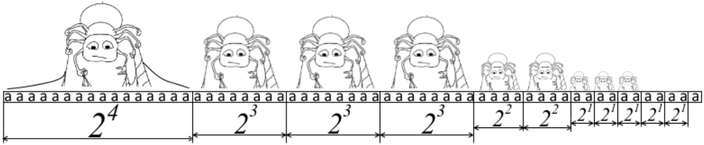


Idea of the detector with backtracking

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)

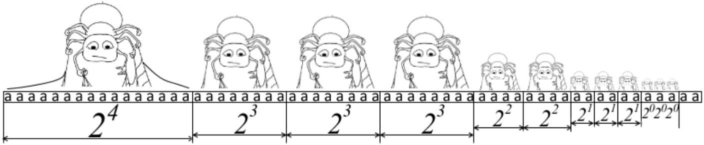


Idea of the detector with backtracking

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)

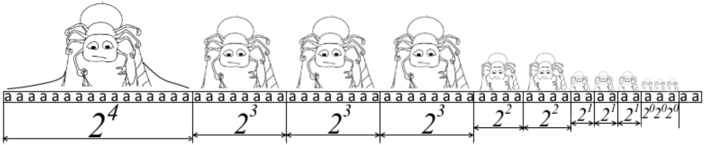


Idea of the detector with backtracking

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)



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

Time and space consumptions

Time and space consumptions

Space

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

Time and space consumptions

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 and space consumptions

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

Time and space consumptions

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)

Time and space consumptions

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*)

Idea of the detector without backtracking

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 $|u| \leq |r| < \frac{e}{e-1}|u|$.

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 $|u| \leq |r| < \frac{e}{e-1}|u|$.

Unioccurrent suffixes can be found online (without backtracking)

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 $|u| \leq |r| < \frac{e}{e-1}|u|$.

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

Thank you for your attention!