Validating the Knuth-Morris-Pratt failure function, fast and online

Paweł Gawrychowski [1] Łukasz Jeż [1] Artur Jeż [1]

University of Wrocław, Poland

5th International Computer Science Symposium in Russia
Kazan, 18 VI 2010

Knuth-Morris-Pratt function (π')

- Morris-Pratt failure function: in first linear-time pattern-recognition algorithm
- for $w[1\,..\,n]$ defined as

$$\pi_w[i] = \text{ the largest } k < i \text{ such that } w[1\,..\,k] = w[i-k+1\,..\,i]$$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$w[i]$	a	a	b	a	a	b	a	a	a	b	a	a	b	a	a	c
$\pi_w[i]$	0	1	0	1	2	3	4	5	2	3	4	5	6	7	8	0
$\pi'_w[i]$	-1	1	-1	-1	1	-1	-1	5	1	-1	-1	2	-1	-1	8	0

- important in word combinatorics

Knuth-Morris-Pratt function (π')

- Morris-Pratt failure function: in first linear-time pattern-recognition algorithm
- for $w[1\,..\,n]$ defined as

$$\pi_w[i] = \text{ the largest } k < i \text{ such that } w[1\,..\,k] = w[i-k+1\,..\,i]$$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$w[i]$	a	a	b	a	a	b	a	a	a	b	a	a	b	a	a	c
$\pi_w[i]$	0	1	0	1	2	3	4	5	2	3	4	5	6	7	8	0
$\pi'_w[i]$	-1	1	-1	-1	1	-1	-1	5	1	-1	-1	2	-1	-1	8	0

- important in word combinatorics
- Knuth-Morris-Pratt function π' defined as $\pi'_w[i] =$ the largest $k < i$ such that
  - $w[1\,..\,k] = w[i-k+1\,..\,i]$
  - $w[k+1] \neq w[i+1]$
- (assume additionally $\pi'[n] = \pi[n]$)

Validation

Problem

Input: integer array A'

Output: Is there w *such that* $\pi'_w = A'$*?*

- *size of the alphabet*
- *the word* w
- *. . .*

Validation

Problem

Input: integer array A'

Output: Is there w *such that* $\pi'_w = A'$*?*

- *size of the alphabet*
- *the word* w
- *. . .*

Reverse engineering, algorithmic characterisation, validation,

Validation

Problem

Input: integer array A'
Output: Is there w such that $\pi'_w = A'$?

- *size of the alphabet*
- *the word w*
- *. . .*

Reverse engineering, algorithmic characterisation, validation, done for

- Morris-Pratt failure function by Franěk et al.[2002]
- parametrised border array by Tomohiro I et al. [LATA 2009, CPM 2010]
- prefix array by Clément et al. [STACS 2009]
- cover array by Crochemore et al. [CPM 2010]
- for a similar function by Duval et al. [2007]

History

- for π: a standard exercise
- Franěk et al. gave linear offline solution
- later improved to linear online solution

History

- for π: a standard exercise
- Franěk et al. gave linear offline solution
- later improved to linear online solution

Definition (Online algorithm)

The algorithm is online, if it reads input value by value and provides a proper answer after each letter of input.

History

- for π: a standard exercise
- Franěk et al. gave linear offline solution
- later improved to linear online solution

Definition (Online algorithm)

The algorithm is online, if it reads input value by value and provides a proper answer after each letter of input.

- this may cripple the running time

History

- for π: a standard exercise
- Franěk et al. gave linear offline solution
- later improved to linear online solution

Definition (Online algorithm)

The algorithm is online, if it reads input value by value and provides a proper answer after each letter of input.

- this may cripple the running time

Why online?

- linear time transformation between π and π', inherently offline
- related problems studied in online scenario
- only online problem is challenging

History

- for π: a standard exercise
- Franěk et al. gave linear offline solution
- later improved to linear online solution

Definition (Online algorithm)

The algorithm is online, if it reads input value by value and provides a proper answer after each letter of input.

- this may cripple the running time

Why online?

- linear time transformation between π and π', inherently offline
- related problems studied in online scenario
- only online problem is challenging

Similar result

- studied for a related problem ($g[n] = \pi'[n-1] + 1$), offline, the linear transformation not applicable, quadratic lower bound

Our approach

- for π the online solution is known
- recreate (some) π from π'
- validate π using known algorithm

Our approach

- for π the online solution is known
- recreate (some) π from π'
- validate π using known algorithm

Possible problems

- do it online
- there are many such π's
- choose wise (maximal one)
  - fast update
  - good properties
  - no backtrack

Our approach

- for π the online solution is known
- recreate (some) π from π'
- validate π using known algorithm

Possible problems

- do it online
- there are many such π's
- choose wise (maximal one)
  - fast update
  - good properties
  - no backtrack

Result

Knuth-Morris-Pratt failure function can be validated online in $\mathcal{O}(n)$ time

Our approach

- for π the online solution is known
- recreate (some) π from π'
- validate π using known algorithm

Possible problems

- do it online
- there are many such π's
- choose wise (maximal one)
  - fast update
  - good properties
  - no backtrack

Result

Knuth-Morris-Pratt failure function can be validated online in $\mathcal{O}(n)$ time (minimal size of the alphabet, word w).

Consistent functions

Definition (consistent functions)

$A[1\,..\,n+1]$ is consistent with $A'[1\,..\,n]$ iff there is a word $w[1\,..\,n+1]$ such that

- $A[1\,..\,n+1] = \pi_w[1\,..\,n+1]$,
- $A'[1\,..\,n] = \pi'_w[1\,..\,n]$.

Consistent functions

Definition (consistent functions)

$A[1\,..\,n+1]$ is consistent with $A'[1\,..\,n]$ iff there is a word $w[1\,..\,n+1]$ such that

- $A[1\,..\,n+1] = \pi_w[1\,..\,n+1]$,
- $A'[1\,..\,n] = \pi'_w[1\,..\,n]$.

Definition (maximal consistent function)

$A[1\,..\,n+1]$ is the maximal consistent function if for every consistent $B[1\,..\,n+1]$, $B[1\,..\,n+1] \leq A[1\,..\,n+1]$.

Consistent functions

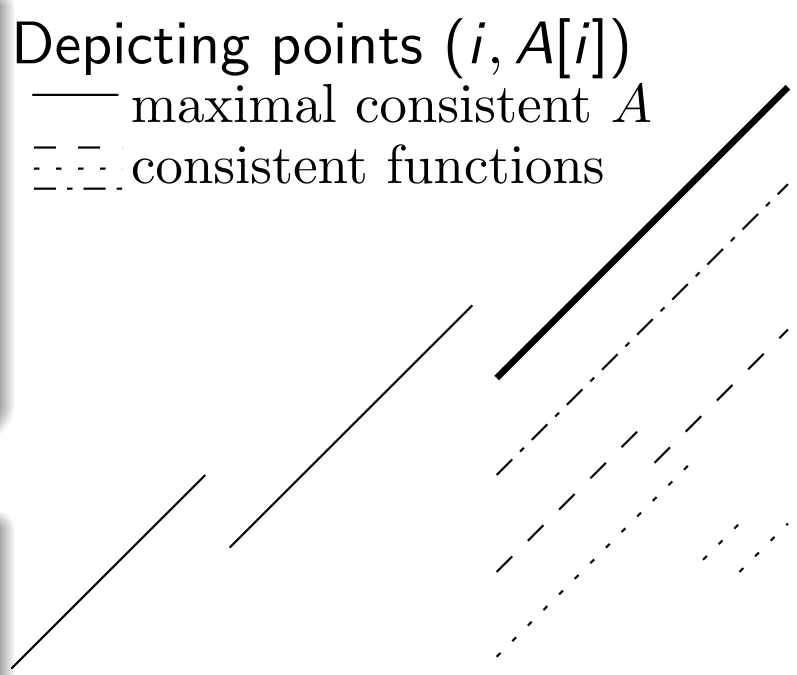
Definition (consistent functions)

$A[1\,..\,n+1]$ is consistent with $A'[1\,..\,n]$ iff there is a word $w[1\,..\,n+1]$ such that

- $A[1\,..\,n+1] = \pi_w[1\,..\,n+1]$,
- $A'[1\,..\,n] = \pi'_w[1\,..\,n]$.

Definition (maximal consistent function)

$A[1\,..\,n+1]$ is the maximal consistent function if for every consistent $B[1\,..\,n+1]$, $B[1\,..\,n+1] \leq A[1\,..\,n+1]$.



Consistent functions

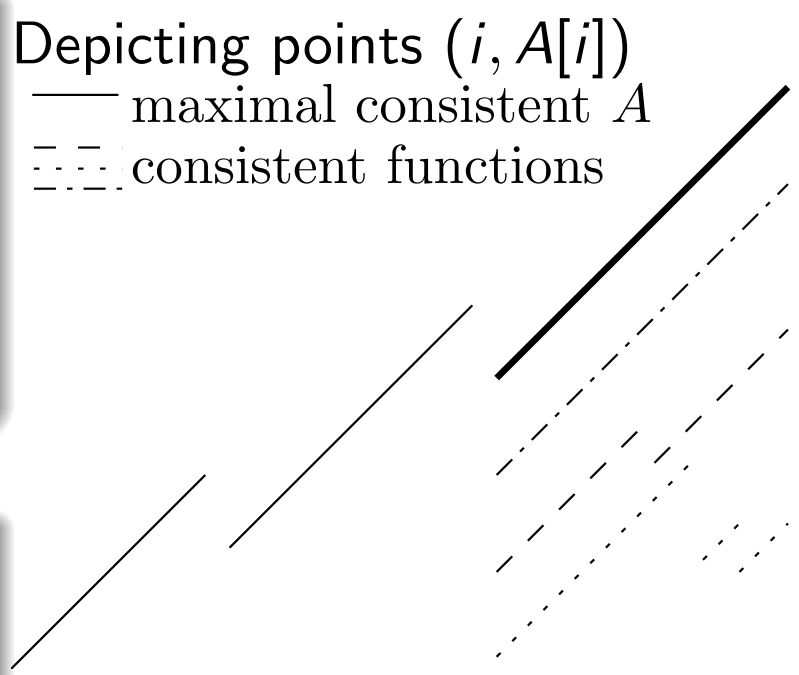
Definition (consistent functions)

$A[1\,..\,n+1]$ is consistent with $A'[1\,..\,n]$ iff there is a word $w[1\,..\,n+1]$ such that

- $A[1\,..\,n+1] = \pi_w[1\,..\,n+1]$,
- $A'[1\,..\,n] = \pi'_w[1\,..\,n]$.

Definition (maximal consistent function)

$A[1\,..\,n+1]$ is the maximal consistent function if for every consistent $B[1\,..\,n+1]$, $B[1\,..\,n+1] \leq A[1\,..\,n+1]$.

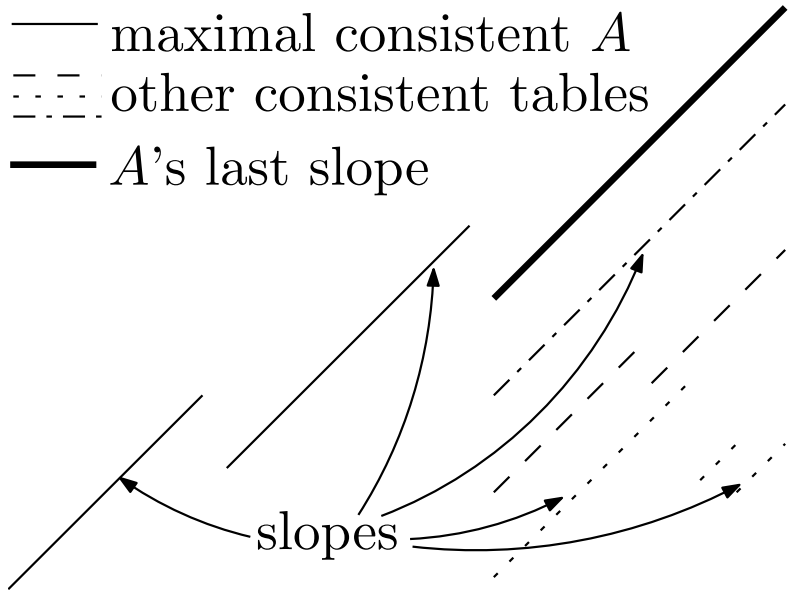


Idea

The algorithm maintains the maximal consistent function.

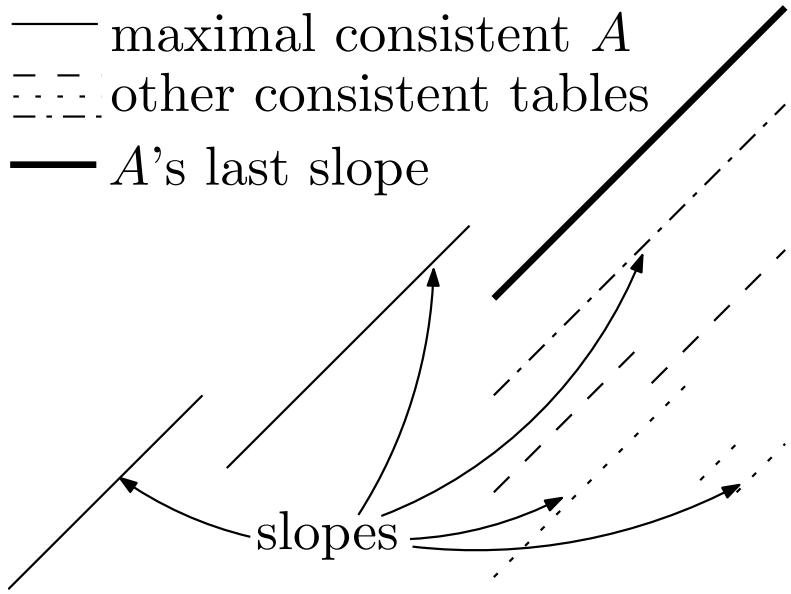
Slope

- $\pi[i+1] = \pi[i] + 1$ or
  $\pi[i+1] < \pi[i] + 1$ (and $\pi[i] = \pi'[i]$)
- slope: maximal set of indices
  $i, i+1, \ldots, i+j$ such that
  $A[i+k] = A[i] + k$ for $k = 1, \ldots, j$
- focus on the last slope of the
  maximal consistent function



Slope

- $\pi[i+1] = \pi[i] + 1$ or
  $\pi[i+1] < \pi[i] + 1$ (and $\pi[i] = \pi'[i]$)
- slope: maximal set of indices
  $i, i+1, \ldots, i+j$ such that
  $A[i+k] = A[i] + k$ for $k = 1, \ldots, j$
- focus on the last slope of the
  maximal consistent function

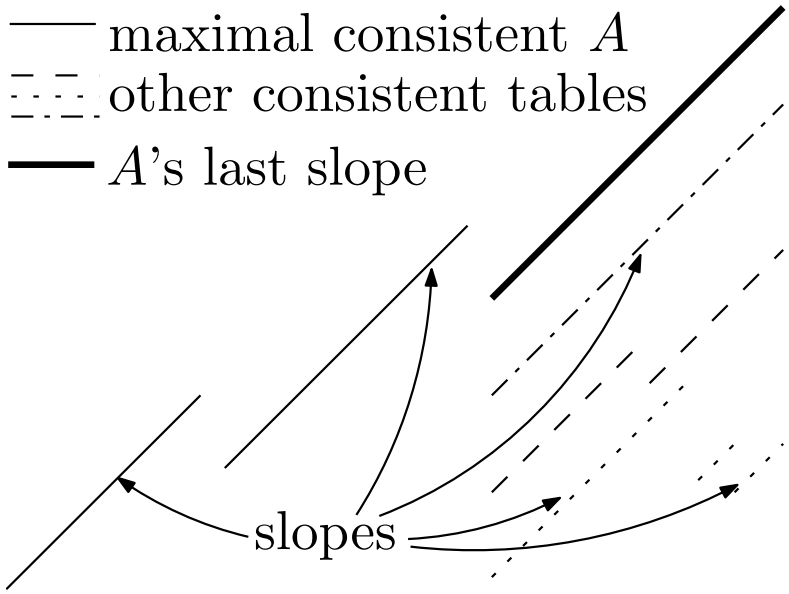


Lemma

Let $[i\,..\,n]$ be the last slope of the maximal consistent function A and B some other consistent function. Then $B[j] = A[j]$ for $j < i$.

Slope

- $\pi[i+1] = \pi[i] + 1$ or
  $\pi[i+1] < \pi[i] + 1$ (and $\pi[i] = \pi'[i]$)
- slope: maximal set of indices
  $i, i+1, \ldots, i+j$ such that
  $A[i+k] = A[i] + k$ for $k = 1, \ldots, j$
- focus on the last slope of the
  maximal consistent function



Lemma

Let $[i \,..\, n]$ be the last slope of the maximal consistent function A and B some other consistent function. Then $B[j] = A[j]$ for $j < i$.

- i as above: A-pin
- if $\pi[n] > \pi'[n]$ then slope extends, if $\pi[n] = \pi'[n]$, slope ends

Update

Read $A'[n]$.

- Recursive relation: $\pi'[j] = \pi[j]$ (when slope ends) or $\pi'[j] = \pi'[\pi[j]]$ (slope continues).
- if $A'[n] = A'[A[n]]$: continue, otherwise, update
- we can lower the last slope or split it

Update

Read $A'[n]$.

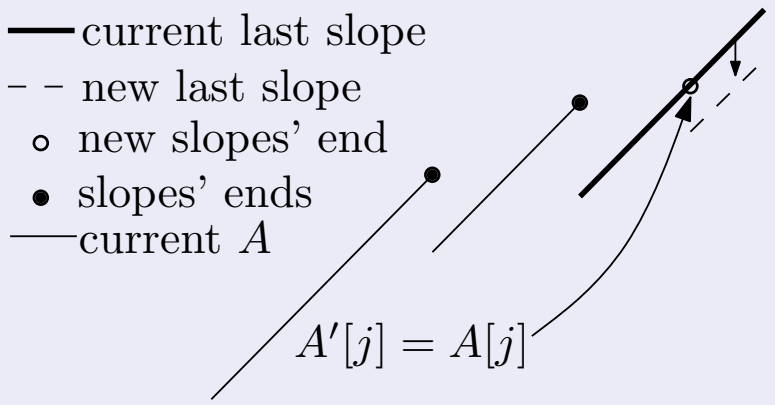
- Recursive relation: $\pi'[j] = \pi[j]$ (when slope ends) or $\pi'[j] = \pi'[\pi[j]]$ (slope continues).
- if $A'[n] = A'[A[n]]$: continue, otherwise, update
- we can lower the last slope or split it
- we check
  - $A[j] < A'[j]$ for $j \in [i\,..\,n]$ (pin is well defined)
  - $A'[j] = A'[A[j]]$ for $j \in [i\,..\,n]$ (values on the last slope are consistent)

Update

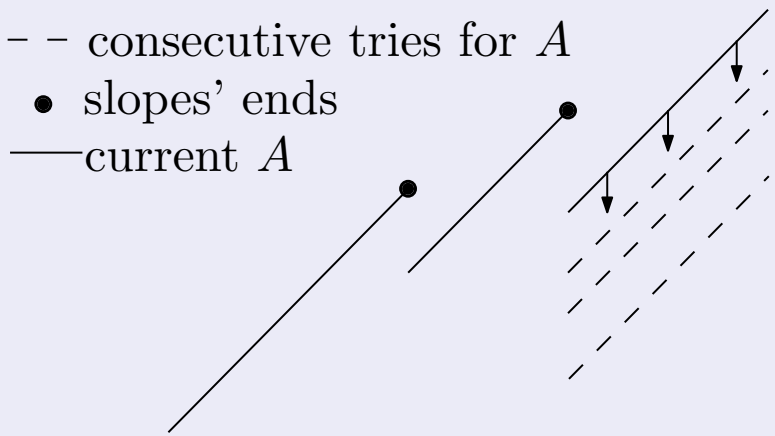
Read $A'[n]$.

- Recursive relation: $\pi'[j] = \pi[j]$ (when slope ends) or $\pi'[j] = \pi'[\pi[j]]$ (slope continues).
- if $A'[n] = A'[A[n]]$: continue, otherwise, update
- we can lower the last slope or split it
- we check
  - $A[j] < A'[j]$ for $j \in [i \,..\, n]$ (pin is well defined)
  - $A'[j] = A'[A[j]]$ for $j \in [i \,..\, n]$ (values on the last slope are consistent)
- if not
  - we change i or reject
  - we change $A[i]$ or reject
- iterate

Splitting the last slope



Adjusting the last slope



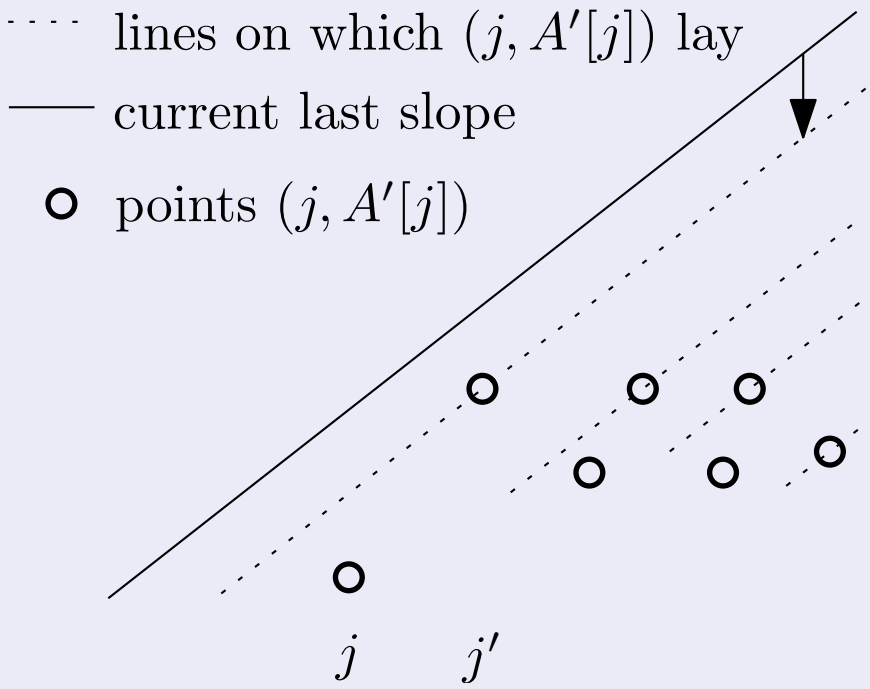
Pin test and slope splitting

We do it the other way around: find first j such that $A[j] \leq A'[j]$

Pin test and slope splitting

We do it the other way around: find first j such that $A[j] \leq A'[j]$

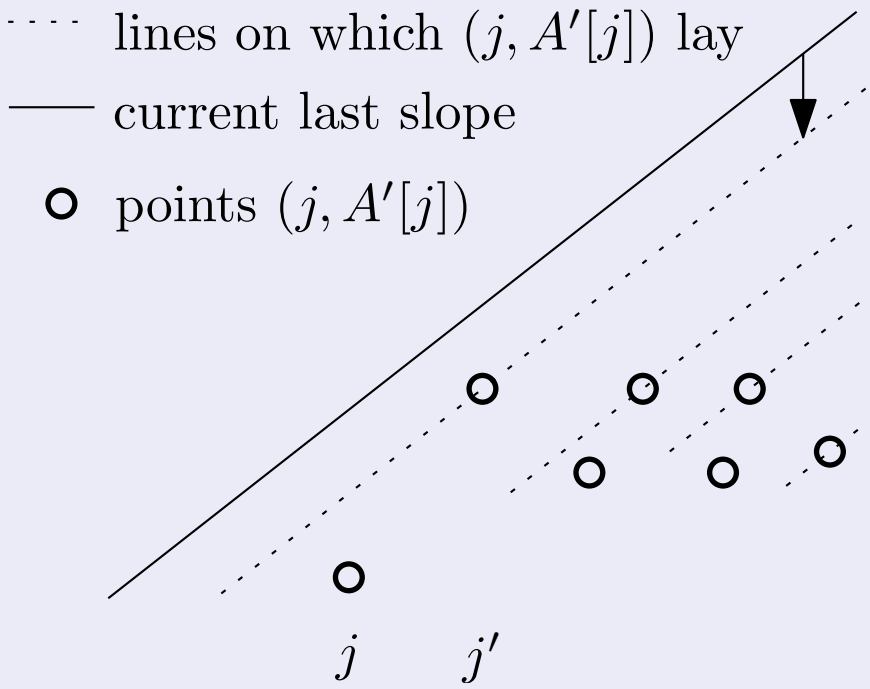
- if $A'[j'] - A'[j] > j' - j > 0$ then j is not important ($j \prec j'$)
- possible j can be kept in a sorted list of indices such that $j \not\prec j'$
- updates only at the end:
  - indices at the end may be removed (due to new element)
- checks only at front



Pin test and slope splitting

We do it the other way around: find first j such that $A[j] \leq A'[j]$

- if $A'[j'] - A'[j] > j' - j > 0$ then j is not important ($j \prec j'$)
- possible j can be kept in a sorted list of indices such that $j \not\prec j'$
- updates only at the end:
  - indices at the end may be removed (due to new element)
- checks only at front



If $A'[j] = A[j]$

- slope is split
- $i \leftarrow j + 1$
- set new $A[i]$

Consistency check

Check fast if

$$A'[i\,..\,n] = A'[A[i]\,..\,A[i] + (n - i)]$$

Consistency check

Check fast if

$$A'[i\,..\,n] = A'[A[i]\,..\,A[i] + (n - i)]$$

- natural approach: suffix trees with lca queries
- online: $\Theta(n \log(|\Sigma|))$

Consistency check

Check fast if

$$A'[i\,..\,n] = A'[A[i]\,..\,A[i] + (n - i)]$$

- natural approach: suffix trees with lca queries
- online: $\Theta(n \log(|\Sigma|))$

Solution:

- use some additional properties
- specialise the data structures

Details (easy)

First attempt

- $\log(|\Sigma|)$: one needs to access $|\Sigma|$ children fast
  - when $|\Sigma| = \mathcal{O}(\log n)$: done by bit tricks in $\mathcal{O}(1)$
- suffix trees for $|\Sigma| = \mathcal{O}(\log n)$ are doable in $\mathcal{O}(n)$ time

Details (easy)

First attempt

- $\log(|\Sigma|)$: one needs to access $|\Sigma|$ children fast
  - when $|\Sigma| = \mathcal{O}(\log n)$: done by bit tricks in $\mathcal{O}(1)$
- suffix trees for $|\Sigma| = \mathcal{O}(\log n)$ are doable in $\mathcal{O}(n)$ time

We want to reduce alphabet, i.e., different A' values

Lemma

A segment of 2^k consecutive entries in the π' array has at most 48 different values from the interval $[2^k, 2^{k+1})$.

Different large values are rare.

Details (complicated)

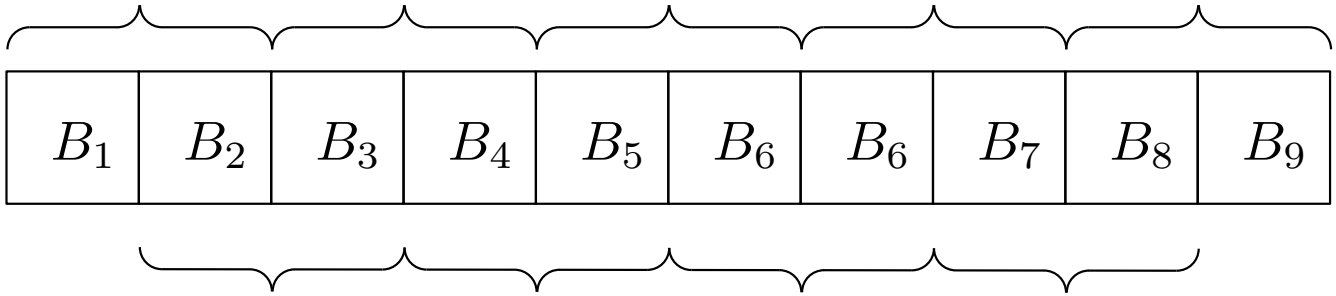
- compress the input function A'
  - small values: rewrite
  - rare large values: encode in binary
  - common large values: link to the previous position
- the size of compressed function is linear

Details (complicated)

- compress the input function A'
  - small values: rewrite
  - rare large values: encode in binary
  - common large values: link to the previous position
- the size of compressed function is linear
- suffix tree for the compressed text — work except for the common large values

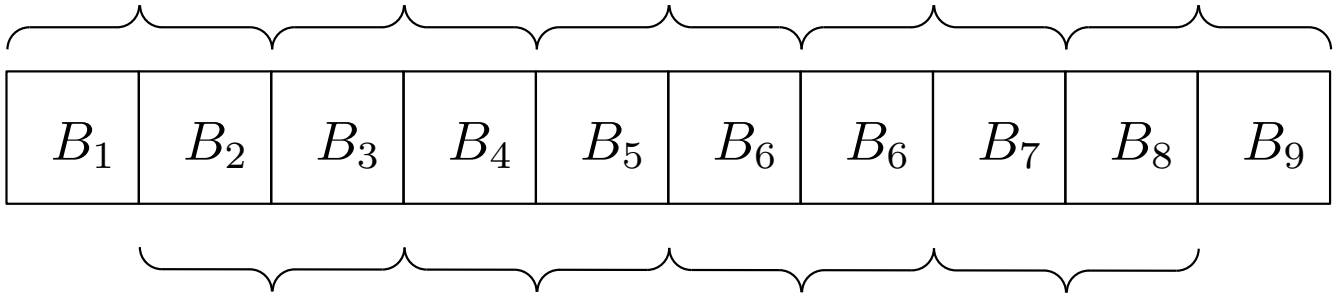
Details (complicated)

- compress the input function A'
  - small values: rewrite
  - rare large values: encode in binary
  - common large values: link to the previous position
- the size of compressed function is linear
- suffix tree for the compressed text — work except for the common large values
- common large values close: small suffix trees for small blocks



Details (complicated)

- compress the input function A'
  - small values: rewrite
  - rare large values: encode in binary
  - common large values: link to the previous position
- the size of compressed function is linear
- suffix tree for the compressed text — work except for the common large values
- common large values close: small suffix trees for small blocks



- common large values far away: reuse previous queries
- amortised analysis

Conclusions

- online validation of the KMP failure function
- linear algorithm
- algorithm: produces a word, calculates the minimum size of the alphabet
- applicable to $g[n] = \pi'[n-1] + 1$, improves Duval et al. result