Algorithmic Research: Cooperation around Oresound

Time-Space Trade-Offs for Longest Common Extensions

(To appear at CPM 2012)

Philip Bille[1], Inge Li Gørtz[1], Benjamin Sach[2], and Hjalte Wedel Vildhøj[1]

[1]Technical University of Denmark, DTU Informatics, {phbi,ilg,hwvi}@imm.dtu.dk
[2]University of Warwick, Department of Computer Science, sach@dcs.warwick.ac.uk

University of Copenhagen, April 17, 2012

The Longest Common Extension Problem

Definition

Problem: Preprocess a string T of length n to support LCE queries:

- LCE(i, j) = The length of the longest common prefix of the suffixes starting at position i and j in T.

Example

$$T = \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \end{array} \qquad \text{LCE}(2, 4) = \ ?$$

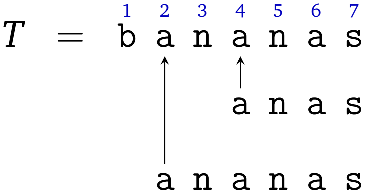
The Longest Common Extension Problem

Definition

Problem: Preprocess a string T of length n to support LCE queries:

- LCE(i, j) = The length of the longest common prefix of the suffixes starting at position i and j in T.

Example



T = b a n a n a s

1 2 3 4 5 6 7

a n a s

a n a n a s

LCE$(2, 4)$ = ?

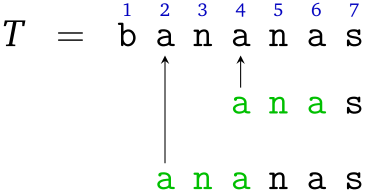
The Longest Common Extension Problem

Definition

Problem: Preprocess a string T of length n to support LCE queries:

- $\text{LCE}(i, j) =$ The length of the longest common prefix of the suffixes starting at position i and j in T.

Example



1 2 3 4 5 6 7

T = b a n a n a s

a n a s

a n a n a s

$\text{LCE}(2, 4) = 3$

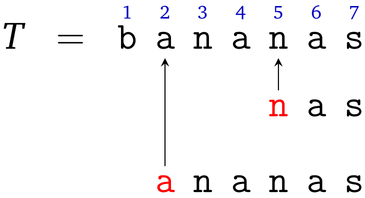
The Longest Common Extension Problem

Definition

Problem: Preprocess a string T of length n to support LCE queries:

- LCE(i, j) = The length of the longest common prefix of the suffixes starting at position i and j in T.

Example



$\text{LCE}(2, 5) = 0$

The Longest Common Extension Problem

Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- Approximate string matching. I.e., find substrings of T such that $T[i \dots j] \approx P$ (hamming or edit distance).
- Finding palindromes. I.e., find substrings of T such that $T[i \dots j] = T[i \dots j]^R$.
- Finding tandem repeats. I.e., find substrings of T such that $T[i \dots j] = UU$ for some string U.

Example: Palindromes

$$\begin{array}{ccccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ T & = & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{c} \end{array}$$

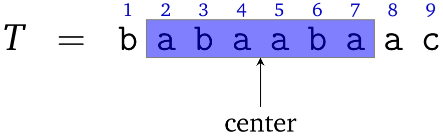
The Longest Common Extension Problem

Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- Approximate string matching. I.e., find substrings of T such that $T[i \ldots j] \approx P$ (hamming or edit distance).
- Finding palindromes. I.e., find substrings of T such that $T[i \ldots j] = T[i \ldots j]^R$.
- Finding tandem repeats. I.e., find substrings of T such that $T[i \ldots j] = UU$ for some string U.

Example: Palindromes



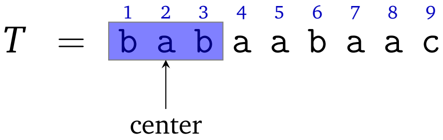
The Longest Common Extension Problem

Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- Approximate string matching. I.e., find substrings of T such that $T[i \ldots j] \approx P$ (hamming or edit distance).
- Finding palindromes. I.e., find substrings of T such that $T[i \ldots j] = T[i \ldots j]^R$.
- Finding tandem repeats. I.e., find substrings of T such that $T[i \ldots j] = UU$ for some string U.

Example: Palindromes



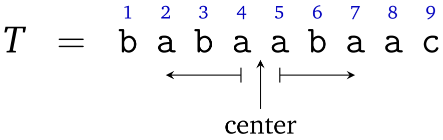
The Longest Common Extension Problem

Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- Approximate string matching. I.e., find substrings of T such that $T[i \dots j] \approx P$ (hamming or edit distance).
- Finding palindromes. I.e., find substrings of T such that $T[i \dots j] = T[i \dots j]^R$.
- Finding tandem repeats. I.e., find substrings of T such that $T[i \dots j] = UU$ for some string U.

Example: Palindromes



All maximal palindromes in P can be reported by performing $2n - 1$ LCE queries (one for each possible center).

Two Simple Solutions

#1: Store nothing

$$T \;=\; \overset{1}{\texttt{b}}\;\underset{\underset{i}{\uparrow}}{\overset{2}{\texttt{a}}}\;\overset{3}{\texttt{n}}\;\underset{\underset{j}{\uparrow}}{\overset{4}{\texttt{a}}}\;\overset{5}{\texttt{n}}\;\overset{6}{\texttt{a}}\;\overset{7}{\texttt{s}} \qquad\qquad \mathrm{LCE}(i,j) \;=$$

Two Simple Solutions

#1: Store nothing

$$T = \overset{1}{\texttt{b}}\ \underset{\underset{i}{\uparrow}}{\overset{2}{\texttt{a}}}\ \overset{3}{\texttt{n}}\ \underset{\underset{j}{\uparrow}}{\overset{4}{\texttt{a}}}\ \overset{5}{\texttt{n}}\ \overset{6}{\texttt{a}}\ \overset{7}{\texttt{s}} \qquad\qquad \mathrm{LCE}(i,j) = 1$$

Two Simple Solutions

#1: Store nothing

$$T = \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \\ & \uparrow & & \uparrow & & & \\ & i & & j & & & \end{array} \qquad \mathrm{LCE}(i,j) = 2$$

Two Simple Solutions

#1: Store nothing

$$T \;=\; \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \\ & \uparrow & & \uparrow & & & \\ & i & & j & & & \end{array} \qquad\qquad \mathrm{LCE}(i,j) \;=\; 3$$

Two Simple Solutions

#1: Store nothing

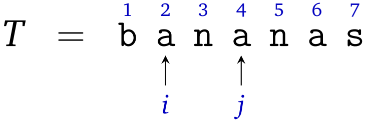
$$T \;=\; \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \\ & \uparrow & & \uparrow & & & \\ & i & & j & & & \end{array}$$

$\mathrm{LCE}(i,j) \;=\; 3$

Time:	$O(n)$
Space:	$O(1)$

Two Simple Solutions

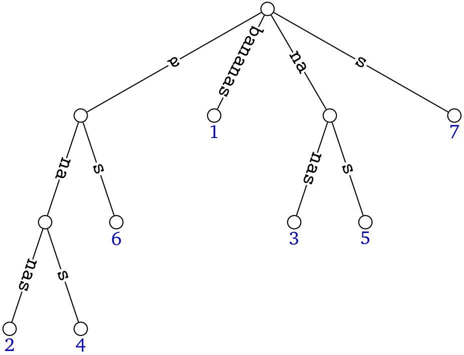
#1: Store nothing



$\mathrm{LCE}(i, j) = 3$

Time: $O(n)$
Space: $O(1)$

#2: Store the suffix tree



Two Simple Solutions

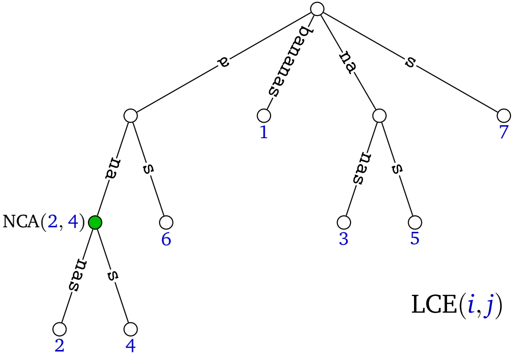
#1: Store nothing

$$T \;=\; \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \\ & \uparrow & & \uparrow & & & \\ & i & & j & & & \end{array}$$

$\text{LCE}(i,j) \;=\; 3$

Time: $O(n)$
Space: $O(1)$

#2: Store the suffix tree



$\text{LCE}(i,j) = |\text{NCA}(i,j)| = 3$

Two Simple Solutions

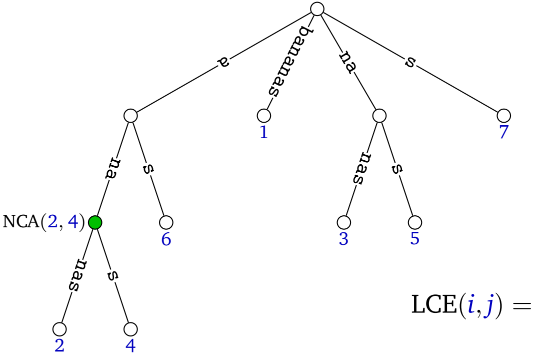
#1: Store nothing

$$T \;=\; \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \texttt{b} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{n} & \texttt{a} & \texttt{s} \\ & \uparrow & & \uparrow & & & \\ & i & & j & & & \end{array}$$

$\mathrm{LCE}(i,j) = 3$

Time: $O(n)$
Space: $O(1)$

#2: Store the suffix tree



Time: $O(1)$
Space: $O(n)$

$\mathrm{LCE}(i,j) = |\mathrm{NCA}(i,j)| = 3$

Two Simple Solutions

#1: Store nothing

$$T \;=\; \overset{1}{\texttt{b}}\;\underset{\overset{\uparrow}{i}}{\overset{2}{\texttt{a}}}\;\overset{3}{\texttt{n}}\;\underset{\overset{\uparrow}{j}}{\overset{4}{\texttt{a}}}\;\overset{5}{\texttt{n}}\;\overset{6}{\texttt{a}}\;\overset{7}{\texttt{s}}$$

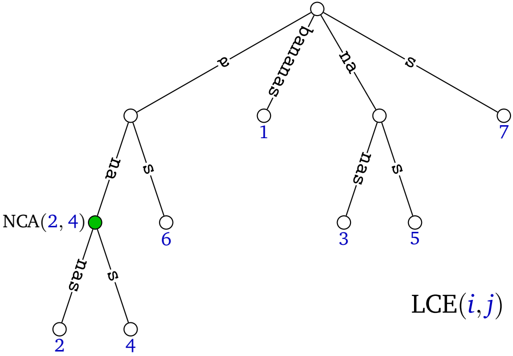
$\text{LCE}(i,j) \;=\; 3$

Time: $O(n)$
Space: $O(1)$

Trade-off?

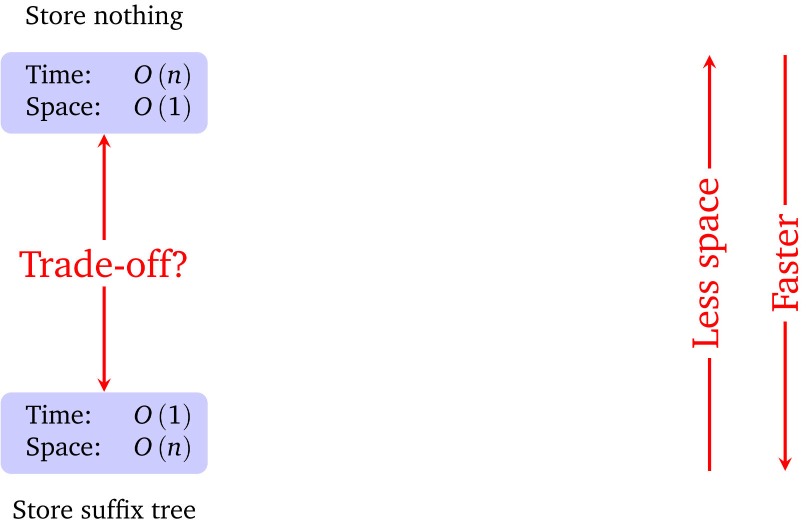
Time: $O(1)$
Space: $O(n)$

#2: Store the suffix tree



$\text{LCE}(i,j) = |\text{NCA}(i,j)| = 3$

Our Results



Store nothing

Time: $O(n)$
Space: $O(1)$

Trade-off?

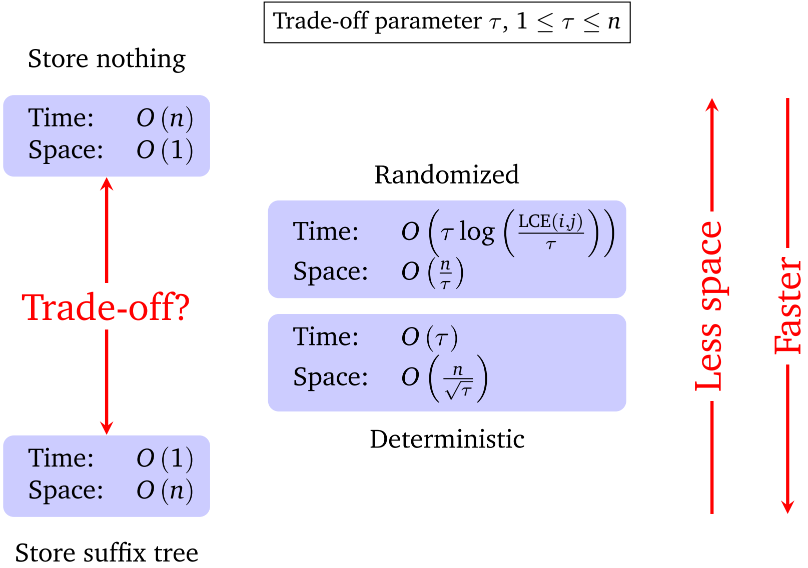
Time: $O(1)$
Space: $O(n)$

Store suffix tree

Less space

Faster

Our Results



Trade-off parameter τ, $1 \leq \tau \leq n$

Store nothing

Time: $O(n)$
Space: $O(1)$

Trade-off?

Time: $O(1)$
Space: $O(n)$

Store suffix tree

Randomized

Time: $O\left(\tau \log \left(\frac{\mathrm{LCE}(i,j)}{\tau}\right)\right)$
Space: $O\left(\frac{n}{\tau}\right)$

Time: $O(\tau)$
Space: $O\left(\frac{n}{\sqrt{\tau}}\right)$

Deterministic

Less space

Faster

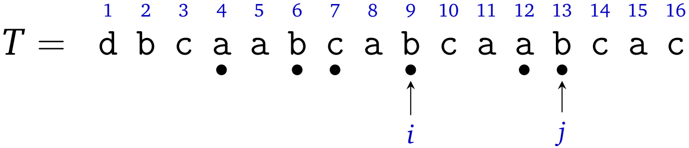
A Deterministic Solution

Idea: Store a subset of the *n* suffixes in a compacted trie.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			

A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			
									↑ i				↑ j			

A Deterministic Solution

Idea: Store a subset of the *n* suffixes in a compacted trie.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			
		↑							↑							
		i							*j*							

A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			
		↑ i							↑ j							

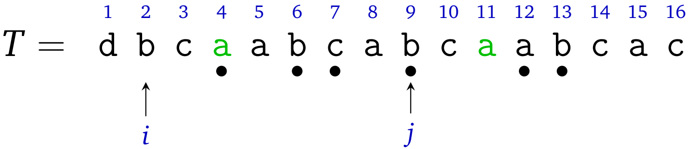
A Deterministic Solution

Idea: Store a subset of the *n* suffixes in a compacted trie.

$$T = \begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \mathtt{d} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} \\ & & & \bullet & & \bullet & \bullet & & \bullet & & & \bullet & \bullet & & & \\ & \uparrow & & & & & & & \uparrow & & & & & & & \\ & i & & & & & & & j & & & & & & & \end{array}$$

A Deterministic Solution

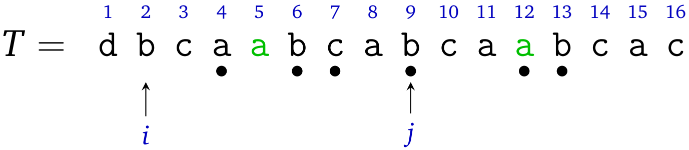
Idea: Store a subset of the *n* suffixes in a compacted trie.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			
		↑ *i*							↑ *j*							

A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

$T =$ d b c a a b c a b c a a b c a c

i

j

A Deterministic Solution

Idea: Store a subset of the *n* suffixes in a compacted trie.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				●		●	●		●			●	●			
		↑							↑							
		i							j							

A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.

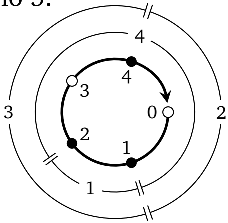
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c

Difference Covers

A *difference cover modulo* τ is a set of integers $D \subseteq \{0, 1, \ldots, \tau - 1\}$ such that for any distance $d \in \{0, 1, \ldots, \tau - 1\}$, D contains two elements separated by distance d modulo τ.

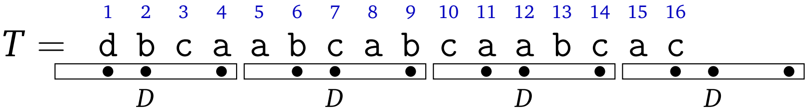
Ex: The set $D = \{1, 2, 4\}$ is a difference cover modulo 5.

d	0	1	2	3	4
i,j	1, 1	2, 1	1, 4	4, 1	1, 2



A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.

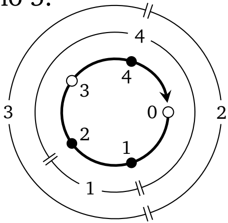


Difference Covers

A *difference cover modulo* τ is a set of integers $D \subseteq \{0, 1, \ldots, \tau - 1\}$ such that for any distance $d \in \{0, 1, \ldots, \tau - 1\}$, D contains two elements separated by distance d modulo τ.

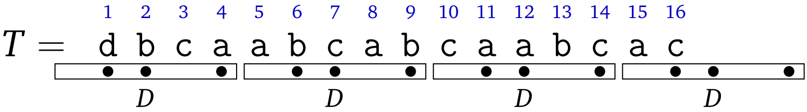
Ex: The set $D = \{1, 2, 4\}$ is a difference cover modulo 5.

d	0	1	2	3	4
i,j	1, 1	2, 1	1, 4	4, 1	1, 2



A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.



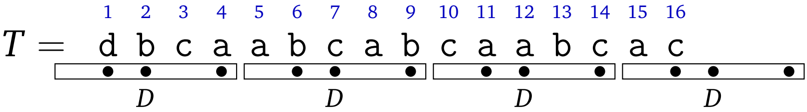
Lemma (Colbourn and Ling[1])

For any τ, a difference cover modulo τ of size at most $\sqrt{1.5\tau} + 6$ can be computed in $O(\sqrt{\tau})$ time.

[1] C. J. Colbourn and A. C. Ling. Quorums from difference covers. Inf. Process. Lett. 75(1-2):9–12, 2000

A Deterministic Solution

Idea: Store a subset of the n suffixes in a compacted trie.



Lemma (Colbourn and Ling[1])

For any τ, a difference cover modulo τ of size at most $\sqrt{1.5\tau} + 6$ can be computed in $O(\sqrt{\tau})$ time.

Analysis

Time: $O(\tau)$

Space: $O(\#\text{stored suffixes}) = O\left(\frac{n}{\tau}|D|\right) = O\left(\frac{n}{\sqrt{\tau}}\right)$

[1] C. J. Colbourn and A. C. Ling. Quorums from difference covers. Inf. Process. Lett. 75(1-2):9–12, 2000

A Randomized Solution

Rabin-Karp Fingerprints

Let p be a sufficiently large prime and choose $b \in \mathbb{Z}_p$ uniformly at random.

$$\phi(S) = \sum_{k=1}^{|S|} S[k]b^k \bmod p\,.$$

$$\begin{array}{rcccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
T = & \texttt{d} & \texttt{b} & \texttt{c} & \texttt{a} & \texttt{a} & \texttt{b} & \texttt{c} & \texttt{a} & \texttt{b} & \texttt{c} & \texttt{a} & \texttt{a} & \texttt{b} & \texttt{c} & \texttt{a} & \texttt{c} \\
= & 3 & \multicolumn{6}{c}{\underbrace{1 \quad 2 \quad 0 \quad 0 \quad 1 \quad 2}_{\phi(T[2\ldots7]) = 120012 \bmod 31 = 11}} & 0 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 2
\end{array}$$

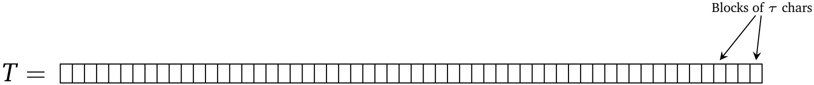
Crucial property: With high probability ϕ is collision-free on substrings of T, i.e., $\phi(S_1) = \phi(S_2)$ iff $S_1 = S_2$.

Also important: $\phi(T[i\ldots j+1])$ can be computed from $\phi(T[i\ldots j])$ in $O(1)$ time.

A Randomized Solution

How to answer a query

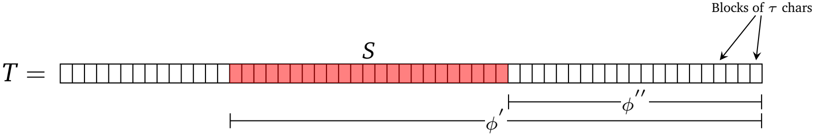
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

Idea: Store fingerprints of suffixes starting at every τ'th position in T.

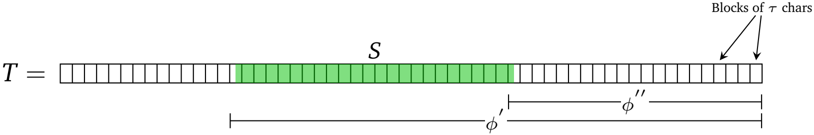


Observation: If S is block aligned we can compute $\phi(S)$ in $O(1)$ time. Otherwise, the time needed is $O(\tau)$.

A Randomized Solution

How to answer a query

Idea: Store fingerprints of suffixes starting at every τ'th position in T.

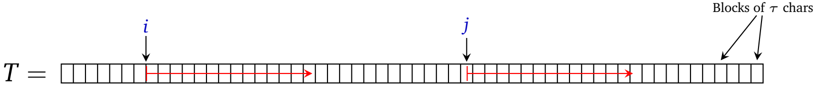


Observation: If S is block aligned we can compute $\phi(S)$ in $O(1)$ time. Otherwise, the time needed is $O(\tau)$.

A Randomized Solution

How to answer a query

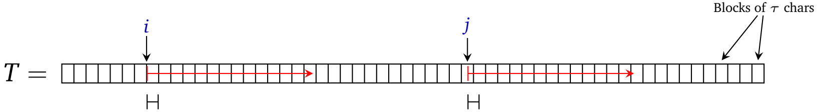
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

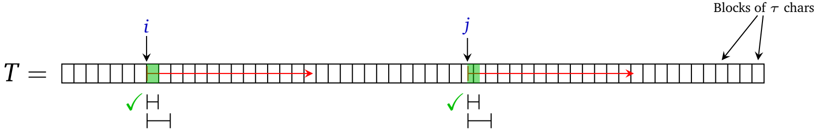
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

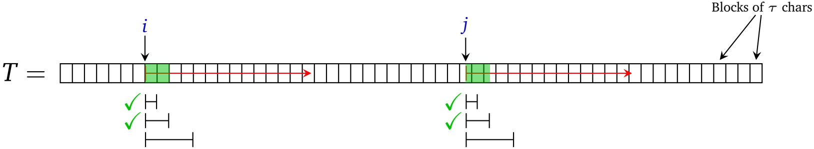
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

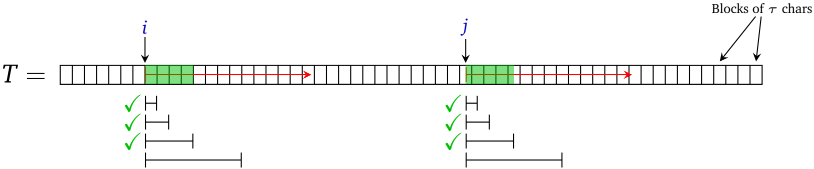
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

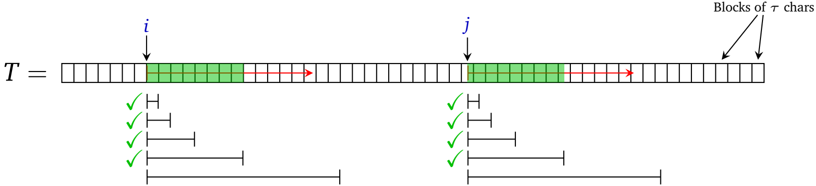
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

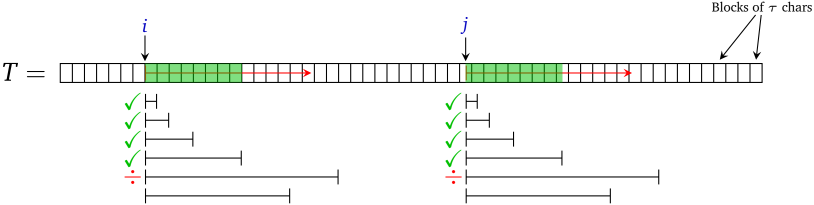
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

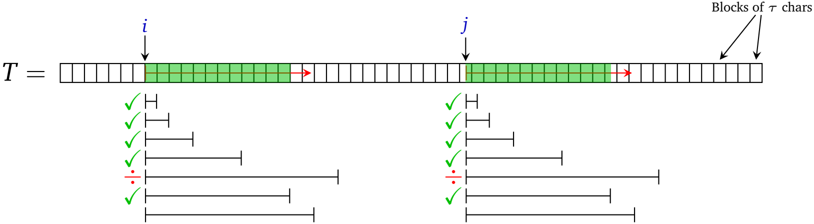
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

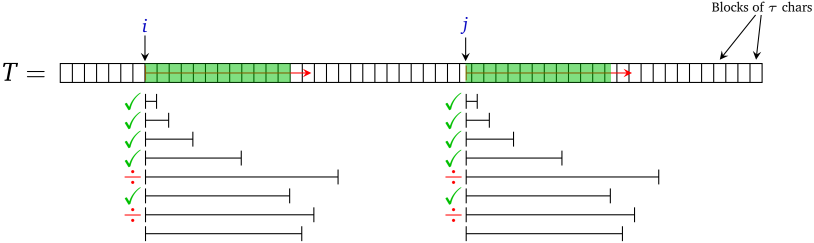
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

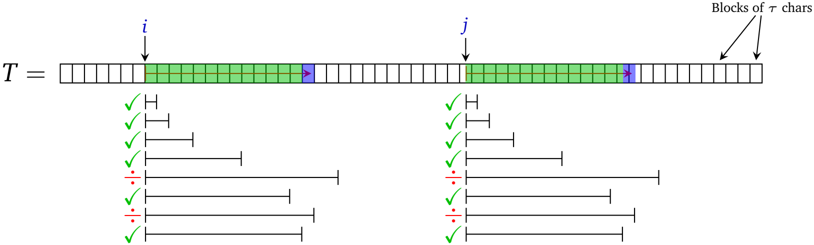
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution

How to answer a query

Idea: Store fingerprints of suffixes starting at every τ'th position in T.



Analysis

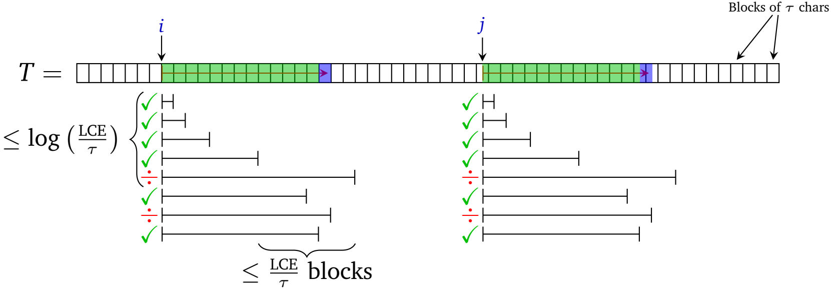
Time: At most $2\log(\frac{\text{LCE}}{\tau})$ fingerprint comparisons each taking time $O(\tau)$. Hence query time $O\left(\tau\log\left(\frac{\text{LCE}}{\tau}\right)\right)$.

Space: $O\left(\frac{n}{\tau}\right)$.

A Randomized Solution

How to answer a query

Idea: Store fingerprints of suffixes starting at every τ'th position in T.



Blocks of τ chars
i
j
$T =$
$\leq \log\left(\frac{\text{LCE}}{\tau}\right)$
$\leq \frac{\text{LCE}}{\tau}$ blocks

Analysis

Time: At most $2\log(\frac{\text{LCE}}{\tau})$ fingerprint comparisons each taking time $O(\tau)$. Hence query time $O\left(\tau \log\left(\frac{\text{LCE}}{\tau}\right)\right)$.

Space: $O\left(\frac{n}{\tau}\right)$.