Time-Space Trade-Offs for Longest Common Extensions

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

CPM 2012, Helsinki
July 4, 2012

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

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

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

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

The Longest Common Extension Problem

Definition

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

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

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

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

$$T = \overset{1}{\texttt{b}}\ \overset{2}{\texttt{a}}\ \overset{3}{\texttt{n}}\ \overset{4}{\texttt{a}}\ \overset{5}{\texttt{n}}\ \overset{6}{\texttt{a}}\ \overset{7}{\texttt{s}}$$

$$\texttt{n a s}$$

$$\texttt{a n a n a s}$$

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

The Longest Common Extension Problem

Definition

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

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

Example

$$T = \overset{1}{\mathtt{b}}\ \overset{2}{\mathtt{a}}\ \overset{3}{\mathtt{n}}\ \overset{4}{\mathtt{a}}\ \overset{5}{\mathtt{n}}\ \overset{6}{\mathtt{a}}\ \overset{7}{\mathtt{s}}$$

$\mathtt{n\ a\ s}$

$\mathtt{a\ n\ a\ n\ a\ s}$

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

- We assume that the input is given in read-only memory and is not included in the space complexity.

Two Simple Solutions

#1: Store nothing

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

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

Two Simple Solutions

#1: Store nothing

$$T = \overset{1}{\texttt{b}}\ \overset{2}{\texttt{a}}\ \overset{3}{\texttt{n}}\ \overset{4}{\texttt{a}}\ \overset{5}{\texttt{n}}\ \overset{6}{\texttt{a}}\ \overset{7}{\texttt{s}}$$

$$\underset{i}{\uparrow} \qquad \underset{j}{\uparrow}$$

$\text{LCE}(i, j) = 1$

Two Simple Solutions

#1: Store nothing

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

$\mathrm{LCE}(i,j) = 2$

Two Simple Solutions

#1: Store nothing

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

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

Two Simple Solutions

#1: Store nothing

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

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

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

Two Simple Solutions

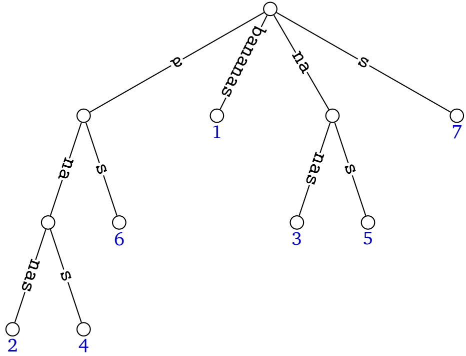
#1: Store nothing

	1	2	3	4	5	6	7
T =	b	a	n	a	n	a	s
		↑ i		↑ j			

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

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

#2: Store the suffix tree



Two Simple Solutions

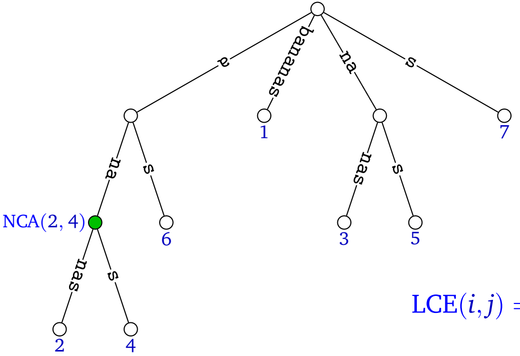
#1: Store nothing

T	$=$	1	2	3	4	5	6	7
		b	a	n	a	n	a	s
			$\uparrow$ i		$\uparrow$ j			

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

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

#2: Store the suffix tree



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

Two Simple Solutions

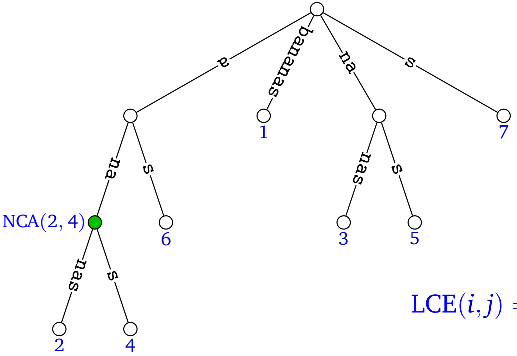
#1: Store nothing

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

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

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

#2: Store the suffix tree



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

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

Two Simple Solutions

#1: Store nothing

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

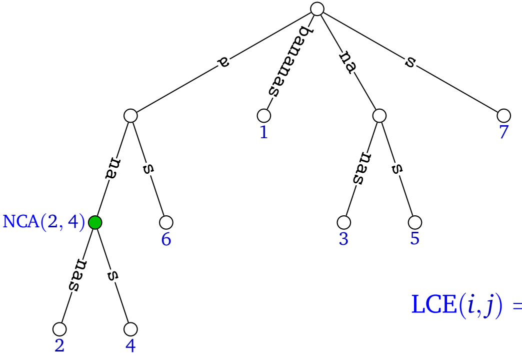
$\mathrm{LCE}(i,j) = 3$

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

Trade-off?

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

#2: Store the suffix tree



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

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, $1 \le \tau \le n$

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
				•		•	•		•			•	•			
		$\uparrow$ i							$\uparrow$ 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
				•		•	•		•			•	•			
		$\uparrow$							$\uparrow$							
		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
				•		•	•		•			•	•			
		$\uparrow$ i							$\uparrow$ 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
		$\uparrow$		$\bullet$		$\bullet$	$\bullet$		$\bullet$ $\uparrow$			$\bullet$	$\bullet$			
		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.

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

A Deterministic Solution

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

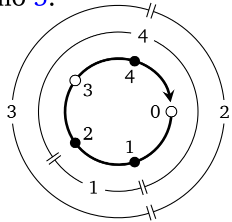
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T =$	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c

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.

	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
	•	•		•		•	•		•		•	•		•	•	•

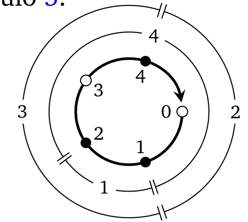
D D D D

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.

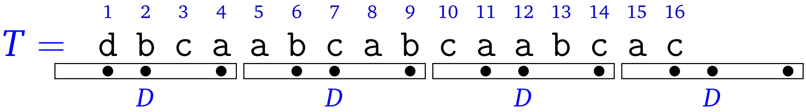
Ex: The set $D = \{1, 2, 4\}$ is a difference cover modulo 5.

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.

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 (Monte Carlo)

Rabin-Karp Fingerprints

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

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

	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 Randomized Solution (Monte Carlo)

Rabin-Karp Fingerprints

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

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

$$\begin{array}{cccccccccccccccccc}
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
T & = & \mathtt{d} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} \\
 & = & 3 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 2
\end{array}$$

$$\phi(T[2 \ldots 7]) = 1b^1 + 2b^2 + 0b^3 + 0b^4 + 1b^5 + 2b^6 \bmod p$$

A Randomized Solution (Monte Carlo)

Rabin-Karp Fingerprints

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

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

$$\begin{array}{cccccccccccccccccc}
 & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
T & = & \mathtt{d} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} \\
 & = & 3 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 2
\end{array}$$

$$\phi(T[2\dots7]) = 1b^1 + 2b^2 + 0b^3 + 0b^4 + 1b^5 + 2b^6 \bmod p$$

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

A Randomized Solution (Monte Carlo)

Rabin-Karp Fingerprints

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

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

$$\begin{array}{rcccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
T = & \mathtt{d} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} \\
= & 3 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 2
\end{array}$$

$$\phi(T[2\ldots7]) = 1b^1 + 2b^2 + 0b^3 + 0b^4 + 1b^5 + 2b^6 \bmod p$$

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

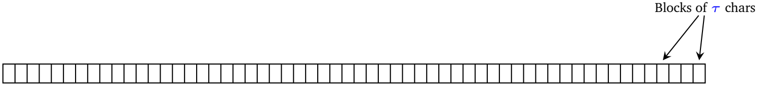
Also important: $\phi(T[i\ldots j+1])$ can be computed from $\phi(T[i\ldots j])$ in $O(1)$ time.

A Randomized Solution (Monte Carlo)

How to answer a query

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

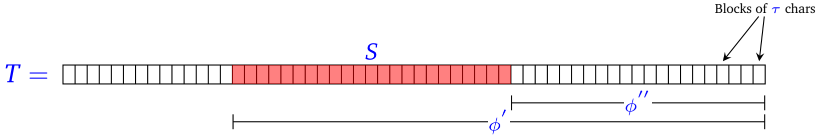
$T =$



A Randomized Solution (Monte Carlo)

How to answer a query

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

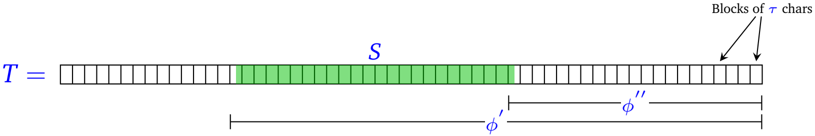


Observation: If S is block aligned we can compute $\phi(S)$ in $O(1)$ time. Otherwise, the time needed is $O(\tau)$.

A Randomized Solution (Monte Carlo)

How to answer a query

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

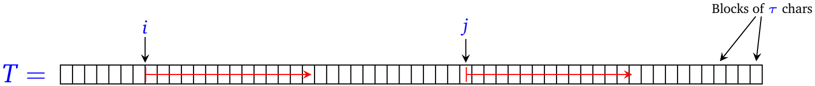


Observation: If S is block aligned we can compute $\phi(S)$ in $O(1)$ time. Otherwise, the time needed is $O(\tau)$.

A Randomized Solution (Monte Carlo)

How to answer a query

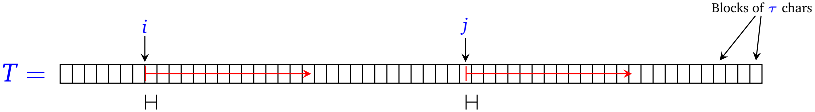
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

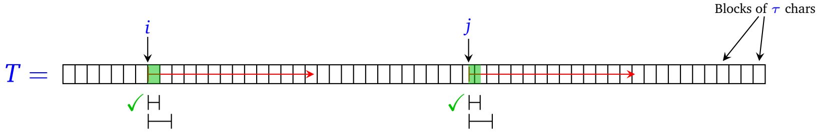
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

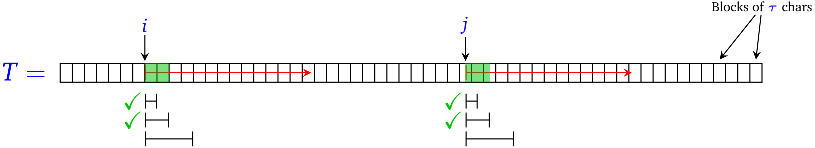
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

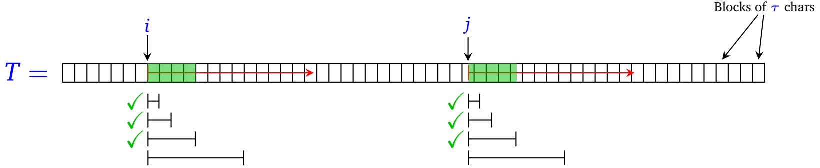
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

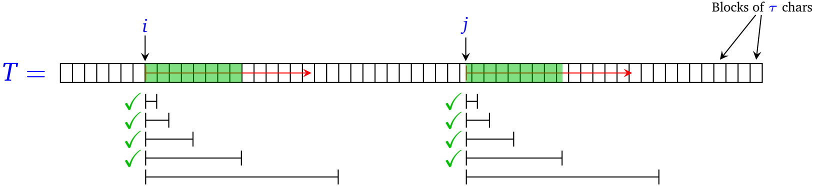
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

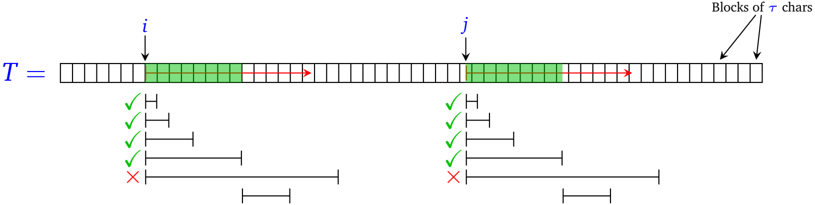
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

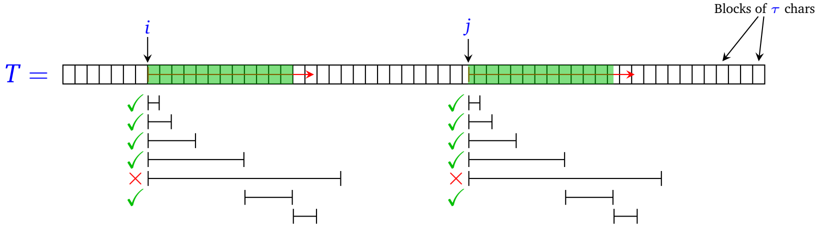
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

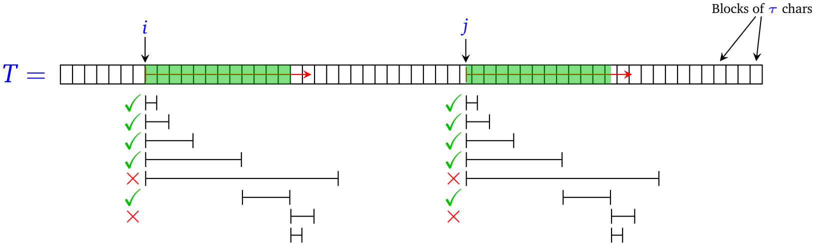
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

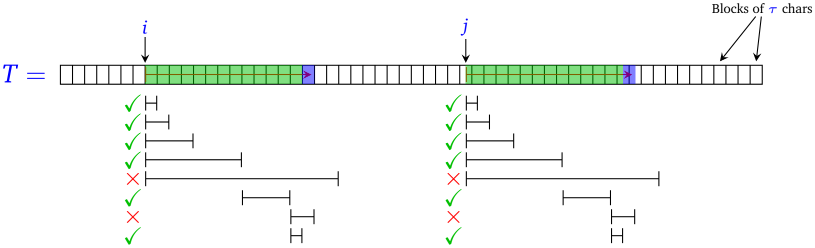
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

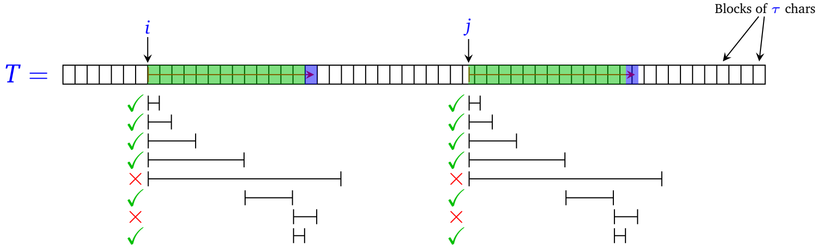
Idea: Store fingerprints of suffixes starting at every τ'th position in T.



A Randomized Solution (Monte Carlo)

How to answer a query

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



Analysis

Time: Only $O(\log(\frac{\text{LCE}}{\tau}))$ 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 (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1]$ for some ℓ, j

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1]$ for some ℓ, j

... *this cuts down the number of fingerprints we need to check!*

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

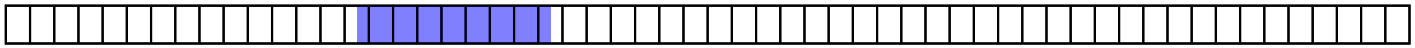
Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^\ell - 1]$ for some ℓ, j

. . . this cuts down the number of fingerprints we need to check!

General idea: For each $\ell \geq 0$ in increasing order, check that for all i, j,

$$\begin{array}{lccc} & \phi(T[i \ldots i + \tau{\cdot}2^\ell - 1]) & = & \phi(T[j\tau \ldots j\tau + \tau{\cdot}2^\ell - 1]) \\ \text{iff} & T[i \ldots i + \tau{\cdot}2^\ell - 1] & = & T[j\tau \ldots j\tau + \tau{\cdot}2^\ell - 1] \end{array}$$

$\phi(T[i \ldots i + \tau \cdot 2^\ell - 1])$

$T =$ 

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1]$ for some ℓ, j

... this cuts down the number of fingerprints we need to check!

General idea: For each $\ell \geq 0$ in increasing order, check that for all i, j,

$$\begin{aligned} \phi(T[i \ldots i + \tau \cdot 2^{\ell} - 1]) &= \phi(T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1]) \\ \text{iff} \quad T[i \ldots i + \tau \cdot 2^{\ell} - 1] &= T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1] \end{aligned}$$

$\phi(T[i \ldots i + \tau \cdot 2^{\ell} - 1]) \quad = \phi(T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1])$

$T =$ 

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^\ell - 1]$ for some ℓ, j

…this cuts down the number of fingerprints we need to check!

General idea: For each $\ell \geq 0$ in increasing order, check that for all i, j,

$$\begin{array}{rcl} \phi(T[i \ldots i + \tau{\cdot}2^\ell - 1]) & = & \phi(T[j\tau \ldots j\tau + \tau{\cdot}2^\ell - 1]) \\ \text{iff} \quad T[i \ldots i + \tau{\cdot}2^\ell - 1] & = & T[j\tau \ldots j\tau + \tau{\cdot}2^\ell - 1] \end{array}$$

$\phi(T[i \ldots i + \tau \cdot 2^\ell - 1]) \quad = \phi(T[j\tau \ldots j\tau + \tau \cdot 2^\ell - 1])$

$T =$ 

$\phi(T[i \ldots i + \tau \cdot 2^{\ell-1} - 1]) \stackrel{?}{=} \phi(T[j\tau \ldots j\tau + \tau \cdot 2^{\ell-1} - 1])$

A Randomized Solution (Las Vegas)

Question: Can we verify that ϕ is collision free during preprocessing?

Challenge: Doing this quickly while using $O(\frac{n}{\tau})$ space.

Observation: Whenever we compare two fingerprints, we can ensure that one of them is of the form $T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1]$ for some ℓ, j

. . . *this cuts down the number of fingerprints we need to check!*

General idea: For each $\ell \geq 0$ in increasing order, check that for all i, j,

$$\begin{aligned} &\phi(T[i \ldots i + \tau{\cdot}2^{\ell} - 1]) = \phi(T[j\tau \ldots j\tau + \tau{\cdot}2^{\ell} - 1]) \\ \text{iff} \quad &T[i \ldots i + \tau{\cdot}2^{\ell} - 1] = T[j\tau \ldots j\tau + \tau{\cdot}2^{\ell} - 1] \end{aligned}$$

$\phi(T[i \ldots i + \tau \cdot 2^{\ell} - 1]) \quad = \phi(T[j\tau \ldots j\tau + \tau \cdot 2^{\ell} - 1])$

$T =$ 

$\phi(T[i + \tau \cdot 2^{\ell-1} \ldots i + \tau \cdot 2^{\ell} - 1])$

$\stackrel{?}{=} \phi(T[j\tau + \tau \cdot 2^{\ell-1} \ldots j\tau + \tau \cdot 2^{\ell} - 1])$

Conclusions

We gave three time-space trade-offs for LCE on a single string:

- A deterministic solution
  - $O(\tau)$ query time
  - $O(n/\sqrt{\tau})$ space (even during preprocessing)
  - $O(n^2/\sqrt{\tau})$ preprocessing time
- A Monte-Carlo solution
  - $O\left(\tau \log\left(\mathrm{LCE}(i,j)/\tau\right)\right)$ query time (correct with high prob.)
  - $O(n/\tau)$ space (even during preprocessing)
  - $O(n)$ preprocessing time.
- A Las-Vegas solution
  - $O\left(\tau \log\left(\mathrm{LCE}(i,j)/\tau\right)\right)$ query time (correct with certainty)
  - $O(n/\tau)$ space (even during preprocessing)
  - $O(n \log n)$ preprocessing time with high prob.

Conclusions

We gave three time-space trade-offs for LCE on two strings:

- A deterministic solution
  - $O(\tau)$ query time
  - $O(n/\tau + m/\sqrt{\tau})$ space (even during preprocessing)
  - $O(nm/\sqrt{\tau})$ preprocessing time
- A Monte-Carlo solution
  - $O\left(\tau \log\left(\mathrm{LCE}(i,j)/\tau\right)\right)$ query time (correct with high prob.)
  - $O((n+m)/\tau)$ space (even during preprocessing)
  - $O(n)$ preprocessing time.
- A Las-Vegas solution
  - $O\left(\tau \log\left(\mathrm{LCE}(i,j)/\tau\right)\right)$ query time (correct with certainty)
  - $O((n+m)/\tau)$ space (even during preprocessing)
  - $O(n \log n)$ preprocessing time with high prob.