Text Indexing

Lecture 12: Longest Common Extensions

Florian Kurpicz

Recap: Document Listing and Top-k Retrieval

Definition: Document Listing

Given a collection of D documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ and a pattern $P \in \Sigma^*$, return all $j \in [1, D]$, such that d_j contains P.

Recap: Document Listing and Top-k Retrieval

Definition: Document Listing

Given a collection of D documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ and a pattern $P \in \Sigma^*$, return all $j \in [1, D]$, such that d_j contains P.

- $d_1 =$ ATA
- $d_2 =$ TAAA
- $d_3 =$ TATA

And for queries:

- $P =$ TA is contained in d_1, d_2, and d_3
- $P =$ ATA is contained in d_1 and d_3

Recap: Document Listing and Top-k Retrieval

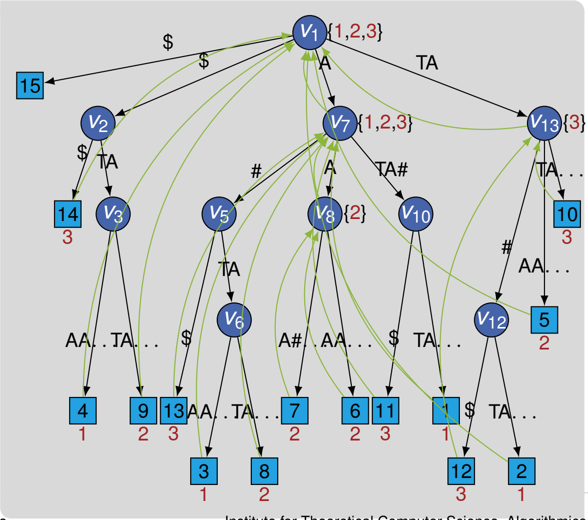
Definition: Document Listing

Given a collection of D documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ and a pattern $P \in \Sigma^*$, return all $j \in [1, D]$, such that d_j contains P.

- $d_1 = \text{ATA}$
- $d_2 = \text{TAAA}$
- $d_3 = \text{TATA}$

And for queries:

- $P = \text{TA}$ is contained in d_1, d_2, and d_3
- $P = \text{ATA}$ is contained in d_1 and d_3



Recap: Pattern Matching with the LCP-Array (1/3)

- remember how many characters of the pattern and suffix match
- identify how long the prefix of the old and next suffix is
- do so using the LCP-array and
- range minimum queries detailed introduction in **Advanced Data Structures**

- $lcp(i,j) = \max\{k\colon\ T[i..i+k)$
- $lcp(i,j) = T[j..j+k)\} = LCP[RMQ_{LCP}(i+1,j)]$
- RMQs can be answered in $O(1)$ time and
- require $O(n)$ space

Definition: Range Minimum Queries

Given an array $A[1..m)$, a **range minimum query** for a range $\ell \leq r \in [1,n)$ returns

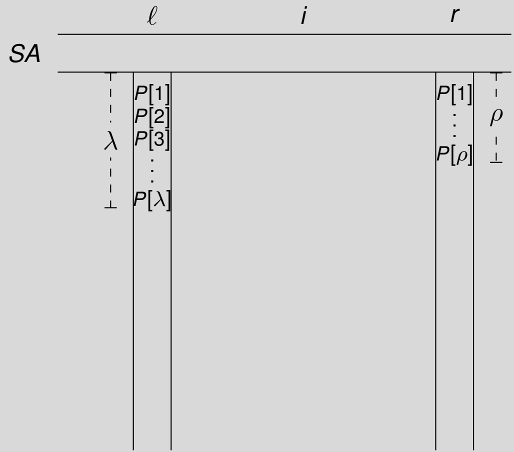
$$RMQ_A(\ell,r) = \arg\min\{A[k]\colon\ k \in [\ell,r]\}$$

Recap: Pattern Matching with the LCP-Array (2/3)

- during binary search matched
- λ characters with left border ℓ and
- ρ characters with right border r
- w.l.o.g. let $\lambda \geq \rho$

- middle position i
- decide if continue in $[\ell, i]$ or $[i, r]$

- let $\xi = lcp(SA[\ell], SA[i])$ ℹ $O(1)$ time with RMQs



Recap: Pattern Matching with the LCP-Array (3/3)

- let $\xi = lcp(SA[\ell], SA[i])$

$\xi > \lambda$

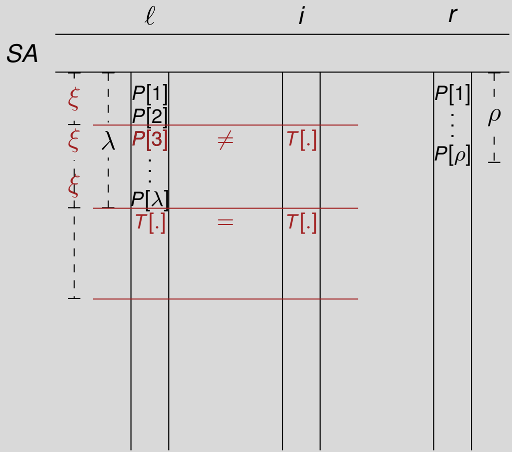
- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before

$\xi < \lambda$

- $\xi \geq \rho$ and $P[\xi + 1] < T[SA[i] + \xi]$
- $r = i$ and $\rho = \xi$ without character comparison



Old Problem, New Name

Karlsruhe Institute of Technology

Definition: Longest Common Extensions

Given a text T of size n over an alphabet of size σ, construct data structure that answers for $i, j \in [1, n]$

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0\colon T[i, i+\ell) = T[j, j+\ell)\}$$

- also denoted as $lcp(i, j)$ ℹ in this lecture

Old Problem, New Name

Definition: Longest Common Extensions

Given a text T of size n over an alphabet of size σ, construct data structure that answers for $i, j \in [1, n]$

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

- also denoted as $lcp(i, j)$ ℹ in this lecture

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$\mathrm{lce}_T(1, 14) = 0\ 1\ 2\ 3\ 4\ 5$

Old Problem, New Name

Definition: Longest Common Extensions

Given a text T of size n over an alphabet of size σ, construct data structure that answers for $i, j \in [1, n]$

$$\mathrm{lce}_T(i,j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

- also denoted as $lcp(i,j)$ in this lecture

Applications

- (sparse) suffix sorting
- approximate pattern matching
- ...

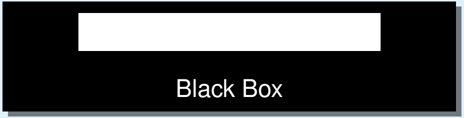
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$$\mathrm{lce}_T(1,14) = 0\ 1\ 2\ 3\ 4\ 5$$

Practical Algorithms for Longest Common Extensions [IT09]

Sophisticated Black Box (BB)

- based on ISA, LCP, and RMQ

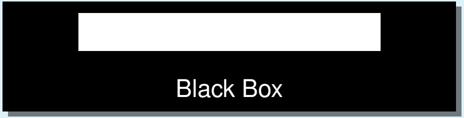


- $O(1)$ query time, $\approx 9n$ bytes additional space

Practical Algorithms for Longest Common Extensions [IT09]

Sophisticated Black Box (BB)

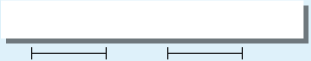
- based on ISA, LCP, and RMQ



- $O(1)$ query time, $\approx 9n$ bytes additional space

Ultra Naive Scan (UNS)

- compare character by character



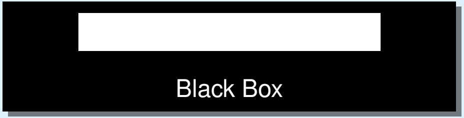
- $O(n)$ query time, no additional space

Practical Algorithms for Longest Common Extensions [IT09]

Karlsruhe Institute of Technology

Sophisticated Black Box (BB)

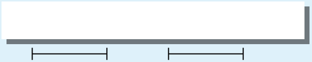
- based on ISA, LCP, and RMQ



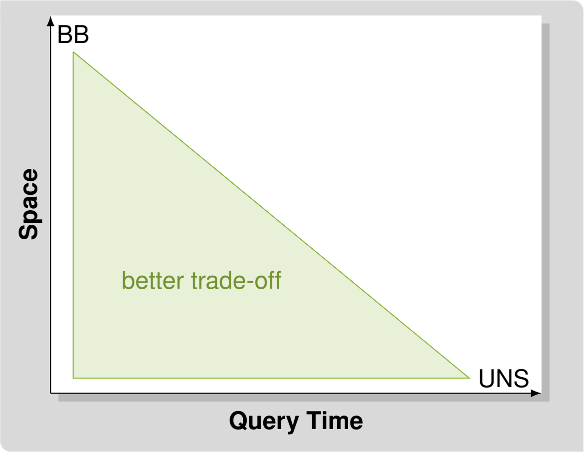
- $O(1)$ query time, $\approx 9n$ bytes additional space

Ultra Naive Scan (UNS)

- compare character by character



- $O(n)$ query time, no additional space



Monte Carlo and Las Vegas Algorithms

- setting: randomized algorithms

Monte Carlo Algorithm

- returns wrong result with small probability
- deterministic running time

Monte Carlo and Las Vegas Algorithms

- setting: randomized algorithms

Monte Carlo Algorithm

- returns wrong result with small probability
- deterministic running time

Las Vegas Algorithm

- returns correct result
- only expected running time

Monte Carlo and Las Vegas Algorithms

- setting: randomized algorithms

Monte Carlo Algorithm

- returns wrong result with small probability
- deterministic running time

Las Vegas Algorithm

- returns correct result
- only expected running time

- some Monte Carlo algorithms can be turned into Las Vegas algorithms
- depends on correctness check
- all Monte Carlo algorithms presented today can be turned into Las Vegas algorithms

Randomized String Matching

- compute s of strings
- application not limited to LCEs

Randomized String Matching

- compute s of strings
- application not limited to LCEs

Definition: Karp-Rabin Fingerprint [KR87]

Given a text T of length n over an alphabet of size σ and a random prime number $q \in \Theta(n^c)$, the Karp-Rabin fingerprint of $T[i..j]$ is

$$(i, j) = \left(\sum_{k=i}^{j} T[k] \cdot \sigma^{j-k}\right) \bmod q$$

ℹ $(x + y) \bmod z = z \bmod z + y \bmod z \pmod{z}$

Randomized String Matching

- compute 🫆s of strings
- application not limited to LCEs

Definition: Karp-Rabin Fingerprint [KR87]

Given a text T of length n over an alphabet of size σ and a random prime number $q \in \Theta(n^c)$, the Karp-Rabin fingerprint of $T[i..j]$ is

$$🫆(i,j) = \left(\sum_{k=i}^{j} T[k] \cdot \sigma^{j-k}\right) \bmod q$$

ℹ $(x + y) \bmod z = z \bmod z + y \bmod z \pmod{z}$

- if $T[i..i+\ell] = T[j..j+\ell]$, then

$$🫆(i, i+\ell) = 🫆(j, j+\ell)$$

- if $T[i..i+\ell] \neq T[j..j+\ell]$, then

$$\mathrm{Prob}(🫆(i, i+\ell) = 🫆(j, j+\ell)) \in O\left(\frac{\ell \lg \sigma}{n^c}\right)$$

- prime has to be chosen uniformly at random
- how to turn it into Las Vegas algorithm?

Randomized String Matching

- compute 🫆s of strings
- application not limited to LCEs

Definition: Karp-Rabin Fingerprint [KR87]

Given a text T of length n over an alphabet of size σ and a random prime number $q \in \Theta(n^c)$, the Karp-Rabin fingerprint of $T[i..j]$ is

$$\text{🫆}(i,j) = \left(\sum_{k=i}^{j} T[k] \cdot \sigma^{j-k}\right) \bmod q$$

$(x+y) \bmod z = z \bmod z + y \bmod z \pmod{z}$

- if $T[i..i+\ell] = T[j..j+\ell]$, then

$$\text{🫆}(i, i+\ell) = \text{🫆}(j, j+\ell)$$

- if $T[i..i+\ell] \neq T[j..j+\ell]$, then

$$\text{Prob}(\text{🫆}(i, i+\ell) = \text{🫆}(j, j+\ell)) \in O\left(\frac{\ell \lg \sigma}{n^c}\right)$$

- prime has to be chosen uniformly at random
- how to turn it into Las Vegas algorithm?

- example on the board

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word ℹ e.g., 64 bit

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\lg\sigma)$ 8 for byte alphabet

Overwriting the Text with Fingerprints (1/2) [Pre18]

Karlsruhe Institute of Technology

- given a text T over an alphabet of size σ
- let w be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\lg\sigma)$ 8 for byte alphabet
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\lg \sigma)$ 8 for byte alphabet
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- group the text into size-τ blocks: $\mathrm{B}[1..n/\tau]$ with

$$B[i] = T[(i-1)\tau + 1..i\tau]$$

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\lg\sigma)$ 8 for byte alphabet
- choose random prime $q \in [\frac{1}{2}\sigma^{\tau}, \sigma^{\tau})$
- group the text into size-τ blocks: $B[1..n/\tau]$ with

$$B[i] = T[(i-1)\tau + 1..i\tau]$$

- compute $P'[i] = (i, \tau i)$ for $i \in [1, n/\tau]$

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word ⓘ e.g., 64 bit
- choose $\tau \in \Theta(w/\lg\sigma)$ ⓘ 8 for byte alphabet
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- group the text into size-τ blocks: B$[1..n/\tau]$ with

$$B[i] = T[(i-1)\tau + 1..i\tau]$$

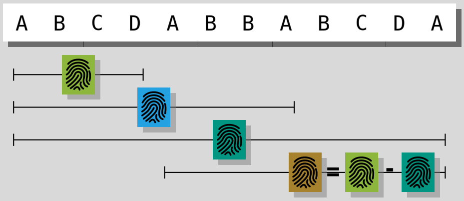
- compute $P'[i] =$  $(i, \tau i)$ for $i \in [1, n/\tau]$
- $P'[i]$ can fits in $B[i]$

Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text T over an alphabet of size σ
- let w be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\lg\sigma)$ 8 for byte alphabet
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- group the text into size-τ blocks: $B[1..n/\tau]$ with

$$B[i] = T[(i-1)\tau + 1..i\tau]$$

- compute $P'[i] = (i, \tau i)$ for $i \in [1, n/\tau]$
- $P'[i]$ can fits in $B[i]$

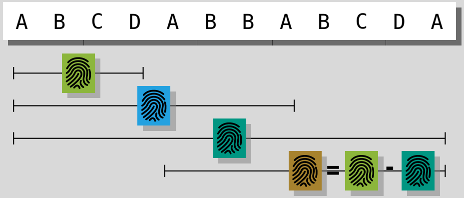


- overwrite text with fingerprints (in-place)

- all parts of text are restorable
- how?

Overwriting the Text with Fingerprints (2/2)

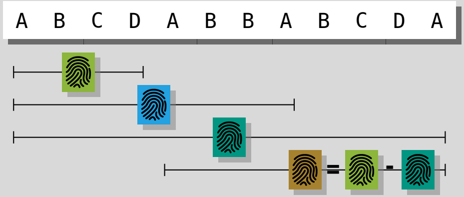
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$



- overwrite text with fingerprints (in-place)

Overwriting the Text with Fingerprints (2/2)

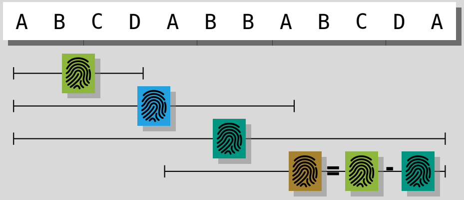
- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$



- overwrite text with fingerprints (in-place)

Overwriting the Text with Fingerprints (2/2)

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ ℹ bit vector of size n/τ

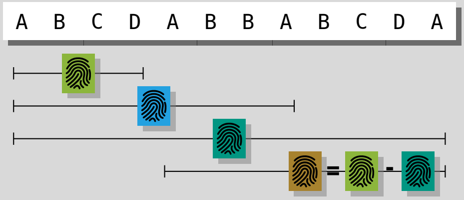


- overwrite text with fingerprints (in-place)

Overwriting the Text with Fingerprints (2/2)

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ ℹ bit vector of size n/τ
- $P'[i] =$ 🫆$(i, \tau i)$ and together with D:

$$B[i] = (P'[i] - \sigma^\tau \cdot P'[i-1] \bmod q) + D[i] \cdot q$$



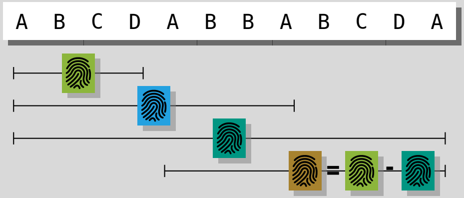
- overwrite text with fingerprints (in-place)

Overwriting the Text with Fingerprints (2/2)

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ bit vector of size n/τ
- $P'[i] = (i, \tau i)$ and together with D:

$$B[i] = (P'[i] - \sigma^\tau \cdot P'[i-1] \bmod q) + D[i] \cdot q$$

- this gives us access to the text(!)



- overwrite text with fingerprints (in-place)

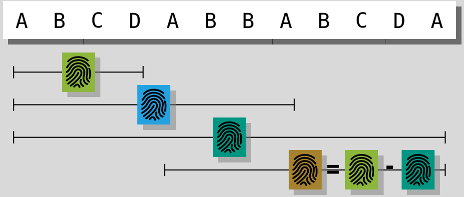
Overwriting the Text with Fingerprints (2/2)

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ ℹ bit vector of size n/τ
- $P'[i] = (i, \tau i)$ and together with D:

$$B[i] = (P'[i] - \sigma^\tau \cdot P'[i-1] \bmod q) + D[i] \cdot q$$

- this gives us access to the text(!)

- q can be chosen such that MSB of $P'[i]$ is zero w.h.p., then
- D can be stored in the MSBs



- overwrite text with fingerprints (in-place)

Overwriting the Text with Fingerprints (2/2)

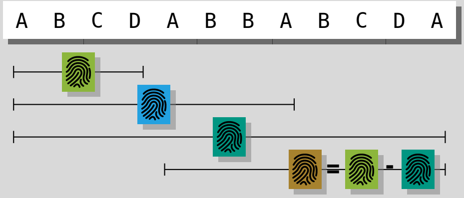
Karlsruhe Institute of Technology

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ bit vector of size n/τ
- $P'[i] = (i, \tau i)$ and together with D:

$$B[i] = (P'[i] - \sigma^\tau \cdot P'[i-1] \bmod q) + D[i] \cdot q$$

- this gives us access to the text(!)

- q can be chosen such that MSB of $P'[i]$ is zero w.h.p., then
- D can be stored in the MSBs



- overwrite text with fingerprints (in-place)

- enough to answer LCE queries

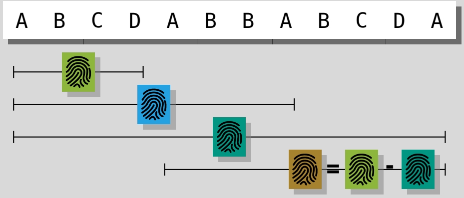
Overwriting the Text with Fingerprints (2/2)

- choose random prime $q \in [\frac{1}{2}\sigma^\tau, \sigma^\tau)$
- $B[i] = T[(i-1)\tau + 1..i\tau]$
- $\lfloor B[i]/q \rfloor \in \{0, 1\}$
- $D[i] = \lfloor B[i]/q \rfloor$ ℹ bit vector of size n/τ
- $P'[i] =$  $(i, \tau i)$ and together with D:

$$B[i] = (P'[i] - \sigma^\tau \cdot P'[i-1] \bmod q) + D[i] \cdot q$$

- this gives us access to the text(!)

- q can be chosen such that MSB of $P'[i]$ is zero w.h.p., then
- D can be stored in the MSBs



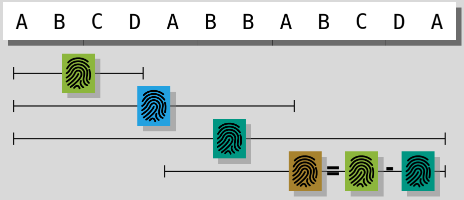
- overwrite text with fingerprints (in-place)

- enough to answer LCE queries
- how?

Answering LCE Queries with Fingerprints

LCEs with Fingerprints

- compute LCE of i and j
- exponential search until
  $(i, i+2^k) \neq (j, j+2^k)$
- binary search to find correct block m
- recompute $B[m]$ and find mismatching character



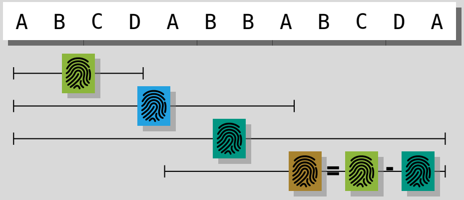
- overwrite text with fingerprints (in-place)

Answering LCE Queries with Fingerprints

LCEs with Fingerprints

- compute LCE of i and j
- exponential search until
  🫆$(i, i + 2^k) \neq$ 🫆$(j, j + 2^k)$
- binary search to find correct block m
- recompute $B[m]$ and find mismatching character

- requires $O(\lg \ell)$ time for LCEs of size ℓ



- overwrite text with fingerprints (in-place)

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n - 2\tau + 1] \colon min\{(j, j + \tau - 1) \colon j \in [i, i + \tau]\} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\}\}$$

T

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n-2\tau+1] \colon \mathit{min}\{\mathrm{fp}(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{\mathrm{fp}(i, i+\tau-1), \mathrm{fp}(i+\tau, i+2\tau-1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n - 2\tau + 1] \colon min\{\mathit{fp}(j, j + \tau - 1) \colon j \in [i, i + \tau]\} \in \{\mathit{fp}(i, i + \tau - 1), \mathit{fp}(i + \tau, i + 2\tau - 1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n-2\tau+1] \colon \mathit{min}\{(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{(i, i+\tau-1), (i+\tau, i+2\tau-1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

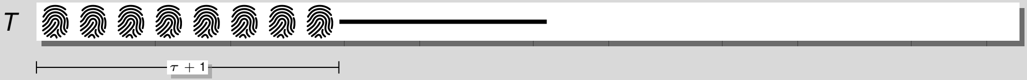
$$S = \{i \in [1, n-2\tau+1] \colon min\{\text{🫆}(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{\text{🫆}(i, i+\tau-1), \text{🫆}(i+\tau, i+2\tau-1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n-2\tau+1] \colon \mathit{min}\{\text{🫆}(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{\text{🫆}(i, i+\tau-1), \text{🫆}(i+\tau, i+2\tau-1)\}\}$$



String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n-2\tau+1] \colon min\{\text{🫆}(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{\text{🫆}(i, i+\tau-1), \text{🫆}(i+\tau, i+2\tau-1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

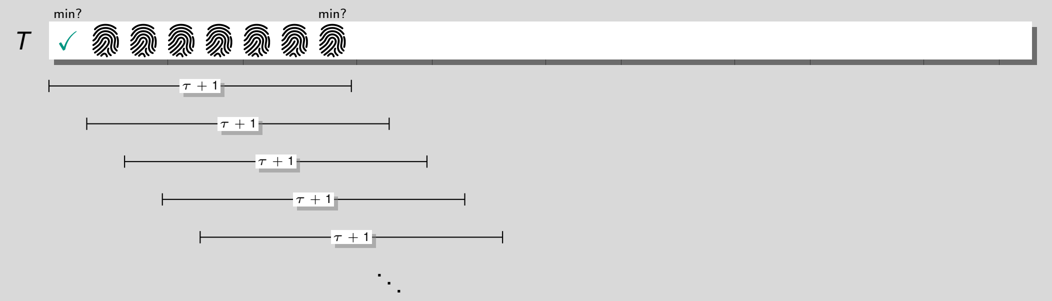
$$S = \{i \in [1, n-2\tau+1] \colon min\{\mathrm{fp}(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{\mathrm{fp}(i, i+\tau-1), \mathrm{fp}(i+\tau, i+2\tau-1)\}\}$$

String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified τ-Synchronizing Sets [KK19]

Given a text T of length n and $0 < \tau \leq n/2$, a simplified τ-synchronizing set S of T is

$$S = \{i \in [1, n-2\tau+1] \colon min\{(j, j+\tau-1) \colon j \in [i, i+\tau]\} \in \{(i, i+\tau-1), (i+\tau, i+2\tau-1)\}\}$$



String Synchronizing Sets (Simplified, 2/2)

- $|S| = \Theta(n/\tau)$ in practice (on most data sets)
- more complex definition required to obtain this size

Consistency & (Simplified) Density Property

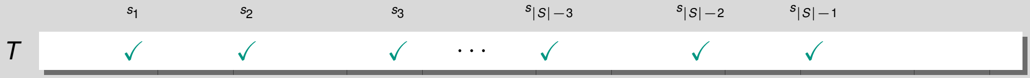
- for all $i, j \in [1, n - 2\tau + 1]$ we have

$$T[i, i+2\tau-1] = T[j, j+2\tau-1] \Rightarrow i \in S \Leftrightarrow j \in S$$

- for any τ consecutive positions there is at least one position in S

Answering LCE Queries with String Synchronizing Sets (1/2)

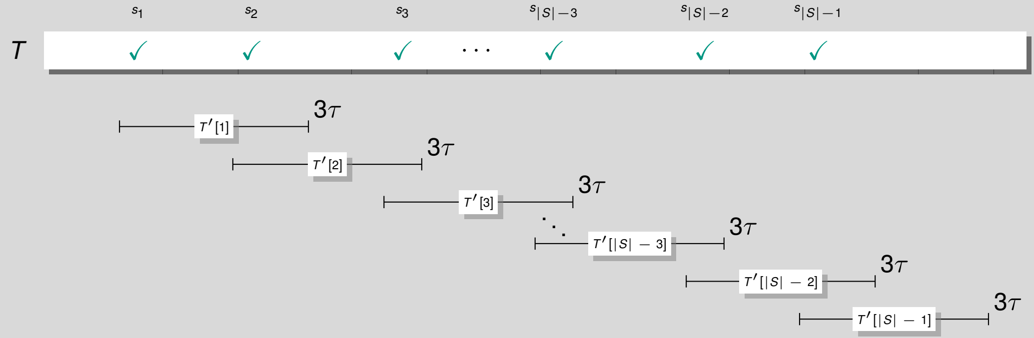
Text T' for Positions in S



Answering LCE Queries with String Synchronizing Sets (1/2)

Karlsruhe Institute of Technology

Text T' for Positions in S

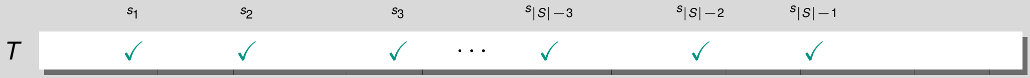


Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

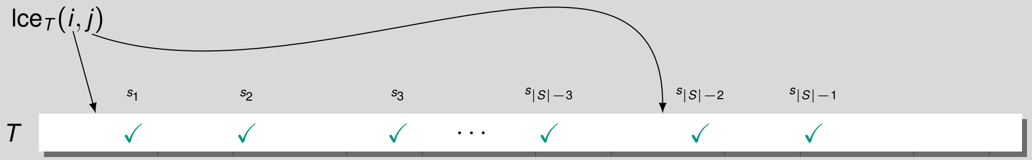


Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

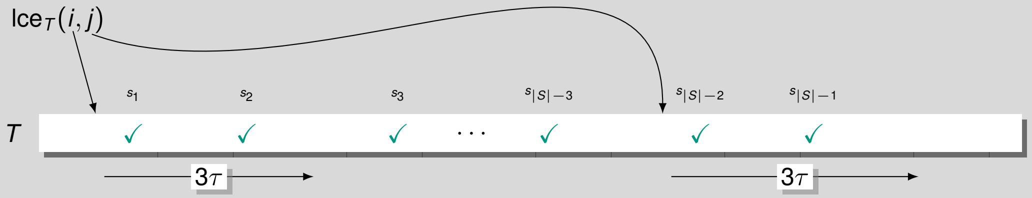


Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

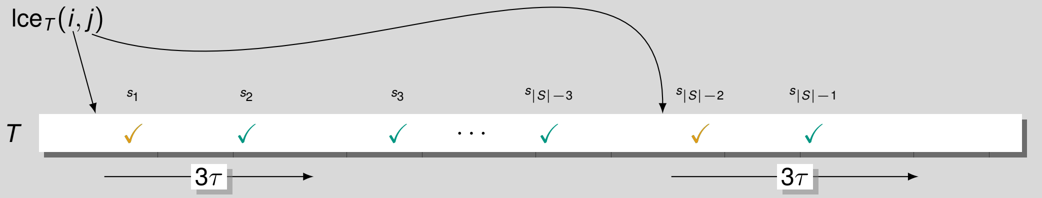


Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

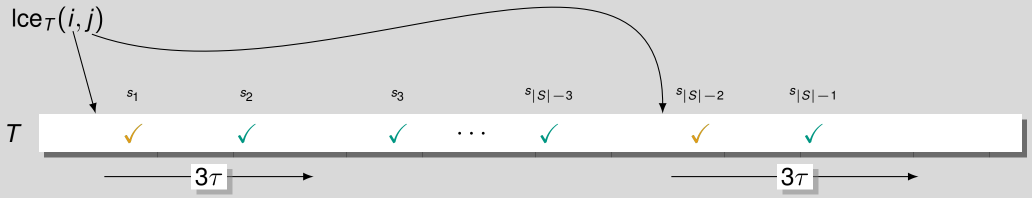


Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



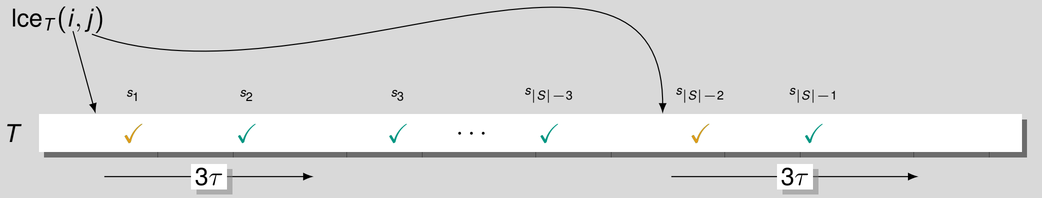
- in this example: $\mathrm{lce}_T(i, j) = s_1 - i + \mathrm{lce}_{T'}(1, |S| - 2)$

Answering LCE Queries with String Synchronizing Sets (2/2)

- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build BB LCE for T' w.r.t. length in T

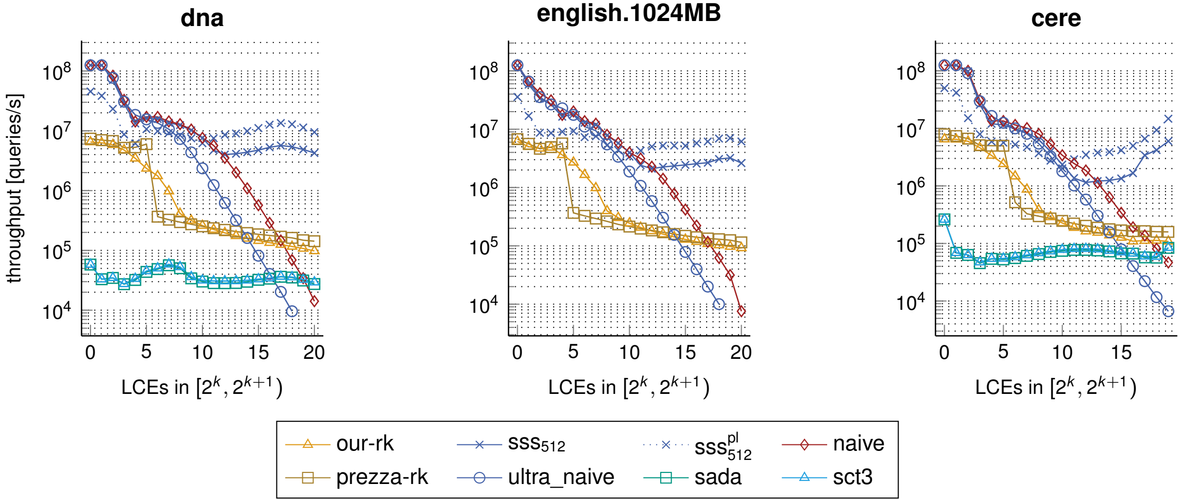
Answering Queries

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

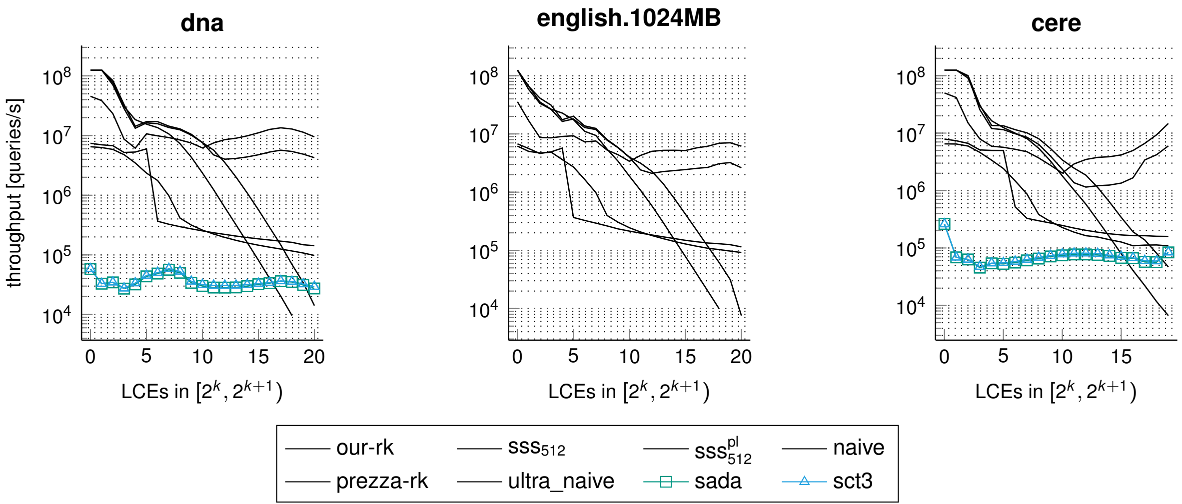


- in this example: $\mathrm{lce}_T(i,j) = s_1 - i + \mathrm{lce}_{T'}(1, |S| - 2)$
- in practice: we have a very fast static successor data structure

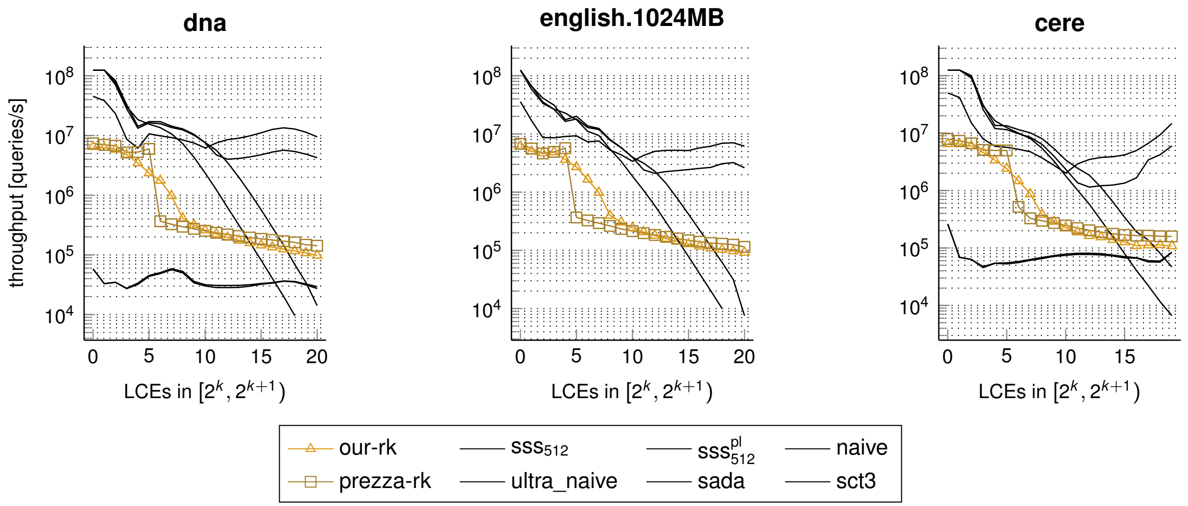
Practical Evaluation [Din+20]



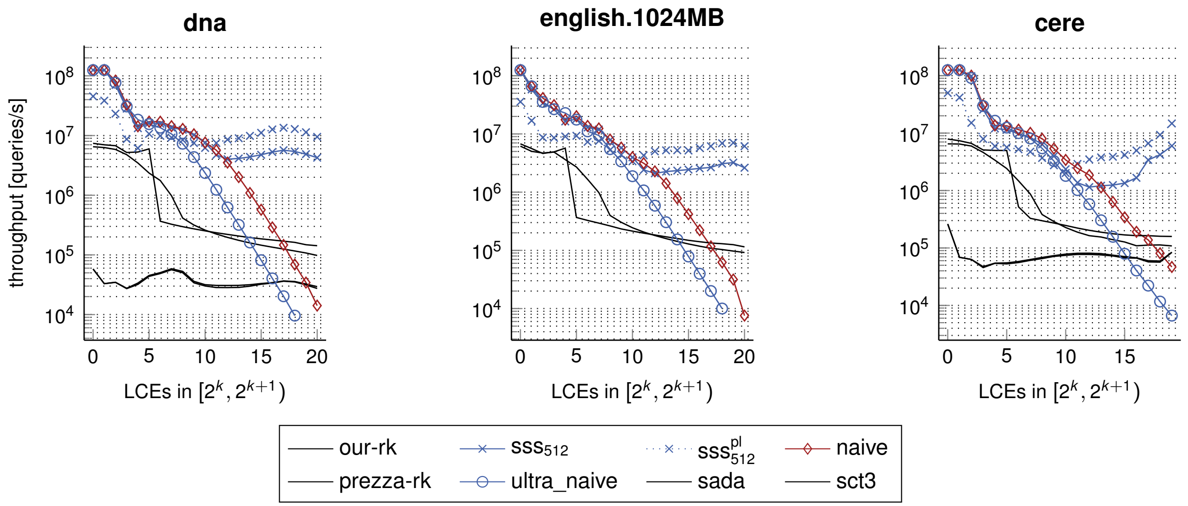
Practical Evaluation [Din+20]



Practical Evaluation [Din+20]



Practical Evaluation [Din+20]



Conclusion and Outlook

This Lecture

- longest common extension queries
- Karp-Rabin fingerprints
- string synchronizing sets

Thats all! We are (mostly) done.

Conclusion and Outlook

Karlsruhe Institute of Technology

This Lecture

- longest common extension queries
- Karp-Rabin fingerprints
- string synchronizing sets

Next Lecture

- big recap and Q&A

Thats all! We are (mostly) done.

Misc

Anmeldung Projekt

&

Discussion of the evaluation

Bibliography I

[Din+20] Patrick Dinklage, Johannes Fischer, Alexander Herlez, Tomasz Kociumaka, and Florian Kurpicz. “Practical Performance of Space Efficient Data Structures for Longest Common Extensions”. In: *ESA*. Volume 173. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 39:1–39:20. DOI: `10.4230/LIPIcs.ESA.2020.39`.

[IT09] Lucian Ilie and Liviu Tinta. “Practical Algorithms for the Longest Common Extension Problem”. In: *SPIRE*. Volume 5721. Lecture Notes in Computer Science. Springer, 2009, pages 302–309. DOI: `10.1007/978-3-642-03784-9\_30`.

[KK19] Dominik Kempa and Tomasz Kociumaka. “String Synchronizing Sets: Sublinear-Time BWT Construction and Optimal LCE Data Structure”. In: *STOC*. ACM, 2019, pages 756–767.

[KR87] Richard M. Karp and Michael O. Rabin. “Efficient Randomized Pattern-Matching Algorithms”. In: *IBM J. Res. Dev.* 31.2 (1987), pages 249–260. DOI: `10.1147/rd.312.0249`.

[Pre18] Nicola Prezza. “In-Place Sparse Suffix Sorting”. In: *SODA*. SIAM, 2018, pages 1496–1508. DOI: `10.1137/1.9781611975031.98`.