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 \(P\).

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

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: 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 \(d_1, d_2, \text{ and } d_3\)
- \(P = \text{ATA}\) 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

Definition: Range Minimum Queries

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

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

- $lcp(i, j) = \max\{k : T[i..i+k)\}$
- $lcp(i, j) = T[j..j+k) = LCP[RMQ_{LCP}(i+1, j)]$
- RMQs can be answered in $O(1)$ time and
- require $O(n)$ space
Recap: Pattern Matching with the LCP-Array (2/3)

- during binary search matched
- \( \lambda \) characters with left border \( \ell \) and
- \( \rho \) 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 \( \xi = lcp(SA[\ell], SA[i]) \) \(\mathcal{O}(1)\) time with RMQs

```
<table>
<thead>
<tr>
<th>( \ell )</th>
<th>( i )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>( \cdots )</td>
<td>( \rho )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( P[\rho] )</td>
<td>( P[\lambda] )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>
```
Recap: Pattern Matching with the LCP-Array (3/3)

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

<table>
<thead>
<tr>
<th>$\xi &gt; \lambda$</th>
<th>$\xi = \lambda$</th>
<th>$\xi &lt; \lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P[\lambda + 1] &gt; T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$</td>
<td>compare as before</td>
<td>$\xi \geq \rho$ and $P[\xi + 1] &lt; T[SA[i] + \xi]$</td>
</tr>
<tr>
<td>$\ell = i$ without character comparison</td>
<td></td>
<td>$r = i$ and $\rho = \xi$ without character comparison</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$i$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi$</td>
<td>$P[1]$</td>
<td>$P[\rho]$</td>
</tr>
<tr>
<td>$\xi$</td>
<td>$P[2]$</td>
<td></td>
</tr>
</tbody>
</table>
| $\xi$ | $P[3]$ | $T[.] 
\neq T[.]$ |
| $\xi$ | $P[\lambda]$ | $T[.]$ |
| $\xi$ | $T[.]$ | $T[.]$ |

$SA$
Old Problem, New Name

Definition: Longest Common Extensions

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

\[
lce_T(i, j) = \max\{\ell \geq 0 : T[i, i + \ell] = T[j, j + \ell]\}\]

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

Applications
(sparse) suffix sorting
approximate pattern matching . . .
Definition: Longest Common Extensions

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

$$\text{lce}_T(i, j) = \max \{ \ell \geq 0 : T[i, i + \ell] = T[j, j + \ell] \}$$

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

Application:
- (sparse) suffix sorting
- approximate pattern matching

Example:

\[ T = \begin{array}{cccccccccccc}
\end{array} \]

\[ \text{lce}_T(1, 14) = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \]

$\text{lce}_T(1, 14)$ returns the maximum length of common substrings starting at indices 1 and 14.
Old Problem, New Name

Definition: Longest Common Extensions
Given a text $T$ of size $n$ over an alphabet of size $\sigma$, construct data structure that answers for $i, j \in [1, n]$

\[ \text{Ice}_T(i, j) = \max \{ \ell \geq 0 : T[i, i+\ell) = T[j, j+\ell) \} \]

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

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

\[
\begin{array}{ccccccccccc}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\
\end{array}
\]

\[ \text{Ice}_T(1, 14) = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \]
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
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

Las Vegas Algorithm
- returns correct result only expected running time

Some Monte Carlo algorithms can be turned into Las Vegas algorithms depending on correctness check. All Monte Carlo algorithms presented today can be turned into Las Vegas algorithms.
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 fingerprints of strings
- application not limited to LCEs
Randomized String Matching

- compute fingerprints of strings
- application not limited to LCEs

**Definition: Karp-Rabin Fingerprint [KR87]**

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

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

1. $(x + y) \mod z = z \mod z + y \mod z \pmod{z}$
Randomized String Matching

- compute fingerprints of strings
- application not limited to LCEs

Definition: Karp-Rabin Fingerprint [KR87]

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

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

$(x + y) \mod z = z \mod z + y \mod z \quad (\mod z)$

- if $T[i..i + \ell] = T[j..j + \ell]$, then
  $$\mathcal{F}(i, i + \ell) = \mathcal{F}(j, j + \ell)$$
- if $T[i..i + \ell] \neq T[j..j + \ell]$, then
  $$\text{Prob}(\mathcal{F}(i, i + \ell) = \mathcal{F}(j, j + \ell)) \in O\left(\frac{\ell \log \sigma}{n^c}\right)$$
- prime has to be chosen uniformly at random
- how to turn it into Las Vegas algorithm?
Randomized String Matching

- compute fingerprints of strings
- application not limited to LCEs

**Definition: Karp-Rabin Fingerprint [KR87]**

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

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

- if $T[i..i + \ell] = T[j..j + \ell]$, then
  $$
  \fingerprint(i, i + \ell) = \fingerprint(j, j + \ell)
  $$
- if $T[i..i + \ell] \neq T[j..j + \ell]$, then
  $$
  \text{Prob}
  \left( \fingerprint(i, i + \ell) = \fingerprint(j, j + \ell) \right)
  \in O\left( \frac{\ell \log \sigma}{n^c} \right)
  $$

- 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 $\sigma$
Overwriting the Text with Fingerprints (1/2) [Pre18]

- given a text $T$ over an alphabet of size $\sigma$
- let $w$ be size of a computer word e.g., 64 bit
Overwriting the Text with Fingerprints (1/2) [Pre18]

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

let \( w \) be size of a computer word \( \mathbb{1} \) e.g., 64 bit

choose \( \tau \in \Theta(\frac{w}{\lg \sigma}) \) \( \mathbb{1} \) 8 for byte alphabet

choose random prime \( q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right) \)
Overwriting the Text with Fingerprints (1/2) [Pre18]

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

$$B[i] = T[(i - 1)\tau + 1..i\tau]$$
given a text $T$ over an alphabet of size $\sigma$

let $w$ be size of a computer word \( \epsilon \) e.g., 64 bit

choose $\tau \in \Theta(\frac{w}{\lg \sigma}) \epsilon 8$ for byte alphabet

choose random prime $q \in [\frac{1}{2} \sigma^{\tau}, \sigma^{\tau})$

group the text into size-$\tau$ blocks: $B[1..n/\tau]$ with

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

compute $P'[i] = \mathcal{F}(i, \tau i)$ for $i \in [1, n/\tau]$
given a text $T$ over an alphabet of size $\sigma$

let $w$ be size of a computer word e.g., 64 bit

choose $\tau \in \Theta(w / \log \sigma)$ 8 for byte alphabet

choose random prime $q \in \left[\frac{1}{2} \sigma^\tau, \sigma^\tau \right)$

group the text into size-$\tau$ blocks: $B[1..n/\tau]$ with

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

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

$P'[i]$ can fits in $B[i]$
given a text $T$ over an alphabet of size $\sigma$

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

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

- compute $P'[i] = \bigotimes(i, \tau i)$ 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 $q \in \left[\frac{1}{2} \sigma^\tau, \sigma^\tau\right)$
- $B[i] = T[(i - 1)\tau + 1..i\tau]$
choose random prime $q \in \left[\frac{1}{2} \sigma^\tau, \sigma^\tau\right)$

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

$\lfloor B[i]/q \rfloor \in \{0, 1\}$

- overwrite text with fingerprints (in-place)
choose random prime $q \in \left[\frac{1}{2} \sigma^\tau, \sigma^\tau\right)$

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

$\lfloor B[i]/q \rfloor \in \{0, 1\}$

$D[i] = \lfloor B[i]/q \rfloor$ bit vector of size $n/\tau$

- overwrite text with fingerprints (in-place)
choose random prime $q \in \left[\frac{1}{2}\sigma^\tau, \sigma^\tau\right)$

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

$\lfloor B[i]/q \rfloor \in \{0, 1\}$

$D[i] = \lfloor B[i]/q \rfloor \quad \text{bit vector of size } n/\tau$

$P'[i] = \begin{cases} 0 \end{cases}(i, \tau i) \text{ and together with } D:

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

- overwrite text with fingerprints (in-place)
choose random prime \( q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right) \)

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

\[ \lfloor B[i]/q \rfloor \in \{0, 1\} \]

\[ D[i] = \lfloor B[i]/q \rfloor \] bit vector of size \( n/\tau \)

\[ P'[i] = \mathcal{A}(i, \tau i) \] and together with \( D \):

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

this gives us access to the text(!)

overwrite text with fingerprints (in-place)
choose random prime \( q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right) \)

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

\( \lfloor B[i]/q \rfloor \in \{0, 1\} \)

\( D[i] = \lfloor B[i]/q \rfloor \) bit vector of size \( n/\tau \)

\( P'[i] = \langle i, \tau i \rangle \) and together with \( D \):

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

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

---

**Overwriting the Text with Fingerprints (2/2)**

- Overwrite text with fingerprints (in-place)
- block
- block
- block
choose random prime $q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right)$

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

$\lfloor B[i]/q \rfloor \in \{0, 1\}$

$D[i] = \lfloor B[i]/q \rfloor$  bit vector of size $n/\tau$

$P'[i] = \mathcal{R}(i, \tau i)$ and together with $D$:

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

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
choose random prime $q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right)$

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

$\lfloor B[i]/q \rfloor \in \{0, 1\}$

$D[i] = \lfloor B[i]/q \rfloor \text{ bit vector of size } n/\tau$

$P'[i] = \oplus(i, \tau i)$ and together with $D$:

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

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?
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)
LCEs with Fingerprints

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

- requires $O(\lg \ell)$ time for LCEs of size $\ell$
String Synchronizing Sets (Simplified, 1/2)

**Definition: Simplified \( \tau \)-Synchronizing Sets [KK19]**

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

\[
S = \{ i \in [1, n - 2\tau + 1] : \min \{ (j, j + \tau - 1) : j \in [i, i + \tau] \} \in \{ (i, i + \tau - 1), (i + \tau, i + 2\tau - 1) \} \}
\]
Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{ i \in [1, n - 2\tau + 1] : \min\{(j, j + \tau - 1) : j \in [i, i + \tau] \} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\}\}$$
String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{ i \in [1, n - 2\tau + 1] : \min\{ S(j, j + \tau) : j \in [i, i + \tau] \} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\} \}$$
Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{ i \in [1, n - 2\tau + 1] : \min\{ \min\{(j, j + \tau - 1) : j \in [i, i + \tau]\} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\} \}$$
String Synchronizing Sets (Simplified, 1/2)

**Definition: Simplified $\tau$-Synchronizing Sets [KK19]**

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

$$S = \{i \in [1, n - 2\tau + 1]: \min\{\min\{j, j + \tau - 1\}: j \in [i, i + \tau]\} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\}\}$$
Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{i \in [1, n - 2\tau + 1]: \min\{(j, j + \tau - 1): j \in [i, i + \tau]\} \in \{(i, i + \tau - 1), (i + \tau, i + 2\tau - 1)\}\}$$
Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{i \in [1, n - 2\tau + 1]: \min\{(j, j + \tau - 1): j \in [i, i + \tau]\} \in \{(i, i - \tau - 1), (i + \tau, i + 2\tau - 1)\}\}$$
Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{i \in [1, n-2\tau+1]: \min\{(j, j+\tau-1): j \in [i, i+\tau]\} \in \{(i, i+\tau-1), (i+\tau, i+2\tau-1)\}\}$$
String Synchronizing Sets (Simplified, 1/2)

Definition: Simplified $\tau$-Synchronizing Sets [KK19]

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

$$S = \{ i \in [1, n-2\tau+1] : \min\{ (j, j+\tau-1) : j \in [i, i+\tau] \} \in \{ (i, i+\tau-1), (i+\tau, i+2\tau-1) \} \}$$
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 $\tau$ consecutive positions there is at least one position in $S$
### Answering LCE Queries with String Synchronizing Sets (1/2)

**Text $T'$ for Positions in $S$**

| $s_1$ | $s_2$ | $s_3$ | $|S|-3$ | $|S|-2$ | $|S|-1$ |
|-------|-------|-------|---------|---------|---------|
| ✓     | ✓     | ✓     | ⋯       | ✓       | ✓       |
Text $T'$ for Positions in $S$

$T$  

$T'[1] 3\tau$  

$T'[2] 3\tau$  

$T'[3] 3\tau$  

$\cdots$  

$T'[|S| - 3] 3\tau$  

$T'[|S| - 2] 3\tau$  

$T'[|S| - 1] 3\tau$  

$T$  

$T'_{s_1}$  

$T'_{s_2}$  

$T'_{s_3}$  

$T'_{|S| - 3}$  

$T'_{|S| - 2}$  

$T'_{|S| - 1}$
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 $3\tau$ characters
- if equal find successors of $i$ and $j$ in $S$
- compute LCE of successors in $T'$
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 \( 3\tau \) characters
• if equal find successors of \( i \) and \( j \) in \( S \)
• compute LCE of successors in \( T' \)
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 \( 3\tau \) characters
- if equal find successors of \( i \) and \( j \) in \( S \)
- compute LCE of successors in \( T' \)
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 $3\tau$ 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)**

- $\text{lce}_{T}(i, j)$
- $s_1$ $s_2$ $s_3$ $s_{|S| - 3}$ $s_{|S| - 2}$ $s_{|S| - 1}$
- $T$ $\checkmark$ $\checkmark$ $\checkmark$ $\cdots$ $\checkmark$ $\checkmark$ $\checkmark$
- $3\tau$ $\rightarrow$ $\rightarrow$ $3\tau$
in practice, we sort the substrings
characters of $T'$ are the ranks of substrings
build BB LCE for $T'$ w.r.t. length in $T$

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

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

Answering Queries

$\text{lce}_T(i, j)$

$T$

$s_1$ $s_2$ $s_3$ $s_{|S| - 3}$ $s_{|S| - 2}$ $s_{|S| - 1}$
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 $3\tau$ characters
- if equal find successors of $i$ and $j$ in $S$
- compute LCE of successors in $T'$

in this example: $lce_T(i, j) = s_1 - i + lce_{T'}(1, |S| - 2)$

in practice: we have a very fast static successor data structure
Practical Evaluation [Din+20]

LCEs in $[2^k, 2^{k+1})$

DNA

english.1024MB

cere

throughput [queries/s]

our-rk

SSS512

prezza-rk

ultra_naive

naive

sada

sct3
Practical Evaluation [Din+20]

![Graphs showing throughput in queries/s for different datasets and algorithms](image)

- **DNA**
  - LCEs in \([2^k, 2^{k+1})\)
  - Throughput in queries/s

- **english.1024MB**
  - LCEs in \([2^k, 2^{k+1})\)
  - Throughput in queries/s

- **cere**
  - LCEs in \([2^k, 2^{k+1})\)
  - Throughput in queries/s

Legend:
- our-rk
- sss\textsubscript{512}
- sss\textsuperscript{pl}\textsubscript{512}
- naive
- prezza-rk
- ultra_naive
- sada
- sct3
Practical Evaluation [Din+20]

![Graphs showing throughput of different algorithms for DNA and English-1024MB datasets, with LCEs in $[2^k, 2^{k+1})$.](image)

- **DNA**
  - `our-rk`
  - `ssS512`
  - `ssS512pl`
  - `naive`
  - `prezza-rk`
  - `ultra_naive`
  - `sada`
  - `sct3`

- **English-1024MB**
  - `our-rk`
  - `ssS512`
  - `ssS512pl`
  - `naive`
  - `prezza-rk`
  - `ultra_naive`
  - `sada`
  - `sct3`

- **Cere**
  - `our-rk`
  - `ssS512`
  - `ssS512pl`
  - `naive`
  - `prezza-rk`
  - `ultra_naive`
  - `sada`
  - `sct3`
Practical Evaluation \cite{Din+20}

![Graphs showing throughput vs. LCEs for different datasets and algorithms: dna, english.1024MB, cere.](image)

- **dna**: Throughput in queries/s for LCEs in $[2^k, 2^{k+1})$.
- **english.1024MB**: Throughput in queries/s for LCEs in $[2^k, 2^{k+1})$.
- **cere**: Throughput in queries/s for LCEs in $[2^k, 2^{k+1})$.

Legend:
- our-rk
- SSS$_{512}$
- SSS$_{512}^{pl}$
- naive
- prezza-rk
- ultra_naive
- sada
- sct3
Thats all! We are (mostly) done.

This Lecture
- longest common extension queries
- Karp-Rabin fingerprints
- string synchronizing sets
Conclusion and Outlook

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

Next Lecture
- big recap and Q&A

That's all! We are (mostly) done.
Anmeldung Projekt
&
Discussion of the evaluation
Bibliography I


