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 $P$. 
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 \(P\).

- \(d_1 = ATA\)
- \(d_2 = TAAA\)
- \(d_3 = TATA\)

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: Inverted Index and List Encodings

**Definition: Inverted Index**

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

1. The old night keeper keeps the keep in the town
2. In the big old house in the big old gown
3. The house in the town had the big old keep
4. Where the old night keeper never did sleep
5. The night keeper keeps the keep in the night
6. And keeps in the dark and sleeps in the light
Recap: Inverted Index and List Encodings

Definition: Inverted Index

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

List Encodings

- ∆-encoding
- unary- and ternary-encoding
- Elia-γ and -δ-encoding
- Golomb- and Fibonacci-encoding

1 The old night keeper keeps the keep in the town
2 In the big old house in the big old gown
3 The house in the town had the big old keep
4 Where the old night keeper never did sleep
5 The night keeper keeps the keep in the night
6 And keeps in the dark and sleeps in the light

<table>
<thead>
<tr>
<th>term</th>
<th>$f_t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1,2,3,5,6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Recap: Inverted Index and List Encodings

Definition: Inverted Index

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

1. The old night keeper keeps the keep in the town
2. In the big old house in the big old gown
3. The house in the town had the big old keep
4. Where the old night keeper never did sleep
5. The night keeper keeps the keep in the night
6. And keeps in the dark and sleeps in the light

<table>
<thead>
<tr>
<th>term</th>
<th>$f_t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1, 2, 3, 5, 6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Recap: Inverted Index and List Encodings

**Definition: Inverted Index**

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$:
- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

**List Encodings**

- $\Delta$-encoding
- unary- and ternary-encoding
- Elia-$\gamma$ and -$\delta$-encoding
- Golomb- and Fibonacci-encoding

---

<table>
<thead>
<tr>
<th>term</th>
<th>$t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1,2,3,5,6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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

$$RMQ_A(\ell, r) = \arg \min \{ A[k] : k \in [\ell, r] \}$$
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 $\xi = \text{lcp}(SA[\ell], SA[i]) \in O(1)$ time with RMQs

\[
\begin{array}{c|c|c}
\ell & i & r \\
\hline
P[1] & \ldots & P[\rho] \\
\lambda & \ldots & \lambda \\
p[\lambda] & \ldots & p[\lambda] \\
\end{array}
\]
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
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
**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.

### Applications

- (sparse) suffix sorting
- approximate pattern matching

---

| $T$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
|     | A | B | C | D | A | B | C | C | D | B | C | C | B | A | B | C | D | A | D | A |

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

Applications

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

better trade-off
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 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

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

$$\Phi(i, j) = \Phi(i, j + \ell) \mod q$$ if $T[i..i+\ell] = T[j..j+\ell]$, then

$$\Phi(i, i+\ell) = \Phi(j, j+\ell) \mod q$$

$$\Phi(i, j) \in O(\ell \lg \sigma n^c)$$

Prime has to be chosen uniformly at random how to turn it into Las Vegas algorithm? Example on the board/teacher.
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

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

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

$$
\mathcal{H}(i, i+\ell) = \mathcal{H}(j, j+\ell)
$$

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

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

- 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

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

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

$$\mathcal{F}(i, i + \ell) = \mathcal{F}(j, j + \ell)$$

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?

- example on the board 📧
given a text $T$ over an alphabet of size $\sigma$
given a text $T$ over an alphabet of size $\sigma$
let $w$ be size of a computer word e.g., 64 bit
given a text $T$ over an alphabet of size $\sigma$
- let $w$ be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w / \lg \sigma)$ for byte alphabet
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 $\tau \in \Theta(w / \log \sigma) \approx 8$ for byte alphabet
- choose random prime $q \in \left[\lfloor \frac{1}{2} \sigma^\tau \right, \sigma^\tau \right)$
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 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-$\tau$ 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 $\tau \in \Theta(w/\lg \sigma)$ 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 $P'[i] = \mathcal{H}(i, \tau i)$ 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 $\sigma$
- let $w$ be size of a computer word e.g., 64 bit
- choose $\tau \in \Theta(w/\log \sigma)$ 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 $P'[i] = \mathbb{H}(i, \tau i)$ 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?
choose random prime $q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right)$

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

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

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

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

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

- choose random prime \( q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right) \)
- \( B[i] = T[(i - 1)\tau + 1..i\tau] \)
- \( \lfloor B[i]/q \rfloor \in \{0, 1\} \)
- \( D[i] = \lfloor B[i]/q \rfloor \) bit vector of size \( n/\tau \)
- \( P'[i] = \mathbb{f}(i, \tau i) \) 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)

\( q \) can be chosen such that MSB of \( P'[i] \) is zero w.h.p., then \( D \) can be stored in the MSBs.
choose random prime $q \in \left[\frac{1}{2}\sigma^\tau, \sigma^\tau\right)$

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

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

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

$P'[i] = \mathbf{0}(i, \tau i)$ 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 \( n/\tau \)

\[ P'[i] = (i, \tau i) \] 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)
choose random prime $q \in \left[\frac{1}{2} \sigma^\tau, \sigma^\tau\right)$

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

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

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

$P'[i] = \overline{i}(i, \tau i)$ 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 \( q \in [\frac{1}{2} \sigma^\tau, \sigma^\tau) \)

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

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

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

\[ P'[i] = \mathbb{F}(i, \tau i) \text{ and together with } D: \]

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

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)

A B C D A B B A B C D A
LCEs with Fingerprints

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

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

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

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

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

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

|       | $s_1$ | $s_2$ | $s_3$ | $|S| - 3$ | $|S| - 2$ | $|S| - 1$ |
|-------|-------|-------|-------|-----------|-----------|-----------|
| $T$   | ✓     | ✓     | ✓     | ⋯         | ✓         | ✓         |
Answering LCE Queries with String Synchronizing Sets (1/2)

Text $T'$ for Positions in $S$

$T'$

| $s_1$ | $s_2$ | $s_3$ | $s_{|S|-3}$ | $s_{|S|-2}$ | $s_{|S|-1}$ |
|-------|-------|-------|-------------|-------------|-------------|
| ✓     | ✓     | ✓     | ✓           | ✓           | ✓           |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |

$T'$

| $|S|-3$ | $|S|-2$ | $|S|-1$ |
|---------|---------|---------|
| $3\tau$ | $3\tau$ | $3\tau$ |
in practice, we sort the substrings
- characters of \( T' \) are the ranks of substrings
- build BB LCE for \( T' \) w.r.t. length in \( T \)

**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 $T'$

$LCE_T(i, j)$

$T$

$s_1$  $s_2$  $s_3$  $s_{|S| -3}$  $s_{|S| -2}$  $s_{|S| -1}$

$✓$  $✓$  $✓$  $✓$  $✓$  $✓$  $✓$
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 $T'$

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

![Diagram](image-url)

- $lce_{T}(i, j) = s1 - i + lce_{T'}(1, |S| - 2)$
in practice, we sort the substrings
character 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 \( T' \)

\[
lce_T(i, j) = s_1 - i + lce_{T'}(1, |S| - 2)
\]
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 $3\tau$ characters
- if equal find successors of $i$ and $j$ in $S$
- compute LCE of successors in $T'$

- in this example: $\text{lce}_T(i, j) = s_1 - i + \text{lce}_{T'}(1, |S| - 2)$
- in practice: we have a very fast static successor data structure
Practical Evaluation [Din+20]

DNA

English.1024MB

Cere

Throughput [queries/s]

LCEs in $[2^k, 2^{k+1})$
Practical Evaluation [Din+20]

**Graphs:**
- **dna**
- **english.1024MB**
- **cere**

**Axes:**
- x-axis: LCEs in $[2^k, 2^{k+1})$
- y-axis: throughput [queries/s]

**Legend:**
- our-rk
- sss$_{512}$
- sss$_{512}^p$
- naive
- prezze-rk
- ultra_naive
- sada
- sct3
Practical Evaluation [Din+20]

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

- dna
- english.1024MB
- cere

Throughput [queries/s]
Practical Evaluation [Din+20]

Throughput [queries/s] vs. LCEs in $[2^k, 2^{k+1})$

- **dna**
  - Our-rk
  - SSS
  - ultra_naive
  - naive
  - prezza-rk
  - sada
  - sct3

- **english.1024MB**

- **cere**

Institute for Theoretical Informatics, Algorithm Engineering
Conclusion and Outlook

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

That's all! We are (mostly) done.
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.