Text Indexing

Lecture 13: Longest Common Extensions

Florian Kurpicz
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 $\xi = lcp(SA[\ell], SA[i]) \in O(1)$ time with RMQs
Recap: Pattern Matching with the LCP-Array (3/3)

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

$\xi > \lambda$
- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ 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

**Applications**

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

![Diagram](image-url)
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
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

**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

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

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

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

- example on the board

\[ (x + y) \mod z = z \mod z + y \mod z \ (\mod z) \]
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 $P'[i] = \bigcirc(i, \tau i)$ 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 \( q \in \left[ \frac{1}{2} \sigma^\tau, \sigma^\tau \right) \)

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

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

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

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

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

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?
compute LCE of $i$ and $j$

exponential search until $f(i, i + 2^k) \neq f(j, j + 2^k)$

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

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

\[
\begin{array}{cccccc}
\text{\( s_1 \)} & \text{\( s_2 \)} & \text{\( s_3 \)} & \text{\( s_{|S|-3} \)} & \text{\( s_{|S|-2} \)} & \text{\( s_{|S|-1} \)} \\
\checkmark & \checkmark & \checkmark & \cdots & \checkmark & \checkmark \\
\end{array}
\]

\[ T' = \{ T'[1], T'[2], T'[3], \ldots, T'[|S|-1] \} \]
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 $T'$

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

- 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

cereal
Warning

This is just a very succinct overview.
Please refer to the lecture slides for more details.
Tries & Suffix Trees

Trie Representations
- different trie representations
- space-time trade-off

Suffix Tree (Compact Trie)
Suffix Array

Given a text $T$ of length $n$, the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

$$T[SA[i]..n] \leq T[SA[j]..n]$$

**SAIS**
- linear time suffix array construction
- induced copying and recursion
  - classification
  - sorting special suffixes
  - inducing other suffixes

**SA Construction in EM**
- Prefix Doubling
- DC3
LCP-Array & LCE-Queries

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>SA</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>LCP</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

- speed up pattern matching in suffix array
- suffix tree construction
- compression

Longest Common Extensions
- lcp-value between any suffix
- scan or RMQ
- Rabin-Karp fingerprints
- string synchronizing sets
Entropy

Given a text $T$ of length $n$ over an alphabet $\Sigma = [1, \sigma]$ and its histogram $Hist$, then

$$H_k = \left( \frac{1}{n} \right) \sum_{S \in \Sigma^k} |T_S| \cdot H_0(T_S)$$

Huffman Codes

- variable length codes
- more frequent characters get shorter codes
- canonical Huffman-codes
- Shannon-Fano codes can be worse, but
- are still optimal

LZ77

$T = \text{abababbbaba}\$

- $f_1 = a$
- $f_2 = b$
- $f_3 = \text{abab}$
- $f_4 = \text{bbb}$
- $f_5 = \text{aba}$
- $f_6 = \$

LZ78

$T = \text{abababbbaba}\$

- $f_1 = a$
- $f_2 = b$
- $f_3 = \text{ab}$
- $f_4 = \text{abb}$
- $f_5 = \text{bb}$
- $f_6 = \text{aba}$
- $f_7 = \$
Burrows-Wheeler Transform

Given a text $T$ of length $n$ and its suffix array $SA$, for $i \in [1, n]$ the **Burrows-Wheeler transform** is

$$BWT[i] = \begin{cases} T[SA[i] - 1] & \text{if } SA[i] > 1 \\ \$ & \text{if } SA[i] = 1 \end{cases}$$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$BWT$</td>
<td>a</td>
<td>b</td>
<td>$$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

**LF-Mapping**

Given a $BWT$, its $C$-array, and its $rank$-Function, then

$$LF(i) = C[BWT[i]] + rank_{BWT[i]}(i)$$

- transform back to text
- used in backwards search

**Compression using BWT**

- move-to-front
- run-length compression
### Wavelet Tree

#### Wavelet Tree

- generalize rank and select to alphabets of size $> 2$

#### Wavelet Matrix

- build over text compressed with canonical Huffman codes

### Compression

- rank and select queries on bit vectors in $O(1)$ time and $o(n)$ space

### Bit Vectors
Function $\text{BackwardsSearch}(P[1..n], C, \text{rank})$:

1. $s = 1, \ e = n$
2. for $i = m, \ldots, 1$ do
3.   $s = C[P[i]] + \text{rank}_{P[i]}(s - 1) + 1$
4.   $e = C[P[i]] + \text{rank}_{P[i]}(e)$
5. if $s > e$ then
6.   return $\emptyset$
7. return $[s, e]$

### FM-Index
- use (compressed wavelet tree for rank)
- compress bit vectors further

### r-Index
- store lots of arrays
- containing information for each run
- size proportional to number of runs
- queries become harder

### Move Data Structure
- make use of “same” intervals in BWT and first row
- constant time mapping on balanced input/output intervals
- balancing with blowup $\leq 2$ achievable
Compressed Indices

Block Tree
- answer rank and select queries
- size proportional to number of LZ-factors

Number of Runs and LZ-Factors
Let $T$ be a text of length $n$, then

$$r(T) \in O(z(T) \log^2 n)$$
Document Retrieval

**Document Listing**
- optimal with document array and chain array

<table>
<thead>
<tr>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SA</th>
<th>15</th>
<th>14</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>3</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>11</th>
<th>1</th>
<th>12</th>
<th>2</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>CA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

- $P = TA$
The old night keeper keeps the keep in the town
In the big old house in the big old gown
The house in the town had the big old keep
Where the old night keeper never did sleep
The night keeper keeps the keep in the night
And keeps in the dark and sleeps in the light

<table>
<thead>
<tr>
<th>term</th>
<th>$f_i$</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>
Longest Common Extensions

**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

**Definition: Simplified \(\tau\)-Synchronizing Sets**

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

\[
S = \{ i \in [1, n - 2\tau + 1] : \min\{ f_j(i, j + \tau - 1) : j \in [i, i + \tau] \} \in \{ f(i, i + \tau - 1), f(i + \tau, i + 2\tau - 1) \} \}
\]
## Conclusion and Outlook

### This Lecture
- longest common extension queries
- Karp-Rabin fingerprints
- string synchronizing sets
- big recap and Q&A

### Next Week
- project presentation

Thats all! We are (mostly) done.