Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]
Given a text $T$ of length $n$, the suffix array (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

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

Definition: Longest Common Prefix Array
Given a text $T$ of length $n$ and its SA, the LCP-array is defined as

$$LCP[i] = \begin{cases} 
0 & i = 1 \\
\max\{\ell : T[SA[i]..SA[i] + \ell) = T[SA[i-1]..SA[i-1] + \ell)\} & i \neq 1 
\end{cases}$$
### Naive Computation of the LCP-Array

<table>
<thead>
<tr>
<th>Task</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>given: text $T$ of length $n$ and its suffix array</td>
<td>naive construction requires $O(n^2)$ time</td>
</tr>
<tr>
<td>wanted: longest common prefix array</td>
<td>all-a texts are worst case</td>
</tr>
</tbody>
</table>

#### Naive Construction
- for each pair $(SA[i - 1], SA[i])$
- compare $T[SA[i - 1] + \ell]$ and $T[SA[i] + \ell]$ until mismatch

- here $LCP[1] = 0, LCP[1] = 0,$ and $LCP[i] = i - 2$
- only distinguishable character is $\$
Properties of the LCP-Array

- do not compare all suffixes naively
- compare only unknown parts

**Lemma: Values in LCP-array**

Given a text $T$ of length $n$, its suffix array $SA$ and $LCP$-array $LCP$, then

$$\exists i \in [1, n): LCP[i] = \ell > 0 \Rightarrow \exists j \in [1, n): LCP[j] = \ell - 1$$

**Proof (Sketch)**

- let $LCP[i] = k > 0$
- $T[SA[i]..SA[i] + k) = T[SA[i - 1]..SA[i - 1] + k)$
- $T[SA[i] + 1..SA[i] + k) = T[SA[i - 1] + 1..SA[i - 1] + k)$
- not necessarily next to each other in $SA$
The Inverse Suffix Array

Definition: Inverse Suffix Array

Given a suffix array $SA$ of length $n$, the inverse suffix array ($ISA = SA^{-1}$) is

$$ISA[SA[i]] = i$$

for $n \in [1..n]$

- inverse permutation ⬠ as hinted by the name
- where is a suffix in the suffix array

<table>
<thead>
<tr>
<th>$T$</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$ISA$</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>11</td>
<td>13</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Function LinearTimeLCP(T, SA[1..n]):
1. for i = 1, ..., n do $ISA[SA[i]] = i$
2. $\ell = 0$, $LCP[1] = 0$
3. for i = 1, ..., n do
   4. if $ISA[i] \neq 1$ then
      5. $j = SA[ISA[i] - 1]$
      6. while $T[i + \ell] = T[j + \ell]$ do
         7. $\ell = \ell + 1$
      8. $LCP[ISA[i]] = \ell$
      9. $\ell = \max\{0, \ell - 1\}$
6. return $LCP$

- compute suffixes in text order
- use $ISA$ to find lex. smaller suffix

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>13 12 1 9 6 3 11 2 10 7 4 8 5</td>
</tr>
<tr>
<td>3 8 6 11 13 5 10 12 4 9 7 2 1</td>
</tr>
<tr>
<td>0 \</td>
</tr>
</tbody>
</table>

- correctness and running time

---

Linear Time Construction [Kas+01]

---

2023-11-13 Florian Kurpicz | Text Indexing | 04 LCP-Array
Institute of Theoretical Informatics, Algorithm Engineering
The $\Phi$-Array

Definition: $\Phi$-Array

Given a text $T$ of length $n$ and its suffix array $SA$, the $\Phi$-array is defined (for $i > 1$) as

$$\Phi[SA[i]] = SA[i - 1]$$

- $\Phi[i]$ gives suffix that is needed for comparison
- not a permutation of $SA$
Better Linear Time Construction [KMP09]

Function $\Phi$-Algorithm($T$, $SA[1..n]$):

1. $\Phi[n] = SA[n] \circ SA[1] = n$; $T$ has sentinel
2. for $i = 2, \ldots, n$ do $\Phi[SA[i]] = SA[i-1]$
3. $\ell = 0$
4. for $i = 1, \ldots, n$ do
   5. $j = \Phi[i]$
   6. while $T[i + \ell] = T[j + \ell]$ do
      7. $\ell = \ell + 1$
      8. $\Phi[i] = \ell$
      9. $\ell = \max\{0, \ell - 1\}$
6. for $i = 1, \ldots, n$ do $LCP[i] = \Phi[SA[i]]$
7. return $LCP$

- compute $LCP$-array in text order
- reorder $LCP$-array as final step

<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$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>12</td>
<td>11</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>$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>

- example:  
- correctness and running time similar

PINGO why better?
Brief Remainder: Cache & Cache Misses

- cache is small but fast memory
- located on CPU
- cache miss is failure to retrieve data from cache
- instead data has to be loaded from main memory

PINGO how much slower is a main memory compared to L1 cache?

Latency Numbers
- L1 cache reference ≈ 1 ns
- L2 cache reference ≈ 4 ns
- main memory reference ≈ 100 ns

Cache Sizes (AMD Ryzen 7 PRO 4750U)
- L1: 256 KiB (8 instances)
- L2: 4 MiB (8 instances)
- L3: 8 MiB (2 instances)
Better Due to Less Cache Misses

**Function** LinearTimeLCP\((T, SA[1..n])\):

1. for \(i = 1, \ldots, n\) do \(ISA[SA[i]] = i\)
2. \(\ell = 0, LCP[1] = 0\)
3. for \(i = 1, \ldots, n\) do
4.  if \(ISA[i] \neq 1\) then
5.      \(j = SA[ISA[i] - 1]\)
6.      while \(T[i + \ell] = T[j + \ell]\) do
7.         \(\ell = \ell + 1\)
8.      \(LCP[ISA[i]] = \ell\)
9.      \(\ell = \max\{0, \ell - 1\}\)
10. return \(LCP\)

**Function** \(\Phi\)-Algorithm\((T, SA[1..n])\):

1. \(\Phi[n] = SA[n] \quad SA[1] = n; T\) has sentinel
2. for \(i = 2, \ldots, n\) do \(\Phi[SA[i]] = SA[i - 1]\)
3. \(\ell = 0\)
4. for \(i = 1, \ldots, n\) do
5.    \(j = \Phi[i]\)
6.    while \(T[i + \ell] = T[j + \ell]\) do
7.       \(\ell = \ell + 1\)
8.    \(\Phi[i] = \ell\)
9.    \(\ell = \max\{0, \ell - 1\}\)
10. for \(i = 1, \ldots, n\) do \(LCP[i] = \Phi[SA[i]]\)
11. return \(LCP\)
<table>
<thead>
<tr>
<th>Pizza &amp; Chili Corpus</th>
<th>Experimental Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://pizzachili.dcc.uchile.cl/">http://pizzachili.dcc.uchile.cl/</a></td>
<td>used text described above</td>
</tr>
<tr>
<td>de facto standard text corpus</td>
<td>on T14s with AMD Ryzen 7 PRO 4750U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Used in Experiment (50 MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp XML-Data providing bibliographic information</td>
</tr>
<tr>
<td>DNA DNA reads from the Gutenberg Project</td>
</tr>
<tr>
<td>english English texts of the Gutenberg Project</td>
</tr>
<tr>
<td>sources Source code from the Linux kernel</td>
</tr>
</tbody>
</table>
## Practical Comparison of Both Algorithms (2/2)

<table>
<thead>
<tr>
<th>Text</th>
<th>Naive (ms)</th>
<th>[Kas+01] (ms)</th>
<th>[KMP09] (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>9121.6</td>
<td>3479.0</td>
<td>2567.2</td>
</tr>
<tr>
<td>DNA</td>
<td>6763.0</td>
<td>6152.2</td>
<td>4174.6</td>
</tr>
<tr>
<td>english</td>
<td>99811.4</td>
<td>4899.8</td>
<td>3316.2</td>
</tr>
<tr>
<td>sources</td>
<td>12687.6</td>
<td>3486.4</td>
<td>2536.6</td>
</tr>
</tbody>
</table>
Permutated LCP-Array [KMP09]

Definition: PLCP-Array

- \( PLCP[SA[i]] = LCP[i] \)
- \( PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i]) \)

- \( PLCP[i] \geq PLCP[i - 1] - 1 \)
- \( T[i - 1] = T[\Phi[i] - 1] \Rightarrow PLCP[i] \) is reducible
- \( PLCP[i] \) is reducible \( \Rightarrow PLCP[i] = PLCP[i - 1] - 1 \)

- only compute irreducible PLCP-values
- sum of all irreducible PLCP-values is \( \leq n \lg n \)
Recap: Pattern Matching with the Suffix Array

Function SeachSA(T, SA[1..n], P[1..m]):
1 \( \ell = 1, r = n + 1 \)
2 while \( \ell < r \) do
3 \( i = \left\lfloor \frac{\ell + r}{2} \right\rfloor \)
4 if \( P > T[SA[i]..SA[i] + m] \) then
5 \( \ell = i + 1 \)
6 else \( r = i \)
7 \( s = \ell, \ell = \ell - 1, r = n \)
8 while \( \ell < r \) do
9 \( i = \left\lceil \frac{\ell + r}{2} \right\rceil \)
10 if \( P = T[SA[i]..SA[i] + m] \) then \( \ell = i \)
11 else \( r = i - 1 \)
12 return \([s, r]\)

Lemma: Running Time SeachSA

The SeachSA answers counting queries in \( O(m \lg n) \) time and reporting queries in \( O(m \lg n + \text{occ}) \) time

Proof (Sketch)
- two binary searches on the SA in \( O(\lg n) \) time
- each comparison requires \( O(m) \) time
- counting in \( O(1) \) additional time
- reporting in \( O(\text{occ}) \) additional time
- comparison of pattern is expensive
Speeding Up Pattern Matching with the LCP-Array (1/4)

- 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 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
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 \( \xi = lcp(SA[\ell], SA[i]) \) \( \Theta O(1) \) time with RMQs
Speeding Up Pattern Matching with the LCP-Array (3/4)

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

<table>
<thead>
<tr>
<th>$\xi &gt; \lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P[\lambda + 1] &gt; T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$</td>
</tr>
<tr>
<td>$\ell = i$ without character comparison</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\xi = \lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare as before</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\xi &lt; \lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\xi \geq \rho$ and $P[\xi + 1] &lt; T[SA[i] + \xi]$</td>
</tr>
<tr>
<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>$\lambda$</td>
<td>$\neq$</td>
</tr>
<tr>
<td>$P[\lambda]$</td>
<td>$T[\cdot]$</td>
<td>$T[\cdot]$</td>
</tr>
<tr>
<td>$\xi$</td>
<td>$P[\rho]$</td>
<td>$\rho$</td>
</tr>
</tbody>
</table>
Lemma:
Using RMQs, SearchSA answers counting queries in $O(m + \lg n)$ time and reporting queries in $O(m + \lg n + \text{occ})$ time.

Proof (Sketch)
- either halve the range in the suffix array ($\xi \neq \lambda$)
  or
- compare characters of the pattern (at most $m$)
Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use SA and LCP
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

<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>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>$$$</td>
</tr>
<tr>
<td>SA</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>LCP</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusion and Outlook

This Lecture
- linear time LCP-array construction
- suffix tree construction based on SA and LCP
- engineered LCP-Array construction algorithms
- cache misses are costly
- interesting properties of the PLCP-array

Linear Time Construction

Next Lecture
- text compression using SA and LCP
Bibliography I


