Text Indexing

Lecture 09: Suffix Array Construction in Distributed and External Memory

Florian Kurpicz
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

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

### Example

<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>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>
Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- darker grey: linear running time
- brown: available implementation

Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible
- until 2021: DivSufSort fastest in practice with $O(n \lg n)$ running time
- since 2021: libSAIS fastest in practice with $O(n)$ running time
External and Distributed Memory

External Memory
- internal memory of size $M$ words
- external memory of unlimited size
- transfer of blocks of size $B$ words

scanning $N$ elements: $\Theta\left(\frac{N}{B}\right)$

sorting $N$ elements: $\Theta\left(\frac{N}{B} \lg \frac{M}{B} \frac{N}{B}\right)$

semi-external memory

Distributed Memory
- $p$ PEs with internal memory
- communication between PEs over network

bulk-synchronous parallel model [Val90]
- supersteps: local work, communication, synchronization
Challenges for Suffix Array Construction

Distributed Memory
- suffixes span over whole input  
- comparing suffixes requires text access
- random access

External Memory
- random access expensive in both models
- whole suffix not available locally in distributed memory

express suffix array construction algorithm using
- scanning
- sorting
- merging
2003
[Prefix-Doubling]

2012
[Induced-Copying]

2014
[Recursion]

2015
[AKA]
cloudSACA

2018
[FA]
PSAC

2018
[BGK]
Doubling

2019
[FK]
Doubling

Distributed Memory

PE 1
Speicher

PE 2
Speicher

PE 3
Speicher

... PE p
Speicher

[KSB]
DC3

[FK]
DC3/7/13

[BGK]
DC3/7/13

6/25
2023-01-09 Florian Kurpicz | Text Indexing | 09 Suffix Array in Distributed and External Memory

Institute for Theoretical Informatics, Algorithm Engineering
**Definition: h-Order**

- **h-Order**: 
  \[ T[i..n] \leq_h T[j..n] \iff T[i..i+h] \leq T[j..j+h] \]
- **SA_h** is the suffix array of all suffixes ordered by \( h \)-order not unambiguously

**Definition: h-Ranks und h-Groups**

- All suffixes that are equal w.r.t. an \( h \)-order are in an **h-group**
- **h-rank**: number of lexicographically smaller \( h \)-groups plus one
Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters
- 3-rank can be computed from first 3 characters
- 4-rank can be computed from first 4 characters
- 4-rank can be computed from two 2-ranks

- Compute $2^{k+1}$-ranks using $2^k$-ranks
Prefix-Doubling: Example

1. initial rank is \( T[i] \) 1-rank
2. for \( k = 0 \) to \( \lceil \log n \rceil \)
3. new \( 2^{k+1} \)-ranks based on
   \[ \text{ISA}_{2^k}[i] \& \text{ISA}_{2^k}[i + 2^k] \]
4. if all ranks are unique, break
5. compute SA from ISA

Simple Algorithm

- N. Jesper Larsson and Kunihiko Sadakane.
Prefix-Doubling: Practical Approaches

Use $ISA_h$ [FA15]
- use $ISA_{2^k}$ to compute rank tuples
- for position $i$ use rank $ISA_{2^k}[i + 2^k]$
- if $i + 2^k > n$, second rank is 0
- example on the board

Sort by Text Positions [Dem+08; FK19]
- especially good if access to $ISA_h$ is expensive
- sort tuples (Textposition $i$, Rang $r$)
- using $(i, r) \leq (j, r')$ iff
  \[(i \mod 2^k, \lfloor i / 2^k \rfloor) < (j \mod 2^k, \lfloor j / 2^k \rfloor)\]
- example on the board
Prefix-Doubling: Running Time

- running time: $O(n \lg n)$
- memory requirements: $8n(+n)$ words for texts $\leq 4$ GiB
- worst-case input: $T = a^{n-1}$

Generalization

- more than doubling is possible
- compute $\alpha^{k+1}$-ranks using $\alpha$ $\alpha^k$-ranks
- can save I/Os in EM $\alpha = 4$ requires 30% less I/Os than $\alpha = 2$ [Dem+08]
Prefix Doubling: Experimental Results [Kur20]

Commoncrawl throughput (MiB/s)

- 512 MiB per PE
- 1024 MiB per PE
- 1536 MiB per PE

Commoncrawl construction space (B/n)

- PEs (20 threads)

Data points for different methods:
- pDivSufSort
- pPreDoubling
- psac
Recap: SAIS

**The Idea: Inducing**

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$T[i..n] < T[j..n] \iff T[i + 1..n] < T[j + 1..n]$

**Suffix Array Construction in 3 Phases**

- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

- classification helps identifying special suffixes
- everything in linear time

**The Algorithm: SAIS**

- using inducing for everything
- described in [NZC11]
SAIS in External Memory [BFO16; Kär+17]

**Classification**
- simple scan of the text
- works well in external memory

**Sort Special Substrings**
- recursion
- works well in external memory if rest works well

**Inducing**
- keep buffer for each $\alpha$-interval of suffix array
- scan text and induce characters by writing them in buffer

- separate text during classification
- blockwise preinducing
- heavily relies on external memory priority queue
Jack of all Trades: DC3

- first direct linear time suffix array construction algorithm: DC3
- suffix tree construction algorithm with similar idea [Far97]
- based on Difference Cover
Definition: Difference Cover

The set $D \subseteq [0, \nu)$ is a difference cover modulo $\nu$, if

$\{(i - j) \mod \nu : i, j \in D\} = [0, \nu)$

- $\{0, 1\}$ is difference cover modulo 3
- $\{0, 1, 3\}$ is difference cover modulo 7
- $\{0, 1, 3, 9\}$ is difference cover modulo 13

0 $\equiv$ 0 $-$ 0 (mod 3)
1 $\equiv$ 1 $-$ 0 (mod 3)
2 $\equiv$ 0 $-$ 1 (mod 3)

0 $\equiv$ 0 $-$ 0 (mod 7)
1 $\equiv$ 1 $-$ 0 (mod 7)
2 $\equiv$ 3 $-$ 1 (mod 7)
3 $\equiv$ 3 $-$ 0 (mod 7)
4 $\equiv$ 0 $-$ 3 (mod 7)
5 $\equiv$ 1 $-$ 3 (mod 7)
6 $\equiv$ 0 $-$ 1 (mod 7)
1. Sample Suffixes

- For $i \in \{0, 1, 2\}$ let be

  $B_i = \{i \in [0, n) : i \mod 3 = k\}$

- $C = B_0 \cdot B_1$

  $\{0, 1\}$ is difference cover modulo 3

- $C = \{0, 3, 6, 9, 1, 4, 7, 10\}$
2. Sort Sampled Suffixes

- for $k = 0, 1$ let be

$$R_k = [T[k] T[k + 1] T[k + 2]] [T[k + 3] T[k + 4] T[k + 5]] \ldots [T[\text{max } B_k] T[\text{max } B_k + 1] T[\text{max } B_k + 2]]$$

- $R = R_0 \cdot R_1$

- sort $R$ with Radix Sort in $O(n)$ time

- all characters unique: ranks of sampled suffixes are known

- otherwise: recursively execute algorithm on $R$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>mis</td>
<td>sis</td>
<td>sip</td>
<td>p</td>
<td></td>
<td>iss</td>
<td>iss</td>
<td>ipp</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

\[ C = \{ 0, 3, 6, 1, 4, 7 \} \]

Recursion: Step 2

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
3 & 6 & 5 & 4 & 2 & 2 & 1 & 0 \\
\end{array}
\]
3. Sort Non-Sampled Suffixes

- let \( i, j \in B_2 \), then
  \[
  S_i \leq S_j \iff (T[i], \text{Rang}(S_{i+1})) \leq (T[j], \text{Rang}(S_{j+1}))
  \]

- ranks of next two suffixes is known
- sort tuples (in \( B_2 \)) using Radix Sort
- \( O(n) \) time
4. Merge Suffixes

Let \( i \in C \) and \( j \in B_2 \), then

- If \( i \in B_0 \), then
  \[ S_i \leq S_j \iff (T[i], \text{Rang}(S_{i+1})) \leq (T[j], \text{Rang}(S_{j+1})) \]

- If \( i \in B_1 \), then
  \[ S_i \leq S_j \iff (T[i], T[i+1], \text{Rang}(S_{i+2})) \leq (T[j], T[j+1], \text{Rang}(S_{j+2})) \]

Ranks: 4 7 6 5 3 2 1 0
## Finish Recursion

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>[mis]</td>
<td>[sis]</td>
<td>[sip]</td>
<td>[pi$]</td>
<td>[iss]</td>
<td>[iss]</td>
<td>[ipp]</td>
<td>[i$$]</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

## Suffix Array Construction with DC3 (6/6)

- **Finish Recursion**

- **ranks**

- **rest can be used as exercise**

  solution: 11 10 7 4 1 0 9 8 6 3 5 2
DC3: Running Times

- everything but recursion obviously in $O(n)$ time
- only sorting tuples of size $\leq 3$
- Radix Sort in $O(n)$ time

- recursion on texts of size $[2n/3]$;
- $T(n) = T(2n/3) + O(n) = O(n)$

Generalization

- works with every difference cover
- sorting somewhat more complicated
- running time: $O(\nu n)$

In Other Models of Computation

- external memory: $O\left(\frac{n}{DB} \lg \frac{M}{B} \frac{n}{B}\right)$ using $D$ disks
- BSP: $O\left(\frac{n \lg n}{P} + L \lg^2 P + g \frac{n \lg n}{P \lg(n/P)}\right)$ using $P$ PEs
- EREW-PRAM: $O(\lg^2 n)$ time and $O(n \lg n)$ work
Prefix Doubling: Experimental Results [Kur20]

Commoncrawl throughput (MiB/s)

- 512 MiB per PE
- 1024 MiB per PE
- 1536 MiB per PE

Commoncrawl construction space (B/n)

- PEs (20 threads)

- pDivSufSort
- pPreDoubling
- psac
- pDC3
- pDC7
- pDC13

PEs (20 threads)
Conclusion and Outlook

This Lecture
- distributed and external memory suffix sorting
- more suffix sorting techniques

Next Lecture
- inverted indices

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- r-Index


Bibliography II


Bibliography III(77,208),(923,864)


Bibliography IV