Optimal In-Place Suffix Sorting

Zhize Li, Jian Li, Hongwei Huo

IIIIS, Tsinghua University

https://zhizeli.github.io/

October 10th, SPIRE 2018
• Problem Definition
• Related Work
• Our Results
• Our Algorithm
• Conclusion
Problem Definition

Suffix array is a fundamental data structure introduced by Manber and Myers as a *space-saving* alternative to suffix trees in SODA’90.

- **Definition:**
  - Given a string $T[0..n-1]$, each $T[i]$ belongs to an integer alphabet $\Sigma$
  - *Suffix:* $suf(i)$ is a substring $T[i…n-1]$ (from index $i$ to the end of $T$)
  - *Suffix array* $SA$ contains the indices of all sorted suffixes

- **Example:**
  - If $T=${"130"} (integer alphabet), then all suffixes are {130, 30, 0}
  - $suf(2)<suf(0)<suf(1)$, i.e. $0<130<30$ (in *lexicographical order*)
  - $SA=[2,0,1]$ $suf(SA[i]) < suf(SA[j])$ for all $i<j$

- **Problem:**
  
  Construct the suffix array $SA$ for a given string $T[0…n-1]$
• Problem Definition
• Related Work
• Our Results
• Our Algorithm
• Conclusion
Related Work

- Manber and Myers [MM90] constructed the SA using a doubling technique.
  - beginning characters -> first two characters-> first four characters….  
    Time: $O(n \log n)$  
    Space: $O(n)$ workspace

**Workspace** denotes the **total space** used by an algorithm except for the input string $T$ and the suffix array $SA$. 
Related Work

• Manber and Myers [MM90] constructed the SA using a doubling technique.
  • beginning characters -> first two characters-> first four characters….  
    Time: $O(n \log n)$   Space: $O(n)$ workspace

• In 2003, the first **linear time** algorithms were obtained by [KSPP03,KS03,KA03] using the divide-and-conquer technique.
  Time: $T(n) = T(cn) + O(n) = O(n)$   Space: $O(n)$ workspace
Related Work

• Manber and Myers [MM90] constructed the SA using a doubling technique.
  • beginning characters -> first two characters -> first four characters….
    Time: $O(n\log n)$  Space: $O(n)$ workspace

• In 2003, the first linear time algorithms were obtained by [KSPP03, KS03, KA03] using the divide-and-conquer technique.
  Time: $T(n)=T(cn)+O(n)=O(n)$  Space: $O(n)$ workspace

• Bottleneck: space
Related Work

• Open problem [Franceschini and Muthukrishnan, ICALP’07]:
  • Design in-place (O(1) workspace) and o(nlogn) time algorithms for integer alphabets \( \Sigma \) with \(|\Sigma| \leq n\).

• Ultimate goal: design in-place algorithms, and maintain the optimal \( O(n) \) time complexity.

• Previous best result [Nong, TOIS’13]:
  Time: \( O(n) \)    Space: \(|\Sigma| + O(1)\) workspace

Theorem. Our optimal in-place algorithm takes \( O(n) \) time to compute the suffix array even if the string \( T \) is read-only and \(|\Sigma| = O(n)\).
• Problem Definition
• Related Work
• **Our Results**
• Our Algorithm
• Conclusion
Our Results

Table 1: Time and workspace of suffix sorting algorithms for integer alphabets $\Sigma$

<table>
<thead>
<tr>
<th>Time</th>
<th>Workspace (words)</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>[SS07]</td>
</tr>
<tr>
<td>$O(n \log^2 n)$</td>
<td>$O(n)$</td>
<td>[Sad98]</td>
</tr>
<tr>
<td>$O(n \sqrt{</td>
<td>\Sigma</td>
<td>\log(n/</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>$O(n)$</td>
<td>[MM90, LS07]</td>
</tr>
<tr>
<td>$O(n \log \log n)$</td>
<td>$O(n)$</td>
<td>[KJP04]</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>[KSPP03, KS03, KA03]</td>
</tr>
<tr>
<td>$O(n \log \log</td>
<td>\Sigma</td>
<td>)$</td>
</tr>
<tr>
<td>$O(vn)$</td>
<td>$O(n/\sqrt{v})$ $v \in [1, \sqrt{n}]$</td>
<td>[KSB06]</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$n + n/ \log n + O(1)$</td>
<td>[NZC09]</td>
</tr>
<tr>
<td>$O(n^2 \log n)$</td>
<td>$cn + O(1)$ $c &lt; 1$</td>
<td>[MF02, MP06]</td>
</tr>
<tr>
<td>$O(n^2 \log n)$</td>
<td>$</td>
<td>\Sigma</td>
</tr>
<tr>
<td>$O(n \log</td>
<td>\Sigma</td>
<td>)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$</td>
<td>\Sigma</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>This paper</td>
</tr>
</tbody>
</table>
• Problem Definition
• Related Work
• Our Results
• **Our Algorithm**
• Conclusion
Our Algorithm

• Notations:
  • A suf(i) (T[i..n-1]) is **L-suffix** if suf(i)>suf(i+1)
  • Type of character T[i] is the same as suf(i)
  • **LMS-suffix** (leftmost S-suffix) if suf(i) is S-type and suf(i-1) is L-type

• Example: T[0..7]=“31221120”

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
Our Algorithm

**Framework:**

1. Sort all LMS-characters of $T$ (counting sort)
2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LMS-substrings are \{1221, 1120, 0\}.

<table>
<thead>
<tr>
<th>SA</th>
<th>Index(LMS)</th>
<th>$E$ $E$ $E$ $E$ $E$ 7 1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>LMS-substring</td>
<td>$E$ $E$ $E$ $E$ $E$ 7 4 1</td>
</tr>
<tr>
<td>3.</td>
<td>$T_1$ LMS-substring</td>
<td>2 1 0 $E$ $E$ 7 4 1</td>
</tr>
</tbody>
</table>
Our Algorithm

• **Framework:**
  1. Sort all LMS-characters of $T$ (counting sort)
  2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
  3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LMS-substrings are $\{1221, 1120, 0\}$.

3. $\begin{array}{|c|c|}
\hline
T_1 & \text{LMS-substring} \\
\hline
2 & 1 & 0 & E & E & 7 & 4 & 1 \\
\hline
\end{array}$

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>4</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS-Substring</td>
<td>1221</td>
<td>1120</td>
<td>0</td>
</tr>
<tr>
<td>Rank ($T_1$)</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$T_1$ ("210") shortens $T$("12211120") by using **one character** to **replace a substring** of $T$.
Our Algorithm

**Framework:**
1. Sort all LMS-characters of $T$ (counting sort)
2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>$T$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LMS-substrings are $\{1221, 1120, 0\}$.
Our Algorithm

• **Framework:**
  1. Sort all LMS-characters of T (counting sort)
  2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
  3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)
  4. Sort LMS-suffixes of T by solving $T_1$ recursively (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LMS-substrings are \{1221, 1120, 0\}.
Our Algorithm

• **Framework:**
  1. Sort all LMS-characters of T (counting sort)
  2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
  3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)
  4. Sort LMS-suffixes of T by solving $T_1$ recursively (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>L</th>
<th>S</th>
<th>L</th>
<th>L</th>
<th>S</th>
<th>S</th>
<th>L</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LMS-substrings are $\{1221, 1120, 0\}$.

LMS-Suffixes are $\{1221120, 1120, 0\}$.

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>4</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS-Substring</td>
<td>1221</td>
<td>1120</td>
<td>0</td>
</tr>
<tr>
<td>Rank ($T_1$)</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$T_1$ ("210") shortens $T$ ("12211120") by using one character to replace a substring of T

A suffix of $T_1$ corresponds to an LMS-suffix of T
Our Algorithm

• **Framework:**
  • 1. Sort all LMS-characters of $T$ (counting sort)
  • 2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
  • 3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)
  • 4. Sort LMS-suffixes of $T$ by solving $T_1$ recursively (simple)

<table>
<thead>
<tr>
<th>Index</th>
<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>$T$</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>$L$</td>
<td>$S$</td>
<td>$L$</td>
<td>$L$</td>
<td>$S$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S$</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

LMS-substrings are $\{1221, 1120, 0\}$. 4. $T_1$ LMS-substring
LMS-Suffixes are $\{1221120, 1120, 0\}$.
Our Algorithm

• **Framework:**
  1. Sort all LMS-characters of T (counting sort)
  2. Induced sort all LMS-substrings from sorted LMS-characters (same as 5)
  3. Construct the reduced subproblem $T_1$ from sorted LMS-substrings (simple)
  4. Sort LMS-suffixes of T by solving $T_1$ recursively (simple)
  5. **Induced sort all suffixes from the sorted LMS-suffixes** (technical part)

<table>
<thead>
<tr>
<th>Index</th>
<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>T</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
<tr>
<td>LMS</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

LMS-substrings are \{1221, 1120, 0\}. LMS-Suffixes are \{1221120, 1120, 0\}.

---

Index (LMS) | $E$ | $E$ | $E$ | $E$ | $E$ | $7$ | $1$ | $4$

LMS-substring | $E$ | $E$ | $E$ | $E$ | $E$ | $7$ | $4$ | $1$

$T_1$ LMS-substring | $2$ | $1$ | $0$ | $E$ | $E$ | $7$ | $4$ | $1$

induced sort

sorted LMS | sorted SA

Zhize Li (Tsinghua)  
Optimal In-Place Suffix Sorting  
19/26
Our Algorithm

• Induced sorting all suffixes from the sorted LMS-suffixes
  • 1. First induced sort all L-suffixes from the sorted LMS-suffixes
  • Divide into two stages

\[ c_p = cn / \log n \]

Stage 1: Construct pointer data structure \( P \) & combine interior counter trick to induced sort the first \( n_L - c_p \) L-suffixes.
Our Algorithm

• Induced sorting all suffixes from the sorted LMS-suffixes
  • 1. First induced sort **all L-suffixes** from the sorted LMS-suffixes
    • Divide into two stages

Stage 1: Construct pointer data structure \( P \) & combine interior counter trick to induced sort the first \( n_L - c_p \) L-suffixes.

Stage 2: Use **binary search** to extend the interior counter trick to induced sort the last \( c_p \) L-suffixes **without** \( P \).

**Key:** without \( P \), Stage 2 also maintains **linear time** since \( c_p \) is small enough (i.e., \( c_p \log n = cn \)).
Our Algorithm

• Induced sorting all suffixes from the sorted LMS-suffixes
  • 1. First induced sort all L-suffixes from the sorted LMS-suffixes
     • Divide into two stages
  • 2. Then induced sort all S-suffixes from the sorted L-suffixes (same as 1)
Our Algorithm

- **Induced sorting all suffixes from the sorted LMS-suffixes**
  - 1. First induced sort all L-suffixes from the sorted LMS-suffixes
  - Divide into two stages
  - 2. Then induced sort all S-suffixes from the sorted L-suffixes (same as 1)
  - 3. **Merge the sorted L- and S-suffixes** to get the final suffix array

![](image.png)
• Problem Definition
• Related Work
• Our Results
• Our Algorithm
• Conclusion
Conclusion

• We propose the *first* in-place suffix sorting algorithm which is *optimal both in time and space*.
  
  Time: $O(n)$  
  Space: $O(1)$ workspace (in-place)

• Our algorithm solves the open problem posed by Franceschini and Muthukrishnan in ICALP 2007.
  
  • Desired time and space in their open problem:
    
    Time: $o(n\log n)$  
    Space: $O(1)$ workspace (in-place)
Thanks!

Zhize Li (IIIS, Tsinghua University)
https://zhizeli.github.io/