Advanced Data Structures

Lecture 08: Compressed Suffix Array

Florian Kurpicz
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

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

<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></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>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
(Compressed) Text Indices #Ad

- **Suffix Tree**
  - 1973

- **Suffix Array**
  - 1993

- **BWT**
  - 1994

- **Wavelet Tree**
  - 2000

- **FM-Index**
  - 2000

- **LCP Array**
  - 1993

- **r-Index**
  - 2018

- **Block Tree**
  - 2021

Memory Requirements

- String Sorting
- LCE Queries
- (Patricia) Tries

Succinct Data Structures

Bit Vectors and Rank/Select Queries

EM Hashing

Submitted (arXiv:2205.04745)
Ψ Function

Definition: Ψ Function
Given a suffix array SA of length n,

Ψ(i) = SA⁻¹[SA[i] + 1]

- SA[Ψ(i)] = SA[i] + 1
- where in SA is the suffix T[SA[i] + 1..n)
- “successor” function

- can be used to obtain suffix array
- can be compressed \( O(n \log n) \) bits
Replacing $SA$ with $\Psi$

- which number does in this example not occur? Answer: 3
- how to obtain $SA[i]$ using $\Psi$ PINGO
- follow positions until last suffix is found
- last suffix is at position 1
- $n - \#\text{steps}$ is $SA$ value
- requires $O(n)$ time
- pattern matching: $O(mn \log n)$ time
- pattern matching with $LCP$ and $RMQ$: $O(mn + \log n)$ time

<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>$$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$$$</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>$$$</td>
</tr>
</tbody>
</table>
Speeding Up Lookups in $\Psi$ (1/2)

- space $SA$: $O(n \log n)$ bits
- space text: $O(n \log \sigma)$ bits
- space compressed suffix array should not more than text

- sample every $\log n$-th $SA$ entry
- $O(n/ \log n)$ samples of size $O(\log n)$ bits
- total space: $O(n)$ bits

- every $\log n$-th entry in $\Psi$
- every $\log n$-th step in $\Psi$
- what is better? PINGO
Speeding Up Lookups in $\Psi$ (2/2)

- every log $n$-th entry in $\Psi$
- every log $n$-th step in $\Psi$
- what is better? 🤔

PINGO

- every log $n$-th step in $\Psi$ is better
- sampled positions may not be reached in better asymptotic time

how much time does recovering SA position from $\Psi$ require with sampling? 🤔

PINGO

answer: $O(\log n)$
Structure of $\Psi$

- does $\Psi$ have some structure?

**Lemma: Structure of $\Psi$**

$$T[SA[i]] = T[SA[i + 1]] \Rightarrow \Psi(i) < \Psi(i + 1)$$

**Proof (Sketch)**

- $T[SA[i]] \leq T[SA[i + 1]]$
- If $T[SA[i]] = T[SA[i + 1]]$ then $T[SA[i] + 1..n) \leq T[SA[i + 1] + 1..n)$
- $T[SA[i] + 1] = T[\Psi(i)]$
- If suffixes share same character, lexicographical order of suffixes without first character holds

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

- Note that not all increasing intervals belong to the same character
- Example on the board 🎨
Compressing Ordered Sequences

**Δ-Encoding**
- store difference between entries
- scanning whole sequence up to value when decoding

**Elias-Fano (Lecture 05)**
- upper and lower halves
- upper half represented in bit vector $p_i + i$
- lower half plain bit compressed

using Elias-Fano is bad for large alphabets
example on the board

<table>
<thead>
<tr>
<th>Value</th>
<th>Delta Encoding</th>
<th>Upper Bit Vector</th>
<th>Lower Bit Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000</td>
<td>111011010</td>
<td>00 01 10 00 11 10 00 01 10 00</td>
</tr>
<tr>
<td>1</td>
<td>000001</td>
<td>010100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>000010</td>
<td>010101</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>000100</td>
<td>010110</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>000111</td>
<td>100000</td>
<td></td>
</tr>
</tbody>
</table>

**upper:** 11101101000111000100
**lower:** 00 01 10 00 11 10 00 01 10 00
Recap: Elias-Fano Coding Space

Lemma: Elias-Fano Coding

Given an array containing \( n \) distinct integers from a universe \( \mathcal{U} = [0, n) \), the array can be represented using

\[
 n(2 + \log \left\lceil \frac{u}{n} \right\rceil) \text{ bits}
\]

while allowing \( O(1) \) access time and \( O(\log \frac{u}{n}) \) predecessor/successor time.
Compressing Sparse Ordered Sequences

- Elias-Fano coding for each increasing interval
  - $\sigma$ many
  - only every $1/\sigma$-th entry is set (sparse)

- if there are $n$ entries of universe with size $u$
  - make entries sparse using $q = u/n$
  - for each value $x$ store pair $(x/q, x \% q)$

- $u = 512$, $n = 8$, $q = 64$
- $(0, 3, 17, 89, 128, 132, 500, 511)$
- $\{0, 0\}, \{0, 3\}, \{0, 7\}, \{1, 25\}, \{2, 0\}, \{2, 4\}, \{7, 52\}, \{7, 63\}$

- store quotient $(x/q)$ using Elias-Fano
- store remainder $(x \% q)$ plain using $\lceil \log q \rceil$ bits

**Lemma: $\Psi$ with Elias-Fano**

Using Elias-Fano with quotienting, $\Psi$ can be stored using $O(n\sigma)$ bits

- more precise: two additional bits per character
Simple Compressed Suffix Array

- compute $\Psi$ and store samples of SA
- compress $\Psi$ Elias-Fano with quotienting
- binary search on SA by decoding $\Psi$

- space: $O(n \log \sigma)$ space
- query time: $O(m \log^2 n)$
Improving Compressed Suffix Arrays [GV05] (1/5)

- improve SA lookup to $O(\log \log n)$ time
- divide-and-conquer approach
- storing $\Psi$ only for half of the entries
- recurs for the other half

### Table

<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>$</td>
</tr>
<tr>
<td>SA</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NEW</td>
<td>13</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td>4</td>
</tr>
</tbody>
</table>

- for which values do we store $\Psi$? [PINGO]
Improving Compressed Suffix Arrays (2/5)

- store bit vector marking odd SA values
- store only odd SA values
- store $\Psi$ for even SA values

- store $\Psi$ as before
- Elias-Fano with quotienting
- without sampling

- right half (SA) still big
- how to recurs?

<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>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NEW</td>
<td>13</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>$BV$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Improving Compressed Suffix Arrays (3/5)

- SA half consists only of odd values
- for value $x$ store $(x - 1)/2$
- reversible since all values are odd

- 13, 1, 9, 3, 11, 7, 5
- 6, 0, 4, 1, 5, 3, 2

- what do we have here? PINGO
- permutation basically a suffix array without text

- recurs on the permutation without explicitly storing it

<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$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NEW</td>
<td>13</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>$BV$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Improving Compressed Suffix Arrays (4/5)

- recurs \( \log \log n \) times
- guarantees \( O(\log \log n) \) time to obtain \( SA \) value
- allows to store final \( SA \) within space bounds

Lemma: Space Final \( SA \)

Using the divide-and-conquer approach, the final \( SA \) requires \( O(n) \) bits of space

Proof (Sketch)

- after \( \log \log n \) recursions \( SA \) has size \( \frac{n}{2^{\log \log n}} \)
- each entry requires \( \log n \) bits
- total space: \( O(n) \) bits
Lemma: Decoding Time Improved CSA

An SA value can be decoded in $O(\log \log n)$ time using the improved CSA.

Proof (Sketch)

- on each level, odd SA values can be decoded using the recursive SA
- there are at most $\log \log n$ levels
- on each level, even SA values can be decoded in one step, as the next SA value is odd
- requires rank and select data structures

<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>a</td>
<td>b</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>$\Psi$</td>
<td>-</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NEW</td>
<td>13</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>$BV$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion and Outlook

This Lecture
- compressed suffix array
- note that CSA can be compressed further
- Elias-Fano for sparse sequences

Advanced Data Structures

- Successor
- RMQ
  - static/dynamic
  - BV
  - succ. trees
  - range min-max tree
  - succ. graphs
Bibliography I

