Advanced Data Structures

Lecture 08: Compressed Suffix Array

Florian Kurpicz
Recap: Suffix Array

<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>
</tbody>
</table>

- space: $O(n \log n)$ bits
- space text: $n \lceil \log \sigma \rceil$ bits
- better: index requiring same space as text
- even better: index requiring same space as compressed text

Recap: Suffix Array
(Compressed) Text Indices #Ad

- **Suffix Tree**
  - 1973
  - Memory Requirements

- **Suffix Array**
  - 1993
  - ALENEX '19, BigData '18

- **LCP Array**
  - 1993
  - ALENEX '19, BigData '18

- **BWT**
  - 1994

- **Wavelet Tree**
  - 2000
  - JEA '21, ALENEX '18,'20, SPIRE '19

- **FM-Index**
  - 2000

- **r-Index**
  - 2018
  - ALENEX '19, BigData '18

- **Block Tree**
  - 2021

- **String Sorting**
- **LCE Queries**
- **(Patricia) Tries**
- **Succinct Data Structures**
- **Bit Vectors and Rank/Select Queries**
- **EM Hashing**

- **Compression**
  - abccaaca
  - 0110001
  - 11011
  - 001
  - 010
  - 110
  - 0
  - a: 0
  - b: 4
  - c: 5

- **(Compressed) Text Indices #Ad**
**ψ Function**

**Definition: ψ Function**

Given a suffix array $SA$ of length $n$,

$$\psi(i) = SA^{-1}[SA[i] + 1]$$

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

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

- Which number does in this example not occur? Answer: 3
- How to obtain $SA[i]$ using $\Psi$?

Follow positions until last suffix is found
- Last suffix is at position 1
- $n - \#steps$ 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
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: $O(\log n)$

<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>$\Psi$</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>
</tr>
<tr>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>$$$</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>$$$</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
</tr>
</tbody>
</table>
Structure of $\Psi$

- does $\Psi$ have some structure?

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

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

**Proof (Sketch)**

- $T[SA[i]] \leq T[SA[i + 1]]$
- if $T[SA[i]] = T[SA[i + 1]]$ then $T[SA[i] + 1..n] \leq T[SA[i + 1] + 1..n]$
- $T[SA[i] + 1] = T[\Psi(i)]$

- note that not all increasing intervals belong to the same character
- example on the board

<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>
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></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>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

$$n(2 + \log \lceil \frac{u}{n} \rceil)$$

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: $O(m \log^2 n)$
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>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{T}$ a b a b c a b c a b b a $</td>
</tr>
<tr>
<td>$\text{SA}$ 13 12 1 9 6 3 11 2 10 7 4 8 5</td>
</tr>
<tr>
<td>$\Psi$ - 1 8 9 10 11 2 6 7 12 13 4 5</td>
</tr>
<tr>
<td>NEW 13 1 9 3 11 7 5 1 10 6 7 13 4</td>
</tr>
</tbody>
</table>

for which values do we store $\Psi$?
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

<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>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 \( n/2^{\log \log n} \)
- 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.
Conclusion and Outlook

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

Next Lecture
- temporal data structures

Advanced Data Structures

- String B-tree
- SA & LCP
- Successor
- CSA
- RMQ
- static/dynamic BV
- static/dynamic succ. trees
- range min-max tree
- succ. graphs